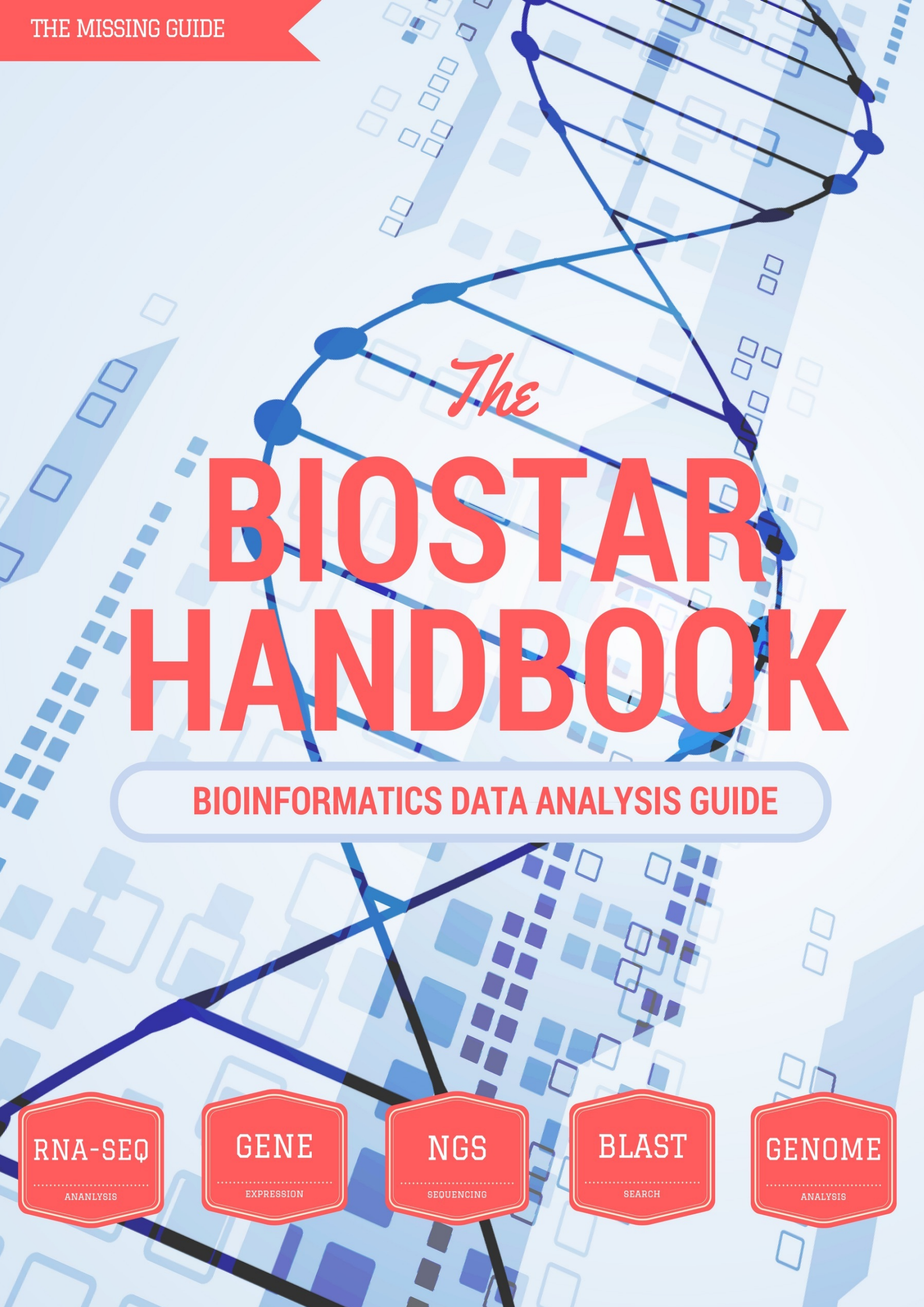
THE MISSING GUIDE

# The BIOSTAR HANDBOOK

## BIOINFORMATICS DATA ANALYSIS GUIDE



# Table of Contents

## 1. Introduction


Introduction 1.1
How to use this book 1.2
What is bioinformatics 1.3
Authors and contributors 1.4
Latest updates 1.5


## 2. Getting started


How is bioinformatics practiced 2.1
How to set up your computer 2.2
    Doctor! Doctor! Give me the news! 2.2.1
    How to troubleshoot everything 2.2.2
Essential biology concepts 2.3
Essential computing skills 2.4
How to solve it 2.5


## 3. Reproducibility


Data reproducibility 3.1
    The 2014 Ebola outbreak 3.1.1
    The 2016 Zika outbreak 3.1.2


## 4. Unix Command Line


How to learn Unix 4.1
The Unix bootcamp 4.2
Data analysis with Unix 4.3
Compressed files 4.4


## 5. Ontologies

What do words mean 5.1
Sequence ontology 5.2
Gene ontology 5.3
Understand the GO data 5.3.1
Functional analysis 5.4
Gene set enrichment 5.4.1
Using the g:Profiler 5.4.2
DAVID functional analysis 5.4.3
ErmineJ functional analysis 5.4.4
AgriGO functional analysis 5.4.5

# 6. DATA formats

Biological data sources 6.1
GenBank format 6.1.1
FASTA format 6.1.2
FASTQ format 6.1.3
Data submission 6.2

# 7. HOW TO GET DATA

Where to get data 7.1
Automating access to NCBI 7.2
Entrez Direct in practice 7.2.1
Accessing the Short Read Archive (SRA) 7.3
How much data is in the SRA 7.3.1
FASTQ manipulation 7.4
Taxonomy manipulation 7.5

# 8. Sequencing Instruments

Sequencing instruments 8.1
Illumina sequencers 8.1.1
PacBio sequencers 8.1.2
MinION sequencers 8.1.3
Data preparation 8.1.4

Sequencing data coverage 8.2

# 9. Data Quality

Visualizing data quality 9.1
Quality control of data 9.2

# 10. Sequence patterns

Pattern matching 10.1
Regular expressions 10.1.1
Sequence K-mers 10.1.2

# 11. Sequence Alignments

Sequence alignments 11.1
Global alignments 11.1.1
Local alignments 11.1.2
Misleading alignments 11.1.3
Multiple sequence alignments 11.2

# 12. BLAST

BLAST: Basic Local Alignment Search Tool 12.1
BLAST Databases 12.1.1
Using BLAST 12.1.2
BLAST Alternatives 12.1.3

# 13. Short read aligners

Short read aligners 13.1
Using the bwa aligner 13.1.1
Using the bowtie aligner 13.1.2
How to compare aligners 13.1.3

# 14. Sequence Alignment Maps

Sequence Alignment Maps (SAM) 14.1
The SAM specification 14.1.1
Filtering SAM files 14.1.2
Analyzing SAM files 14.1.3
A guide to SAM Flags 14.1.4

# 15. Advanced command line

Some programming required 15.1
Programming with Awk 15.1.1
Programming with BioAwk 15.1.2
Writing scripts 15.2
Looping over files 15.2.1
Unix oneliners 15.2.2
Automation with make 15.3

# 16. Data visualization

Genome browsers 16.1
IGV: Integrative Genomics Viewer 16.1.1
How to simulate data 16.2
How to visualize genomic variation 16.3

# 17. Genomic Variation

What is genomic variation 17.1
Representing variants in SAM 17.1.1
Online mendelian inheritance in man 17.1.2
Alignment pileups 17.2
The VCF Format 17.3
Filtering VCF files 17.3.1

# 18. Variant Calling

Variant calling 18.1
Variant calling on the Ebola data 18.1.1

Multi sample variant calling 18.1.2
What makes variant calling difficult 18.1.3
Variant normalization 18.1.4
Variant effect prediction 18.1.5

# 19. RNA-Seq Analysis

The RNA-Seq puzzle 19.1
Introduction to RNA-Seq 19.2
RNA-Seq: Terminology 19.3
RNA-Seq: Statistical analysis 19.4
RNA-Seq: How to choose the right analysis 19.5

# 20. RNA-Seq Brain Data

RNA-Seq: Griffith Test Data 20.1
Analyzing the control samples 20.1.1
RNA-Seq with alignment 20.1.2
RNA-Seq with kallisto 20.1.3
Blast from the past: The Tuxedo suite 20.1.4

# 21. RNA-Seq Zika Project

RNA-Seq: Zika Data 21.1
Zika RNA-Seq with alignment 21.1.1
Zika RNA-Seq with kallisto 21.1.2

# 23. Interval Datatypes

Interval Datatypes 22.1

# 22. Genome Assembly

What is sequence assembly 23.1
Assembly basics 23.2
GAGE: Evaluation of genome assemblies 23.3

Genome assembly terminology 23.4
How to set parameters 23.4.1
The lucky bioinformatician's guide to genome assembly 23.4.2
How to perform a genome assembly 23.4.3
Installing assembly software 23.5

# 23. ChIP-Seq Analysis

ChIP-Seq introduction 24.1
ChIP-Seq alignments 24.1.1
ChIP-Seq peak calling 24.1.2
ChIP-Seq motifs 24.1.3

# 24. Ming Tang's Guide

ChIP-Seq analysis 25.1
ChIP-Seq downstream 1 25.1.1
ChIP-Seq downstream 2 25.1.2

# 25. Metagenomics Classification

Introduction to metagenomics 26.1
How to analyze metagenomics data 26.2
Taxonomies and classification 26.3
Microbial sequence data 26.4
Classifying 16S sequences 26.5
Human Metagenome Demonstration Projects 26.5.1
Classifying whole-genome sequences 26.6
A realistic sample: Alaska Oil Reservoir 26.6.1
Tool installation 26.7

# 26. Software Installation

How to set up your computer 27.1
Setting up Mac OS 27.1.1
Setting up Linux 27.1.2

Setting up Windows 10 27.1.3
Setting up Conda 27.1.4
How to install everything 27.2
How to troubleshoot everything 27.3
How to set the profile 27.3.1
How to set the PATH 27.3.2
How to troubleshoot the PATH 27.3.3
A minimal BASH profile 27.3.4

# Software tools

Tools for accessing data sources 28.1
Installing Entrez Direct 28.1.1
Installing the SRA Toolkit 28.1.2
Installing ReadSeq 28.1.3
Installing EMBOSS 28.1.4
Installing wgsim/dwgsim 28.1.5

# Installing Ontology Tools

Tools for functional analysis 29.1
Installing ErmineJ 29.1.1
Installing GOA Tools 29.1.2

# Sequence manipulation

Tools for FASTQ/BAM files 30.1
Installing SeqKit 30.1.1
Installing SeqTk 30.1.2
Installing SamTools 30.1.3
Installing Picard 30.1.4
Installing TaxonKit 30.2
Installing Bioawk 30.3
Jim Kent's tools 30.4

# Installing QC Tools

Tools for data quality control 31.1
Installing FastQC 31.1.1
Installing Trimmomatic 31.1.2
Installing Cutadapt 31.1.3
Installing BBDuk 31.1.4

# Installing alignment tools

Alignment software 32.1
Installing EMBOSS 32.1.1
Installing BLAST 32.1.2
Installing LASTZ 32.1.3
Installing LAST 32.1.4

# Installing short read aligners

Short read alignment software 33.1
Installing BWA 33.1.1
Installing Bowtie 33.1.2

# BBMap/BBTools suite

Installing the BBMap suite 34.1
BBMap quality control 34.1.1
BBMap aligner 34.1.2
BBmap bbmerge 34.1.3
BBMap bbsplit 34.1.4
BBMap repair 34.1.5
BBMap randomreads 34.1.6
BBMap reformat 34.1.7
BBMap dedupe 34.1.8
BBMap tadpole 34.1.9

# Installing Variation calling tools

Variation calling tools 35.1

Installing Bcftools 35.1.1
Installing FreeBayes 35.1.2
Installing GATK 35.1.3
Installing snpEff 35.1.4

# RNA-Seq analysis tools

RNA-Seq analysis tools 36.1
Installing Tophat 36.1.1
Installing CuffLinks 36.1.2
Installing HiSat 36.1.3
Installing subread 36.1.4
Installing featureCounts 36.1.5
Installing kallisto 36.1.6

# Installing Interval analysis tools

Interval analysis tools 37.1

# ChIP-Seq analysis tools

MACS 38.1
GEM 38.2

# Impressum

About the author 39.1
Development timeline 39.2

# The Biostar Handbook: A Beginner's Guide to Bioinformatics

Life scientists have been working for over fifty years to decode the information contained in DNA. Until recently, this effort has been hampered by tools and methodologies insufficient to the task. Continued advances in digital information processing are now making large-scale analyses of biological information possible. The new field of bioinformatics has emerged to apply these digital tools to questions life scientists have been asking for decades. To date, the results have been promising.

The Biostar Handbook introduces readers to bioinformatics. This new scientific discipline is positioned at the intersection of biology, computer science, and statistical data analysis. It is dedicated to the digital processing of genomic information.

The Handbook has been developed, improved, and refined over the last five years in a research university setting. It is currently used as course material in an accredited PhD training program. The contents of this book have provided the analytical foundation to hundreds of students, many of whom have become full-time bioinformaticians and work at some of the most innovative companies in the world.

## Are there other formats?

The handbook is also available over the web at https://www.biostarhandbook.com. The web based book is separate of the Kindle version as it includes an online course titled **Learn Bioinformatics in 100 hours**. This course may be used to as a more structured path through this course.

## Where does the material covered in the Handbook come from?

The Handbook is used as the textbook for the **BMMB 852: Applied Bioinformatics** graduate course at Penn State. Hence it is based on the materials we've used to train Ph.D level scientists in a research university setting. We've augmented these materials with insights gathered from a website we created and maintain called Biostars: Bioinformatics Explained.

## What is a Biostar?

It's not a what. It's a who! And it could be you.

Just about every modern life science research project depends on large-scale data analysis. Moreover, it's the results of these data analyses that propel new scientific discoveries. Life science research projects thrive or wither by the insights of data analyses. Consequently, bioinformaticians are the stars of the show.

But make no mistake: it is a high-pressure, high-reward job where an individual's skills, knowledge, and determination are each needed to carry their team to success. This Handbook was carefully designed to give you all the skills and knowledge needed to become the star of the life science show, but the determination is up to you.

Science needs you. Be a Biostar!

# How do I use the Handbook?

First of all do note the search box on the top left. Once you are more familiar with site the search becomes the simplest way to find what you are looking for.

# What is currently covered in the book?

The Handbook is divided into the following sections. We cover both the foundations and their applications to realistic data analysis scenarios.

## 1. Bioinformatics foundations

- Data formats and repositories.
- Sequence alignments.
- Data visualization.
- Unix command line usage.

## 2. Bioinformatics data analysis protocols

- Genome variation and SNP calling.
- RNA-seq and gene expression analysis
- Genome Assembly (coming in 2017)
- Metagenomics (coming in 2017)
- ChIP-Seq analysis (coming in 2017)

## 3. Software tool usage

- Using short read aligners
- Using quality control tools
- Manipulating sequence data

The table of contents on the left allows you to jump to the corresponding sections.

# Should all life scientists understand how bioinformatics operates?

**Yes!**

The results of bioinformatic analyses are relevant for most areas of study inside the life sciences. Even if a scientist isn't performing the analysis themselves, they need to be familiar with how bioinformatics operates so they can accurately interpret and incorporate the findings of bioinformaticians into their work.

All scientists informing their research with bioinformatic insights should understand how it works by studying its principles, methods, and limitations—the majority of which is available for you in this Handbook.

We believe that this book is of great utility even for those that don't plan to run the analysis themselves.

## Was the book designed to be read from top to bottom?

This book follows a curricula that teaches practical data analysis for life scientists. We gradually introduce concepts and chapters tend to build on information covered before. For newcomers following top bottom might be the best approach. Yet not all chapters need to be followed in order -- readers may jump ahead to any topic of interest.

## Is there a theme to the book?

The book explains most concepts through the task of analyzing the genomic data obtained from the 2014 Ebola virus outbreak in Africa. The data representing 99 sequenced Ebola virus genomes published in the scientific article Genomic surveillance elucidates Ebola virus origin and transmission during the 2014 outbreak is used to demonstrate the data processing and data analysis tasks that a scientist might need to undertake.

## What type of computer is required?

All tools and methods presented in this book have been tested and will run on all three major operating systems: **MacOS**, **Linux** and **Windows 10**. See the Computer Setup page.

For best results Windows 10 users will need to join the Windows Insider program (a free service offered by Microsoft) that will allow them to install the newest release of "Bash Unix for Windows."

## Is there data distributed with the book?

Yes, we have a separate data site at http://data.biostarhandbook.com. Various chapters will refer to content distributed from this site.

## Who is the Handbook for?

The Biostar Handbook provides training and practical instructions for students and scientists interested in data analysis methodologies of genome-related studies. Our goal is to enable readers to perform analyses on data obtained from high throughput DNA sequencing instruments.

All of the Handbook's content is designed to be simple, brief, and geared towards practical application.

# Is bioinformatics challenging to learn?

Bioinformatics engages the distinct fields of biology, computer science, and statistical data analysis. Practitioners must navigate the various philosophies, terminologies, and research priorities of these three domains of science while keeping up with the ongoing advances of each.

Its position at the intersection of these fields might make bioinformatics more challenging than other scientific subdisciplines, but it also means that you're exploring the frontiers of scientific knowledge, and few things are more rewarding than that!

# Can I learn bioinformatics from this book?

**Yes!**

The questions and answers in the Handbook have been carefully selected to provide you with steady, progressive, accumulating levels of knowledge. Think of each question/answer pair as a small, well-defined unit of instruction that builds on the previous ones.

- Reading this book will teach you what bioinformatics is all about.
- Running the code will teach you the skills you need to perform the analyses.

# How long will it take me to learn bioinformatics from this book?

About **100** hours.

Of course, a more accurate answer depends on your background preparation, and each person's is different. Prior training in at least one of the three fields that bioinformatics builds upon (biology, computer science, and data analysis) is recommended. The time required to master all skills also depends on how you plan to use them. Solving larger and more complex data problems will require greater skills, which need more time to develop fully.

That being said, based on several years of evaluating trainees in the field, we have come to believe that an active student would be able to perform publication quality analyses after dedicating about **100** hours of study. This is what this book is really about -- to help you put those **100** hours to good use.

# What is bioinformatics?

Bioinformatics is a new, computationally-oriented Life Science domain. Its primary goal is to make sense of the information stored within living organisms. Bioinformatics relies on and combines concepts and approaches from biology, computer science, and data analysis. Bioinformaticians evaluate and define their success primarily in terms of the new insights they produce about biological processes through digitally parsing **genomic** information.

> Bioinformatics is a data science that investigates how information is *stored within* and *processed by* living organisms.

# How has bioinformatics changed?

In its early days—perhaps until the beginning of the 2000s—bioinformatics was synonymous with **sequence analysis**. Scientists typically obtained just a few DNA sequences, then analyzed them for various properties. Today, sequence analysis is still central to the work of bioinformaticians, but it has also grown well beyond it.

In the mid-2000s, the so-called *next-generation*, *high-throughput* sequencing instruments (such as the Illumina HiSeq) made it possible to measure the full genomic content of a cell in a single experimental run. With that, the quantity of data shot up immensely as scientists were able to capture a snapshot of *everything* that is DNA-related.

These new technologies have transformed bioinformatics into an entirely new field of *data science* that builds on the "classical bioinformatics" to process, investigate, and summarize massive data sets of extraordinary complexity.

# What subfields of bioinformatics exist?

DNA sequencing was initially valued for revealing the DNA content of a cell. It may come as a surprise to many, however, that the greatest promise for the future of bioinformatics might lie in other applications. In general, most bioinformatics problems fall under one of four categories:

1. **Assembly**: establishing the nucleotide composition of genomes
2. **Resequencing**: identifying mutations and variations in genomes
3. **Classification**: determining the species composition of a population of organisms
4. **Quantification**: using DNA sequencing to measure the functional characteristics of a cell

The Human Genome Project fell squarely in the **assembly** category. Since its completion, scientists have assembled the genomes of thousands of others species. The genomes of many millions of species, however, remain completely unknown.

Studies that attempt to identify changes relative to known genomes fall into the **resequencing** field of study. DNA mutations and variants may cause phenotypic changes like emerging diseases, changing fitness, different survival rates, etc. For example, there are several ongoing efforts to compile all variants

present in the human genome—these efforts would fall into the resequencing category. Thanks to the work of bioinformaticians, massive computing efforts are underway to produce clinically valuable information from the knowledge gained through resequencing.

Living micro-organisms surround us, and we coexist with them in complex collectives that can only survive by maintaining interdependent harmony. **Classifying** these mostly-unknown species of micro-organisms by their genetic material is a fast-growing subfield of bioinformatics.

Finally, and perhaps most unexpectedly, bioinformatics methods can help us better understand biological processes, like gene expressions, through **quantification**. In these protocols, the sequencing procedures are used to determine the relative abundances of various DNA fragments that were made to correlate with other biological processes

Over the decades biologists have become experts at manipulating DNA and are now able to co-opt the many naturally-occurring molecular processes to copy, translate, and reproduce DNA molecules and connect these actions to biological processes. Sequencing has opened a new window into this world, new methods and sequence manipulations are being continuously discovered. The various methods are typically named as *???-Seq* for example *RNA-Seq*, *Chip-Seq*, *RAD-Seq* to reflect on what phenomena is being captured/connected to sequencing. For example, RNA-Seq reveals the messenger RNA by turning it into DNA. Sequencing this construct allows for simultaneously measuring the expression levels of all genes of a cell.

# Is there a list of functional assays used in bioinformatics?

In the Life Sciences, an **assay** is an investigative procedure used to *assess* or measure the presence, amount, or function of some target (like a DNA fragment). Dr. Lior Pachter, professor of Mathematics at Caltech, maintains a list of "functional genomics" assay technologies on the page called Star-Seq.

All of these techniques fall into the **quantification** category. Each assay uses DNA sequencing to quantify another measure, and many are examples of connecting DNA abundances to various biological processes.

Notably, the list now contains nearly 100 technologies. Many people, us included, believe that these applications of sequencing are of greater importance and impact than identifying the base composition of genomes.

Below are some examples of the assay technologies on Dr. Pachter's list:

**dsRNA-Seq:** Qi Zheng et al., "Genome-Wide Double-Stranded RNA Sequencing Reveals the Functional Significance of Base-Paired RNAs in Arabidopsis," *PLoS Genet* 6, no. 9 (September 30, 2010): e1001141, doi:10.1371/journal.pgen.1001141.

**FRAG-Seq:** Jason G. Underwood et al., "FragSeq: Transcriptome-wide RNA Structure Probing Using High-throughput Sequencing," *Nature Methods* 7, no. 12 (December 2010): 995–1001, doi:10.1038/nmeth.1529.

**SHAPE-Seq**: (a) Julius B. Lucks et al., "Multiplexed RNA Structure Characterization with Selective 2′-hydroxyl Acylation Analyzed by Primer Extension Sequencing (SHAPE-Seq)," *Proceedings of the National Academy of Sciences* 108, no. 27 (July 5, 2011): 11063–11068, doi:10.1073/pnas.1106501108.
(b) Sharon Aviran et al., "Modeling and Automation of Sequencing-based Characterization of RNA Structure," *Proceedings of the National Academy of Sciences* (June 3, 2011), doi:10.1073/pnas.1106541108.

**PARTE-Seq:** Yue Wan et al., "Genome-wide Measurement of RNA Folding Energies," *Molecular Cell* 48, no. 2 (October 26, 2012): 169–181, doi:10.1016/j.molcel.2012.08.008.

**PARS-Seq:** Michael Kertesz et al., "Genome-wide Measurement of RNA Secondary Structure in Yeast," *Nature* 467, no. 7311 (September 2, 2010): 103–107, doi:10.1038/nature09322.

**Structure-Seq:** Yiliang Ding et al., "In Vivo Genome-wide Profiling of RNA Secondary Structure Reveals Novel Regulatory Features," *Nature* advance online publication (November 24, 2013), doi:10.1038/nature12756.

**DMS-Seq:** Silvi Rouskin et al., "Genome-wide Probing of RNA Structure Reveals Active Unfolding of mRNA Structures in Vivo," *Nature* advance online publication (December 15, 2013), doi:10.1038/nature12894.

## But what is bioinformatics, *really*?

So now that you know what bioinformatics is all about, you're probably wondering what it's like to practice it day-in-day-out as a bioinformatician. The truth is, it's not easy. Just take a look at this "Biostar Quote of the Day" from Brent Pedersen in Very Bad Things:

> I've been doing bioinformatics for about 10 years now. I used to joke with a friend of mine that most of our work was converting between file formats. We don't joke about that anymore.

Jokes aside, modern bioinformatics relies heavily on file and data processing. The data sets are large and contain complex interconnected information. A bioinformatician's job is to simplify massive datasets and search them for the information that is relevant for the given study. Essentially, bioinformatics is the art of finding the needle in the haystack.

# Is creativity required?

Bioinformatics requires a dynamic, creative approach. Protocols should be viewed as guidelines, not as rules that guarantee success. Following protocols by the letter is usually quite counterproductive. At best, doing so leads to sub-optimal outcomes; at worst, it can produce misinformation that spells the end of a research project.

Living organisms operate in immense complexity. Bioinformaticians need to recognize this complexity, respond dynamically to variations, and understand when methods and protocols are not suited to a data set. The myriad complexities and challenges of venturing at the frontiers of scientific knowledge always require creativity, sensitivity, and imagination. Bioinformatics is no exception.

Unfortunately, the misconception that bioinformatics is a *procedural* skill that anyone can quickly add to their toolkit rather than a scientific domain in its own right can lead some people to underestimate the value of a bioinformatician's individual contributions to the success of a project.

As observed in Core services: Reward bioinformaticians, Nature, (2015),

> Biological data will continue to pile up unless those who analyze it are recognized as creative collaborators in need of career paths.

Bioinformatics requires multiple skill sets, extensive practice, and familiarity with multiple analytical frameworks. Proper training, a solid foundation and an in-depth understanding of concepts are required of anyone who wishes to develop the particular creativity needed to succeed in this field.

This need for creativity and the necessity for a bioinformatician to think "outside the box" is what this Handbook aims to teach. We don't just want to list instructions: "do this, do that". We want to help you establish that robust and reliable foundation that will allow you to be creative when (not if) that time comes.

# What are common characteristics of bioinformatics projects?

Most bioinformatics projects start out with a "standardized" plan, like the ones you'll find in this Handbook. But these plans are never set in stone. Depending on the types and features of observations and results of analyses, additional tasks will inevitably deviate from the original plan to account for variances observed in the data. Frequently, the studies need substantive customization.

As the authors of Core services: Reward bioinformaticians, Nature, (2015) have observed of their own projects,

> No project was identical, and we were surprised at how common one-off requests were. There were a few routine procedures that many people wanted, such as finding genes expressed in a disease. But 79% of techniques applied to fewer than 20% of the projects. In other words, most researchers came to the bioinformatics core seeking customized analysis, not a standardized package.

In summary, this question is difficult to answer because there isn't a "typical" bioinformatics project. It is quite common for projects to deviate from the standardized workflow.

# Authors and Collaborators

The Handbook's main author and editor is Dr. Istvan Albert.

- Read more about the author.
- Personal website: https:/www.ialbert.me
- Email: istvan.albert@gmail.com

# Collaborators and Contributors

- Aswathy Sebastian, MS, Bioinformatics Analyst, Bioinformatics Consulting Center, Pennsylvania State University, USA
- Reka Albert, PhD, Professor of Physics and Biology, Department of Physics, Pennsylvania State University, USA
- Jeremy Leipzig, MS, Senior Data Integration Analyst, The Children's Hospital of Philadelphia, USA
- Hemant Kelkar, PhD, Research Associate Professor, Department of Genetics and Center for Bioinformatics, University of North Carolina, USA
- Ming Tang, PhD, Computational biologist in Genomic Medicine Department, MD Anderson Cancer Center, USA
- Ram Srinivasan, MS, Bioinformatician, Icahn School of Medicine and Mount Sinai, New York, NY, USA
- Wei Shen, PhD student, Third Military Medical University, China.
- Wouter De Coster , PhD Student, University of Antwerp, Belgium
- Madelaine Gogol, BS, Programmer Analyst, Stowers Institute, Kansas City, MO, USA

# Contribute to the Handbook!

Join our effort to build a comprehensive and up to date compendium for genomic data analysis.

It is a simple process that works through GitHub via simple, text based, Markdown files. The only permission we ask from authors are the rights to publish their content under our own terms (see below). Please note that this right cannot be revoked later by the author.

Authors retain re-publishing rights for the material that they are the principal author of and may re-distribute the content that they have created for this book under other terms of their choosing.

# What are the licensing terms?

The Handbook's content is copyrighted material owned by Biostar Genomics LLC. Republishing any part of the Handbook is permitted only on a limited basis and must follow the terms of the Fair Use policy unless other, prior agreements have been made.

The only exception to this rule is that authors and contributors to the book retain re-publishing rights for the material that they are the principal (primary) author of and may re-distribute that content under other terms of their choosing. We define principal author as typically done in academia as the person that performed the majority of the work and is primarily responsible for its content.

## What can I reuse from the book?

If you have an account on this site that means that you have been provided with a license to access, apply, modify and reuse information from the book as it applies to your own work. You may not share your login information with others or distribute the book to others.

## Acknowledgements


- The Unix Bootcamp section is based on the Command-line Bootcamp by Keith Bradnam.
- The word cloud was created by Guillaume Fillion from the abstracts of Bioinformatics from 2014 (for a total around 1000 articles).

# Biostar Handbook Updates

This page list the updates to the book.

Please note that you have to be logged in to access the book.

## August 8th, 2017

Download: Biostar-Handbook-August-2017.zip

A chapter on Metagenomics has been added

1. Introduction to metagenomics
2. How to analyze metagenomics data
3. Taxonomies and classification
4. Microbial sequence data
5. Classifying 16S sequences
6. Human Metagenome Demonstration Projects
7. Classifying whole-genome sequences
8. A realistic sample: Alaska Oil Reservoir
9. Tool installation

## July 4th, 2017

Download: Biostar-Handbook-July-2017.zip

Extensive editing of various chapters. Special thanks to Madelaine Gogol and Paige M. Miller

## April 6, 2017

Download: Biostar-Handbook-April-2017.zip

A chapter on Chip-Seq analysis has been added:

1. ChIP-Seq introduction
2. ChIP-Seq alignments
3. ChIP-Seq peak calling
4. ChIP-Seq motifs
5. ChIP-Seq analysis
6. ChIP-Seq downstream 1
7. ChIP-Seq downstream 2

## January 28, 2017

Download: Biostar-Handbook-January-2017.zip

Added the following sections:

1. What is sequence assembly
2. Assembly basics
3. GAGE: Evaluation of genome assemblies
4. Genome assembly terminology
5. How to set parameters
6. The lucky bioinformatician's guide to genome assembly
7. How to perform a genome assembly
8. Installing assembly software

In addition to the new chapter editors have performed an expansive proofreading and editing the prior content. Special thanks to Madelaine Gogol and Ram Srinivasan for their effort.

Added a better web search interface that highlights the matches.

# December 14, 2016

Archive contains PDF, MOBI and EPUB formats.

Download: biostar-handbook-14-12-2016.zip

- Added the RNA-Seq Puzzle.
- Started section on Interval Data Types.
- Added more information on dealing with anxiety in data analysis.
- Proofreading and many other stylistic changes.

# December 5, 2016

The first version of the book is released.

# How is bioinformatics practiced?

Bioinformatics requires a broad skill set. The diverse tasks undertaken by bioinformaticians can be roughly divided into three tiers:

**1. Data management**.

Data management requires accessing, combining, converting, manipulating, storing, and annotating data. It requires routine data quality checks, summarizing large amounts of information, and automating existing methods.

**2. Primary data analysis**.

Analysis of the data requires running alignments, variation callers, and RNA-Seq quantification, as well as finding lists of genes. Strategically, the bioinformatician must anticipate potential pitfalls of planned analyses, find alternatives, and adapt and customize the methods to the data.

**3. Data interpretation**.

Data management and analysis are meaningless without accurate and insightful interpretation. Bioinformaticians discover or support biological hypotheses via the results of their primary analyses, and so they must be able to interpret their findings in the context of ongoing scientific discourse.

# What is the recommended computer for bioinformatics?

In our experience, the most productive setup for a bioinformatician is using a Mac OS based computer to develop and test the methods and then using a high-performance Linux workstation—or cluster—to execute these pipelines on the data. With the release of Windows 10 Creator Edition (accessible for users that are part of the Windows Insider program), a fully functional Linux Bash shell can be installed and run on Windows. See the Computer Setup page for more details.

# How much computing power do we need?

Bioinformatics methodologies are improving at a rapid pace. A regular workstation is often all you need to analyze data from a typical, high-volume sequencing run (hundreds of millions of measurements). For example, you could handle most RNA-Seq data analysis needed in a given day by using only the `hisat2` aligner on a standard iMac computer (32GB RAM and 8 cores used in parallel). Larger analyses like genome assembly, however, typically require larger amounts of memory than are available on a standard desktop computer.

As a rule, low-quality data (contamination, incorrect sample prep, etc) takes substantially longer to analyze than high-quality data.

# Does learning bioinformatics need massive computing power?

**No!**

Mastering bioinformatics requires nothing more than a standard, UNIX-ready laptop. You can learn bioinformatics using only a $100 Chromebook.

Of course, the computational needs for applying the bioinformatics methods on particular problems will depend on the amount and scale of data being analyzed. A $100 Chromebook has the tools one needs but probably doesn't have the compute power, storage or memory to run analyses of the massive datasets that may need to be processed.

You might be surprised, however, just how much can be done on just a Chromebook!

Remember this; you don't need an expensive or powerful system to become one of the best bioinformaticians on the planet! You can learn, practice and gain a profound understanding of this entire field on any simple, Unix capable computing device.

# What about the cloud?

Cloud computing is becoming an increasingly common platform for bioinformatics, in particular for efforts that allow bioinformaticians to "bring the tools to the data" - applying your algorithms to massive sequence databases. Cloud services such as Amazon Web Services also enable anyone to host web applications and services on powerful, accessible machines while only paying for the compute time they use.

Running a cloud service involves learning about object stores, virtual private clouds, file systems, and security. Building and maintaining a cloud instance requires you become something like a part-time systems administrator, which can distract you from learning bioinformatics. In later chapters, we will introduce both cloud computing and cloud-based stacks - commonly referred to as software-as-a-service (SaaS), platform-as-a-service (PaaS, and infrastructure-as-a-service (IaaS). These offerings, combined with *containerized* applications and analyses, have become valued for their potential to provide a scalable platform for reproducible research.

If you are comfortable with these concepts, a cloud-based server can be a companion to this guide, but learning bioinformatics *does not require* the cloud.

# Do I need to know Unix to do bioinformatics?

Bioinformatics has primarily been developed via freely available tools written on the Unix platform. The vast majority of new advances are published with software written for Unix-based operating systems.

It's unlikely you'll be in a position for long-term career advancement as a bioinformatician without a basic understanding of the command line. Starting with 2016, even the Windows platform has begun to offer the so-called "Bash for Windows" application that allows the Windows 10 operating system to run almost all of Unix-specific software.

The good news is that using Unix is not that complicated. While somewhat unforgiving, the Unix philosophy is logical and precise. In our experience, even people with no prior computational background can become confident Unix users in a matter of weeks.

# Do I need to learn a programming language?

Yes — an introductory level of programming ability (in any programming language) will be necessary.

Fortunately, programming languages function by similar underlying logic — even when they seem quite different on the surface. While symbols, syntax, and procedures may differ among programming languages, an introductory level of programming ability in any language is necessary to understand the thought processes required for computational analysis.

We do cover programming concepts in this book. See the section titled Some programming required and the corresponding subsections for a primer.

# Are there alternatives to using Unix?

Alternatives to Unix are limited but do exist. They fall into two categories:

## 1. Software that provides a web interface to the command line tools

These software systems run the same command line tools that one could install into a Unix-based system, but the interfaces to these tools are graphical and provide better information "discoverability."

1. Galaxy (open source)
2. GenePattern (open source)
3. BaseSpace (commercial)
4. DNA Nexus (commercial)
5. Seven Bridges Genomics (commercial)

## 2. Systems that offer custom implementations of bioinformatics methods.

These can be standalone software that runs on your local system — they are compatible with web-based applications. They often run open-source tools via a GUI.

1. CLC Genomics Workbench
2. Partek
3. Golden Helix
4. SoftGenetics
5. DNA Star
6. Geneious

# What is Bioconductor?

Bioconductor is an open-source software project for the analysis and comprehension of genomic data. Bioconductor is based primarily on the R statistical programming language

Users of Bioconductor typically write scripts using R that make calls to functions defined in the Bioconductor library. For example, a Bioconductor-based script that runs an RNA-Seq data analysis could look like this:

```
biocLite("DESeq")
library(DESeq)
count = read.table("stdin", header=TRUE, row.names=1 )
cond1 = c("control", "control", "control")
cond2 = c("treatment", "treatment", "treatment")
conds = factor(c(cond1, cond2))
cdata = newCountDataSet(count, conds)
esize = estimateSizeFactors(cdata)
edisp = estimateDispersions(esize)
rdata = nbinomTest(edisp, "control", "treatment")
write.table(rdata, file="", sep="\t", row.name=FALSE, quote=FALSE)
```

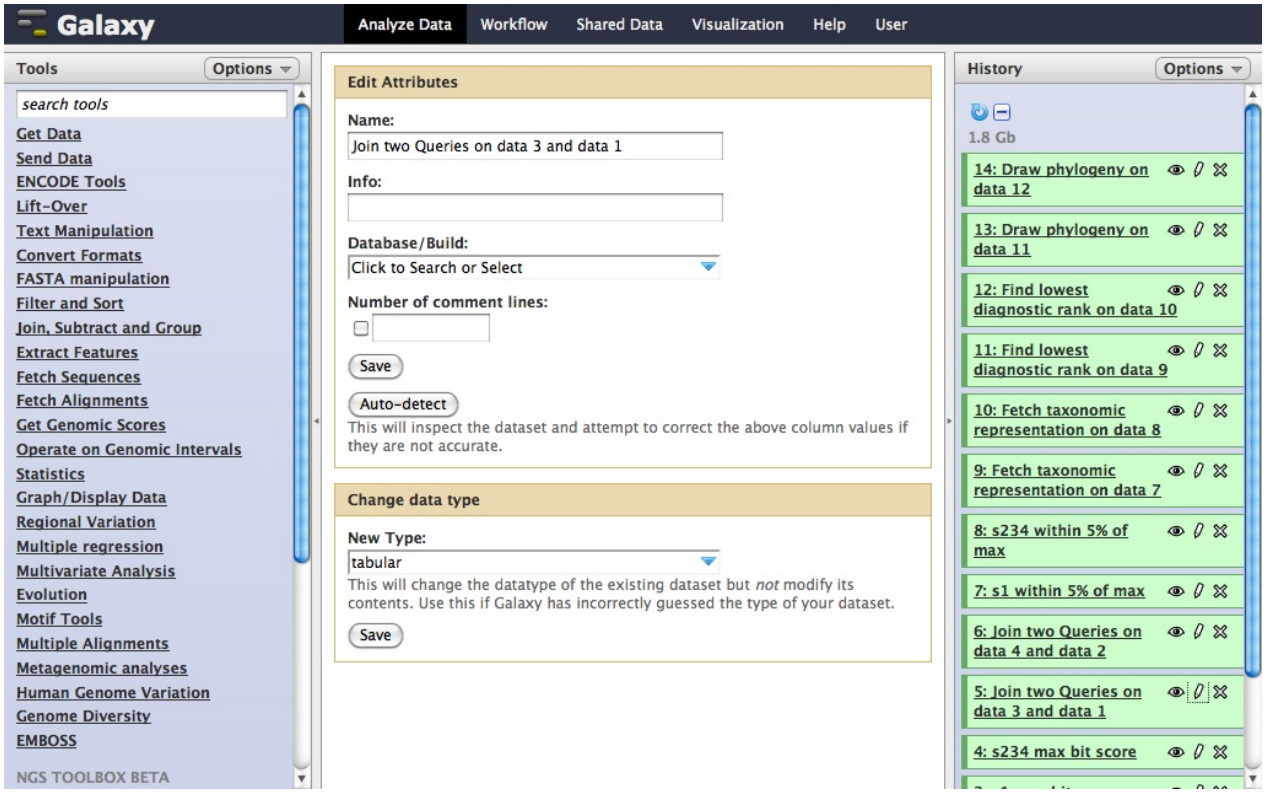
# What is Galaxy?

Galaxy is a web-based scientific workflow and data analysis platform that aims to make computational biology accessible to research scientists with no computer programming experience.

Galaxy can generate and present a web-based user interface to command line tools, making them available to those unable to run the tools themselves.

**Important:** Galaxy is not a data analysis tool! It is an **interface** and a **platform** that provides computational capacity and runs **other** bioinformatics software on your behalf. Galaxy makes it easy to launch tools and manage data -- it is the user's responsibility to understand the underlying concepts and use these tools correctly.

Galaxy uses a three-panel system where the left panel lists the tools, the middle panel describes the tools, and the right panel displays user data




# What is BaseSpace?

BaseSpace is a cloud-based genomics analysis and storage platform provided by Illumina that directly integrates with all Illumina sequencers.

The system can operate only with sequencing data produced by Illumina and only when the data is in the formats generated by the sequencers.

# Are commercial bioinformatics software packages expensive?

From an individual, consumer-level perspective, the cost of commercial bioinformatics software is high — typically measured in thousands of dollars. The price tag means that few people could or would buy them directly.

For larger organizations, the software costs may pay off quickly in productivity gains as long as the software is chosen carefully to match specific research needs.

Most academicians and open-source developers treat commercial software with some distrust. In our opinion, this distrust may be unwarranted. Many commercial tools cater to specific needs and need to be operated by individuals with diverse levels of training. Also, the goals and resource allocations in academia are typically very different than those of private organizations.

# Should I freelance as a bioinformatician?

In our opinion, the answer is **no** (unless the terms are generous).

Successful bioinformatics analysis requires more than skill alone: the information content of the data is a crucial second piece. The world's best bioinformatician won't be able to extract information that is not present in the data and may have trouble demonstrating that they have done their best.

Our experiences line up with the observations in Core services: Reward bioinformaticians, Nature, (2015) that state that analyses always end up being substantially more complicated than anticipated.

It is unexpectedly difficult to estimate accurately the effort that a project requires.

# What do bioinformaticians look like?

Good-looking of course!

Here are some participants at the NSF-funded "Analyzing Next Generations Sequencing Data" workshop at Michigan State University (2014). Graduate students, postdocs, researchers, faculty, were all represented.

# How to set up your computer

Practicing bioinformatics will require the installation of software tools that may are distributed in various formats:

1. Binary "executable" code that can be run directly on the target computer.
2. Source code that needs to be "compiled" to create a "binary" program.
3. Programs that require the presence of another programming language: `java`, `python`, or `perl`.

# How do I set up my computer?

Follow each of the steps below. We may instruct you to visit other pages but after you do visit those make sure to come back to this page.

Ensure that you followed each of the 4 steps and the corresponding subsections: **1.1**, **1.2** etc.

1. **Prepare your computer.**
2. **Install conda.**
3. **Activate bioconda.**
4. **Install the software.**

# 1.0 How do I prepare my computer?

Before installing software you have to prepare you computer for the tasks ahead.

We have are separate instructions for MacOS, Linux and Windows computers. Follow only the instructions for your platform.

There will be multiple steps that need to be followed in addition the processes may take some time, possibly hours. Make sure to complete the system setup before venturing forth:

- Setting up a Mac OS computer
- Setting up a Windows 10 computer
- Setting up a Linux computer

Once your system is set up properly continue on.

# 1.1 How do I initialize my terminal configuration?

You will need a few lines in your setup files. Open a Terminal and run the following below (run them only once!):

```
curl http://data.biostarhandbook.com/install/bash_profile.txt >> ~/.bash_profile
curl http://data.biostarhandbook.com/install/bashrc.txt >> ~/.bashrc
```

These commands will update your so called "shell" (see later) to behave more consistently. If you want the details read the page on Bash Profile.

The new settings won't apply right away to your current terminal window. To have them applied you will need to either start a new terminal window or type:

```
source ~/.bash_profile
```

# 1.2 What are environments?

Modern software packages are interdependent. Each program may rely on the presence of one or more other software. In turn, each required software may also require another program to be present - and so on - down the rabbit hole we go,

In addition software A may only inter-operate properly with a certain version of sofware B. For that reason there may be conflicts between the different requirements. These conflicts can be solved by creating isolated installations where only the compatible software are installed together.

The isolated installations are called *environments*.

To set up and manage our environment we will use the conda installer on all platforms. On MacOS we may also use of the `homebrew` package manager in addition to `conda`.

# 2.0 How do I install `conda` ?

We have a separate page that deal with conda in

- How to install conda

Once `conda` is installed as above you can create an environment called `bioinfo` with:

```
conda create -y --name bioinfo python=3
```

To activate the `bioinfo` environment (this needs to be done in each new window you open) type :

```
source activate bioinfo
```

You can list all existing environments with:

```
conda info --envs
```

You can learn more about managing environments on the conda webpage

# 3.0 How do I activate `bioconda` ?

Bioconda is a distribution of bioinformatics software that can be enabled in the Conda package manager. This operation only needs to be done once:

```
conda config --add channels r
conda config --add channels conda-forge
conda config --add channels bioconda
```

# 4.0 How do I install everything?

We do offer a line of code that installs everything:

```
# Remember to run this once per window.
source activate bioinfo

# Install most software tools used in this book.
curl http://data.biostarhandbook.com/install/conda.txt | xargs conda install -y
```

The command above should take a few minutes to complete.

There is a chapter with details on each software separately but you would only need to visit if you are having problems with one particular tool.

# How do I upgrade every tool?

Tools may have updated versions released. Once a year (or more frequently even) we recommend that you run the following command:

```
curl http://data.biostarhandbook.com/install/conda.txt | xargs conda upgrade -y
```

# 4.1 How do I check that my setup is correct?

We have created a script that verifies that your settings are correct. Throughout the book, if any one of our instructions causes an error, your first choice will be to see what the doctor says :-) In our experience when the "doctor" is happy all of our examples work.

- Install the doctor

Once you installed the `doctor.py` run it:

```
~/bin/doctor.py
```

The doctor will report errors that you might have.

# 4.2 What if I have a "chronic case of Entrez Direct"?

There is recurring problem with Entrez Direct that some people experience. When running `efetch` or other tools these may fail with a lengthy error message, that among many other lines contain:

```
501 Protocol scheme 'https' is not supported (LWP::Protocol::https not installed
```

It is hard to track down the source of this error or under what circumstances do people see it. We believe that it has to do with competing versions of the same Perl library being present. The following usually helps:

```
~/bin/doctor.py --fixme
```

Will print the following commands:

```
# A chronic case of Entrez Direct?

mkdir -p ~/src
curl ftp://ftp.ncbi.nlm.nih.gov/entrez/entrezdirect/edirect.zip > ~/src/edirect.zip
unzip -o ~/src/edirect.zip  -d ~/src
echo 'export PATH=~/src/edirect:$PATH' >> ~/.bashrc
source  ~/.bashrc
```

Execute the commands above to install your local version of Entrez Direct. The problem usually goes away after this.

# Do the automatically installed tools always work?

Alas no. Bioinformatics software can become outdated. Tools packagers ( `conda` or `brew` ) are not always aware or able to integrate these changes. As a result tools installed via these may suddenly not work until the package maintainer update their formulas.

In those cases, visit the source code instruction page in the book and install manually from the source.

- How to Install Everything

# What if I still have setup problems?

The following pages collect a number of tips on dealing with various problems:

1. How to set the profile
2. How to set the PATH
3. How to troubleshoot the PATH
4. How to troubleshoot everything

# How should I set up my file structure?

Many people have strong feelings on how a filesystem should be set up when using Unix command line for scientific projects. Here are some opinions:

- A Quick Guide to Organizing Computational Biology Projects
- A research workflow based on GitHub
- Directory layout advice

In our view, the most important concept is that of simplicity. We typically create the following directories:

- Links to binaries go into `~/bin`.
- Reference genomes go into `~/refs`.
- Source code for tools go into `~/src`.

You can come up with appropriate names for other folders. We like to use `work`, `project` etc.

Just keep the names short and simple because you will have to enter or read them a lot. We often need to type full paths to various files and in those cases it really helps if the paths look like

- `~/refs/hg19.fa`

What you definitely don't want are paths that look like like:

- `/My Bioinformatics Work/Documents and Folders/Unfinished Projects/2016/factor Binding/todo3/mygenome/b19/human.fa`

You will feel the urge to make your names very long and descriptive. You will want to do this because you'll be afraid that you will forget what you did. While that is a valid concern, long file names are not a good approach to handle this information complexity.

What you need is a proper track record of what you did. We'll teach you how to do that. You'll write scripts where each step is documented. So worry not about short file names.

# What is a good text editor?

You will need a proper **TEXT** editor. And no, Microsoft Word is not a text editor. Using Word to edit text will eventually cause (devious) errors. It's best if you learn to use a proper text editor from the start. Essential features that your editor needs to have:

- The editor needs to have the ability to show `TAB` characters versus `SPACE` characters. It does not always have to show these, just when you turn the feature on. Mixing up `TAB` and `SPACE` is a very common error. Note: copy-pasting from a web page may turn `TAB`s into `SPACE`s. Hence you need the ability to see the difference if necessary.
- The editor should be able to show line numbers. This lets you identify the line where an error might be reported.
- The editor has to be able to switch line endings. Line endings have variants: Unix, Windows or Mac OS 9 endings (since Mac OS X the Mav is using Unix line endings). All your files should have always have Unix line endings. Your editor needs to have the ability to convert line endings to Unix format.

Recommendations:

The choice of an editor is somewhat personal preference. The following options are a good starting point:

- NotePad++ on Windows
- Sublime Text 2 is a simple and small text editor that is available for all platforms
- Komodo Edit requires fewer resources than other editors so it works well on laptops.

We use PyCharm for both text editing and programming - it is a bit too heavyweight for text editing alone.

# What if I still have problems?

We have a section titled:

- How to troubleshoot everything

# How do I know that I have installed everything correctly?

A frustrating situation occurs when you can't figure out why an example does not work. You may wonder is your system set up properly? It helps a lot if you can rule out software installation problems. This is when you can call on the `doctor.py` .

First of all you have to do an initial computer setup as instructed before.

# Do I need to also install `R` ?

Only some of the later chapters use `R` but the `doctor.py` checks for the right version of it. We recommend to have the `R` programming available. Since you are already installing programs why not get this done as well. Installing `R` is fairly straigthforward, download and doubleclick the installer from

- https://www.r-project.org/

# How do I call the `doctor.py` ?

We have created a script that verifies that your settings are correct. Throughout the book, if any one of our instructions causes an error, your first choice will be to see what the doctor says :-) In our experience when the "doctor" is happy all of our examples work.

Download and run our doctor.py script from a terminal:

```
mkdir -p ~/bin
curl http://data.biostarhandbook.com/install/doctor.py > ~/bin/doctor.py
chmod +x ~/bin/doctor.py
```

Run it from a terminal and 'doctor' will tell you what (if anything) is out of order:

```
~/bin/doctor.py
```

Later, at any time, when you have a problem run the doctor.py command again to check your settings. This is what you want to see:

```
# Doctor! Doctor! Give me the news.
# Checking 13 symptoms...
# You are doing well!
```

## Can the `doctor.py` fix problems?

The doctor.py can also help your system "get better":

```
doctor.py --fixme
```

This will provide you with advice the doctor thinks might help. If you are adventurous you could do a:

```
doctor.py --fixme | bash
```

to apply the advice directly. But do ensure that you understand the output of the `--fixme` command first.

## What is the next step?

Continue of with the software setup

# How to Troubleshoot Everything

When you get started with something radically new, everything seems to break all the time.

This is because it can be exceedingly difficult to recognize small differences, errors, or inconsistencies when you are unfamiliar with a process.

## Have you followed all installation steps?

Double check that you have followed each step correctly:

1. Set up the computer
2. Installed conda and enabled bioconda
3. Created and activated the environment:

```
source activate bioinfo
```

In the vast majority of cases this will solve the problems you have. If not read on.

## What else can cause problems?

> Tiny errors and/or inconsistencies may have big effects.

1. Make sure there is correct spacing between the commands.
2. Make sure there are no additional, invisible characters such as extra tabs, extra spaces, spaces in the place of tabs in the files that you use.
3. Make sure the capitalization is what it should be.
4. Read the error messages carefully. Sometimes it helps googling the exact message.
5. Error messages are always useful, but sometimes hard to interpret and the important tidbit may be buried among the other information.

## What other solutions should I be looking at?

A few more pages with advice:

- How to set the profile
- How to set the PATH
- How to troubleshoot the PATH
- A minimal BASH profile

## What is the next step?

Continue of with the computer setup

# Biology basics for bioinformaticians

Biology is a domain of the Life Sciences, which include, but are not limited to, organic chemistry, ecology, botany, zoology, physiology, and so on. Biology seeks to understand the structures, functions, origins, interactions, and taxonomies of living organisms.

As a science, biology is still relatively immature. It is eclectic, encompassing several scientific domains that are each insufficiently understood and described.

Organisms are immensely complex and always changing. For centuries, we haven't had precise enough tools to measure, describe, or understand the full extent of their complexity. Digital technologies are changing this. Bioinformatics is at the frontier of these changes, and its potential contributions to Biology and the Life Sciences more broadly are quite exciting. As bioinformatics and other innovative methodologies advance, we expect Life Sciences to mature and develop rich, accurate vocabularies and models to understand and describe living organisms.

For these reasons, current concepts and definitions in the Life Sciences are still just approximations. While they can be sufficiently accurate in some contexts, they may be woefully inadequate in others. In what almost sounds like a "scientific fight club", the "rules" of bioinformatics are as follows:

1. There are no "universal" rules.
2. Every seemingly fundamental paradigm has one or more exceptions.
3. The validity of a bioinformatics method depends on unknown characteristics of the data.
4. Biology is always more complicated than you think, *even after taking this rule into account*.

Below, we have attempted to describe the biological concepts that we deem important to understand the types of information encoded in data. Each of our definitions is short and terse, and we recommend additional self-guided study of each concept of interest.

# What is DNA?

**DNA** stands for Deoxyribo Nucleic Acid.

It is a macromolecule (a molecule constructed of smaller molecules) that carries the genetic instructions required for the development, functioning and reproduction of all known living organisms. In eukaryotic organisms (like animals, plants, and fungi), DNA is present in the nucleus of each cell. In prokaryotic organisms (single-celled organisms like bacteria and mitochondria), DNA is present in the cell's cytoplasm.

## What is DNA composed of?

DNA is made up of two strands of smaller molecules coiled around each other in a double-helix structure. If you uncoiled DNA, you could imagine it looking somewhat like a ladder. Ladders have two important parts: the poles on either side and the rungs that you climb. The "poles" of DNA are made of alternating molecules of deoxyribose (sugar) and phosphate. While they provide the structure, it's the "rungs" of the DNA that are most important for bioinformatics.

To understand the "rungs" of the DNA, imagine a ladder split in half down the middle so that each half is a pole with a bunch of half-rungs sticking out. In DNA, these "half-rungs" are a molecule called nucleotides. Genetic information is encoded into DNA by the order, or sequence, in which these nucleotides occur. The "ladder" of DNA is held together by the bonds between each "half-rung."

## What are nucleotides?

Nucleotides are the building blocks of nucleic acids (DNA and RNA–we'll get to that one later on). In DNA, there are four types of nucleotide: Adenine, Cytosine, Guanine, and Thymine. Because the order in which they occur encodes the information biologists try to understand, we refer to them by their first letter, `A`, `C`, `G` and `T,` respectively.

```
A   Adenine
G   Guanine
C   Cytosine
T   Thymine
```

Back to the ladder analogy. "Nucleotide" is the *type* of molecule that makes up each half-rung of our ladder. Because they function as the units that encode genetic information, each letter is also called a base.

For an example of how we represent the sequence of DNA bases using these letters, if we took the DNA of the *Bacillus anthracis* bacteria that causes Anthrax disease, unfolded the double-helix into a ladder, and then split the ladder in two, the top half (the forward strand–we'll get to that in a bit) would be written like this:

```
ATATTTTTTCTTGTTTTTTATATCCACAAACTCTTTT
```

## What are base pairs?

When you put both sides of the ladder back together, each rung is made by the bond between two bases. When they are bonded, we call them a base pair (or sometimes "bp").

When it comes to base pairs, it's important to remember that Adenine only ever bonds with Thymine, and Guanine with Cytosine. In the ladder analogy, this means that each rung will be denoted as "A-T," "T-A," "G-C," or "C-G."

Certain groups of nucleotides that share some common attribute may be designated by so called *ambiguity* codes, for example, `W` stands for `A` or `T`:

```
Y     Pyrimidine (C or T)
R     Purine (A or G)
W     Weak (A or T)
S     Strong (G or C)
K     Keto (T or G)
M     Amino (C or A)
D     A, G, T (not C - remember as after C)
V     A, C, G (not T - remember as after T/U - We'll get to "U" soon)
H     A, C, T (not G - remember as after G)
B     C, G, T (not A - remember as after A)
N     Any base
-     Gap
```

## What are DNA strands?

Remember when we split the DNA "ladder" in half down the middle so that each side was a pole with half-rungs sticking out? These two halves are called strands. To distinguish the two strands, scientists label one the forward strand and the second, the reverse strand. Above, we gave the example of the forward strand in the *Bacillus anthracis* bacteria. Here's what it looks like paired with its reverse strand:

```
forward --> ATATTTTTTCTTGTTTTTTATATCCACAAACTCTTTT
            ||||||||||||||||||||||||||||||||||||
            TATAAAAAAGAACAAAAAATATAGGTGTTTGAGAAAA <-- reverse
```

The lines that connect the bases on either side denote a basepair relationship.

"Forward" and "reverse" are just labels. The choice of labeling is arbitrary and does not depend on any inherent property of the DNA. The forward strand is not "special." Scientists decide which to call "forward" and which to call "reverse" when they first analyze the DNA of an organism. Even though the decision is arbitrary, it's important to maintain consistency with that decision for the sake of clear communication.

The forward and reverse strands may also be denoted with different terms. For example, in some datasets you may find them labeled as `+` and `-`. They might also be called *top* and *bottom* strands, or even *Watson* and *Crick* strands.

In our opinion, these variances are needlessly confusing. Please avoid referring to strands with any other terms than *forward* and *reverse*.

# Is there a directionality of DNA?

Yes, there is a directionality to the DNA determined by the polarity of the molecules. This direction runs in opposite ways for each strand. Typically, we indicate this polarity with arrows:

```
------>
ATATTTTTTCTTGTTTTTTATATCCACAAACTCTTTT
||||||||||||||||||||||||||||||||||||
TATAAAAAAGAACAAAAAATATAGGTGTTTGAGAAAA
                            <------
```

Most biological mechanisms (but not all) take place on a single strand of the DNA and in the direction of the arrow. Hence, sequences of the DNA above will be "seen" by the biochemical machinery as either:

```
ATATTTTTTCTTGTTTTTTATATCCACAAACTCTTTT
```

or as:

```
AAAAGAGTTTGTGGATATAAAAAACAAGAAAAAATAT
```

This last sequence is called the *reverse complement* of the first and is formed by reversing the order of the letters then swapping `A` for `T` and swapping `C` for `G` (and vice-versa).

Hence a DNA sequence `AAACT` may need to be considered:

- in reverse, `TCAAA`
- as a complement, `TTTGA`
- as a reverse-complement, `AGTTT`

# What is sense/antisense?

When a process occurs in the expected direction, its directionality may be called *sense*; if it is going against the normal direction, its directionality may be called *anti-sense*.

It is very important not to collate the concepts of *forward*/*reverse* with *sense*/*anti-sense*, as these concepts are completely unrelated. The *sense/anti-sense* is relative to a sequence's direction; the sequence, in turn, may come from a *forward* or *reverse* strand.

# What is DNA sequencing?

DNA sequencing is the "catch-all" terminology that describes all the processes for identifying the composition of a DNA macromolecule.

The results of a DNA sequencing process are data files stored in an unprocessed format, typically either `FASTA`, `FASTQ`, or unaligned `BAM` (called `uBAM`) files. Most published papers also store their data in repositories that can be downloaded for reanalysis.

# What gets sequenced?

It is essential to note that instruments do not directly sequence the DNA in its original form. Sequencing requires a laboratory process that transforms the original DNA into a so-called "sequencing library" - an artificial construct based on the original DNA. The process of creating this sequencing library introduces a wide variety of limitations and artificial properties into the results. Also, the method of building the sequencing library will also limit the information that can be learned about the original DNA molecule.

Most (perhaps all) life scientists use the term "sequencing" over-generously and are often under the assumption that it produces more precise information than what it can deliver.

# What is a genome?

A genome is all of an organism's DNA sequence. Each cell typically contains a copy of the entire genome. More accurately, each cell has one or more nearly identical copies of the genome. Replication is the process of duplicating the genome when a cell divides.

While, due to complementarity, the number of `A` and `T` nucleotides and the `C` and `G` nucleotides is equal, the relative ratios of `AT` pairs vs `CG` pairs may be very different. Some genomes may contain more `AT` pairs while others may contain more `CG` pairs.

# What is a genome's purpose?

The genome contains the information that makes the functioning of an organism possible. In cellular organisms, for example, the genome has regions that contain the instructions for making proteins. These are typically called "coding regions".

The genome may also have regions used to produce molecules other than proteins, as well as regions that regulate the rates by which other processes take place.

All genomes are subject to evolutionary principles. Hence, some (or even substantial) parts of a healthy genome may be non-functional and may no longer serve any obvious purpose. Some percent of a genome may consist of copies of various kinds of interspersed, repeat sequences. At some point, biologists labeled these regions "junk DNA", but the term has since become a lightning rod of controversy.

Identifying non-functional regions has turned out to be more difficult, and quite more controversial than biologists originally anticipated. The so-called C Value Paradox captures the observation that the size of a genome does not directly determine the genome's complexity.

# How big is a genome?

Functional genomes range from as short as the 300 base pairs of the prions that cause the mad-cow disease to as long as the 150 billion base pairs of *Paris japonica*, a rare, perennial, star-like flower from Japan. It is common to refer to genome sizes in terms of kilo-bases (thousands), mega-bases (millions), and giga-bases (billions).

Here are some other genome sizes:

- Ebola virus genome: 18 thousand basepairs (18Kb)
- E.coli bacteria genome: 4 million basepairs (4Mb)
- Baker's yeast fungus genome: 12 million basepairs (12Mb)
- Fruit fly genome: 120 million basepairs (120Mb)
- Human genome: 3 billion basepairs (3Gb)
- Some salamander species: 120 billion basepairs (120 Gb)

# What is RNA?

Whereas DNA is the blueprint, RNA is a smaller interval of this plan translated into a new kind of molecule. This molecule is similar to DNA in that it differs only owing to a few chemical modifications that change its properties relative to DNA. For example, in RNA, the base `T` (Thymine) is replaced by the nucleotide `U` (Uracil).

RNA is a polymeric, single-stranded molecule that often performs a function and is believed to exist transiently. Unlike DNA, there are many classes of RNA: `mRNA`, `tRNA`, `rRNA,` and many others. DNA is continuously present in the cell, whereas RNA degrades quickly over time (minutes).

# How does a genome function?

The genome has numerous functions, most of which are too complicated (or insufficiently studied) to describe accurately. Instead, scientists employ a "biological narrative" to describe genomic functions and processes.

The "narrative story" is made very simple by design, often accompanied by a simplified illustration of events: step 1 followed by step 2 followed by step 3, etc. While this helps initially, it also runs of risk of explaining concepts in an oversimplified manner. To give you an example of how this looks in practice, here is a "biological narrative" of what is perhaps the most studied genomic phenomenon: *primary mRNA transcription* in eukaryotic cells.

> "The cell begins by transcribing a "gene" (see later) into an RNA molecule. After transcription, pieces of the RNA are cut out and discarded in a process called "splicing." Each discarded piece is referred to as an *intron*. Each piece between consecutive introns is known as an *exon*, and the RNA molecule with the introns removed is known as messenger RNA, or *mRNA*.
>
> Part of the cell's method for identifying introns is the presence of `GT` and `AG,` the so-called "splice signals" that usually occur as the first and last dinucleotides of an intron. But the mere presence of these dinucleotides is not sufficient for splicing to occur. Perhaps 35% of human genes are alternatively spliced, meaning that under different circumstances, various combinations of exons are selected."

Remember, again, that the above explanation is a simplistic description of a far more complex phenomenon. Any single detail of it opens a series of yet unexplainable questions: why is it that only a small subset of `GT` `AG` pairs cause splicing and how come the polymerase that initiates a splicing event at a `GT` knows that there will be a `AG` later? Or perhaps it does not know and it is all random - but that would have other implications. You see how the rabbit hole only gets deeper and deeper.

# What is a protein?

A protein is a three-dimensional macromolecule built from a series of so-called "amino acid" molecules that can form a 3D structure. There are 20 kinds of amino acids that can form a protein; these are labeled as letters in the "alphabet" of protein sequences, where each letter is an amino acid. The "alphabetical order of the amino acids alphabet is:

```
ARNDCEQGHILKMFPSTWYZ
```

Where `A` stands for Alanine, `R` for Arginine, `N` for Asparagine and so on

Proteins can be described by their sequence alone. But in our current state of understanding, the sequence itself is typically insufficient to fully determine the protein's 3D structure or function.

Whereas DNA and mRNA usually carry *information*, proteins are the actual, physical building blocks of life. Every living organism is built out of proteins and functions via the interactions of proteins that are being continuously produced.

A short series (less than 40) of amino acids without a well-defined 3D structure is called a polypeptide (peptides).

# How are proteins made?

The process of reading *DNA* and creating *mRNA* out of it is called *transcription*. Then, in eukaryotes, the *mRNA* is transported out of the nucleus (to the cell's cytoplasm), where it is converted to a protein sequence in a process called *translation*. Multiple proteins (even hundreds) may be translated from a single mRNA molecule.

- transcription: DNA --> mRNA
- translation: mRNA --> Protein

To perform the translation, the mRNA is partitioned into units of three consecutive letters, each called a *codon*. A *codon* is then translated via a translation table into an *amino acid*. A protein is a sequence of amino acids.

There is a genetic code for translating a codon into an amino acid. For example, the codon `TCA` (or `UGA` if we describe it as an RNA sequence) codes for `S`, the amino acid *Serine*. There are 64 combinations of codons but only 20 amino acids. Some codons code for the same amino acid, this is called the codon degeneracy. For example:

```
CGU, CGC, CGA, CGG, AGA, AGG --> Arginine (Arg/R)
```

Most (but not all) amino acids are encoded by more than one codon. A few codons, in most cases `TAA`, `TAG` and `TGA` are the so called *stop codons*; translation stops when one of them is reached.

The translation process begins with the so-called start codon `ATG` that corresponds to `M` (the *Methionine* amino acid).

For example the subunit B of the Cholera Toxin protein complex responsible for the massive, watery diarrhea characteristic of cholera infection has the following sequence:

```
MIKLKFGVFFTVLLSSAYAHGTPQNITDLCAEYHNTQIYTLNDKIFSYTESLAGKREMAI
ITFKNGAIFQVEVPGSQHIDSQKKAIERMKDTLRIAYLTEAKVEKLCVWNNKTPHAIAAI
SMAN
```

It starts with an `M`, the Methionine, which as we just stated is also the start codon. You may think all protein sequences should start with `M`.

But wait, remember how every rule has exceptions? That applies here as well. While the overwhelming majority of proteins start with an `M` it is possible for this `M` to get cut off during later processes.

# What is an ORF?

An `ORF`, or *open reading frame*, is a sequence of at least, say, 100 consecutive codons without a stop codon.

It is important also to note that while there is a "standard" translation table, some organisms may use slight variations of it. These are called genetic codes. When translating from genome to protein, the use of the correct genetic code is essential.

# What is a gene?

In our experience, the word gene is one of the most misused words in the entirety of biology. It is usually interpreted quite differently by different scientists. Some think of genes as the DNA that codes for proteins; others think genes may include upstream elements, and so on. The "official" definition of the term gene in the Sequence Ontology is

> A region (or regions) that includes all of the sequence elements necessary to encode a functional transcript. A gene may include regulatory regions, transcribed regions and other functional sequence regions.

# Do genomes have other features?

Genomic regions may have a wide variety of functions. Here are a few often-studied types:

## Untranslated regions

The region of the mRNA before the start codon (or the corresponding genomic region) is called the `5' UTR` (5 prime UTR), or untranslated region; the portion from the stop codon to the start of the poly-A tail is the `3' UTR` (three prime UTR).

## Promoter regions

The genomic region just before the `5' UTR` may contain patterns of nucleotides, called the promoter, that are used to position the molecular machinery that performs the transcription to RNA. For about 60% of human genes, the promoter occurs in a `CpG` island: a region of DNA where `C` and `G` are far more frequent than `A` and `T`. Other patterns in the DNA tell the cell when (how frequently and in what tissues) to transcribe the gene; that is, they regulate transcription. A pattern that increases the frequency of transcription operations is an `enhancer`, while one that decreases the rate is a `silencer`.

## CpG islands

CpG islands are regions of DNA where a `C` (cytosine) nucleotide is followed by a `G` guanine nucleotide in the linear sequence of bases along its 5' -> 3' direction. `CpG` is shorthand for 5'—C—phosphate—G—3', that is, cytosine and guanine separated a phosphate; phosphate links any two nucleosides together in DNA. Promoter regions may contain CpG islands.

Cytosines in CpG dinucleotides can be methylated. In turn, methylated cytosines within a gene may change the gene's expression, a mechanism that is part of a larger field of science studying gene regulation, called epigenetics.

Below is a section of a CpG island from chromosome 1 of the human genome. It has 30 CpGs and GC% is 77.14.

```
>gi|568815597:36306860-36307069 Homo sapiens chromosome 1, GRCh38.p7 Primary Assembly
CGGGGCTCCGGAGAGGCGCGGAGGCCGCGCTGTGCGCGCCGCCGAGGTGAGCGCAAGGGCGGGGACGGGC
GCCGGTGGGCGGGTGCACGGAGCCAGTGCGACCCCGGCGTCTCCGGCTCTTAGTGACGGGCGCGGCTCTG
GGCGGGACCTCGGGGCCGCCCTGCGGTCTGTGATTGGTTCTCGAGTGCAATGCTCCGCCCTGGGGCGGGG
```

We obtained the above via:

```
efetch -db=nuccore -id=NC_000001.11 -format=fasta -seq_start=36306860 -seq_stop=36307069
```

This tool will be covered in later sections.

### Enhancers

Enhancers help a gene turn on and regulate where, when, and how much the gene should be expressed for the cell to function properly. One enhancer can regulate many genes, and one gene can have many enhancers. Since DNA is folded and crumpled up in a cell, an enhancer can be far away from a gene it regulates in terms of distance measured along the DNA strand, but physically very close to the gene promoter. The best way to identify a sequence as an enhancer is to experimentally disrupt it and observe gene expression changes. Since this is cumbersome in practice, we use proxies like histone modifications (chemical decorations on the structural backbone upon which DNA is coiled) to identify enhancers. Silencers work similarly -- they can be far from a gene, many-to-many relationships, hard to find -- but cause a gene to be less, rather than more, expressed.

## What is homology?

Two regions of DNA that evolved from the same sequence (through processes of duplication of genomic regions and separation of two species) are `homologous`, or `homologs` of one another.

More specifically, regions in the genomes of two species that are descended from the same area in a common ancestor's genome are `orthologs`. These regions are said to be `orthologous`.

On the other hand, `paralogous` sequences or `paralogs` were separated by duplication of a genomic region within the same genome.

Standard pet peeve of many scientists:

> Homology is not a synonym of sequence similarity!
>
> Homologous sequences are usually similar to one another, but similarity of sequences does not indicate homology.

# Essential computing skills

Bioinformatics requires analyzing big, complex datasets. The recommended approach is learning the Unix command line and working with a computer that offers Unix integration.

## What type of computing skills are needed?

The required computing skills are relatively simple, logical, and straightforward, but since these may be a radically new way for you to interact with the computer, you might need some time to develop these skills. When it comes to bioinformatics, the learning curve starts a little steeper, but it flattens out fairly quickly.

## Where can I learn command line computing skills?

This Handbook dedicates an entire chapter, with several sections, to helping you acquire the necessary computing skills. You may access these sections from the table of contents on the left side (scroll down) or from these links:

1. How to learn Unix
2. The Unix bootcamp
3. Data analysis with Unix
4. Compressed files

More advanced command line usage is demonstrated in

1. Writing scripts
2. Unix oneliners

In addition there are excellent online tutorials, some free, others at a price, each with a different take and philosophy on how to teach the concepts. We recommend that you try out a few and stick with the one that seems to work the best for you.

- CodeAcademy: Learn the command line
- Software Carpentry: The Unix shell
- Command line bootcamp
- Unix and Perl Primer for Biologists
- Learn Enough Command Line to Be Dangerous
- The Command Line Crash Course
- Learn Bash in Y minutes

## Does bioinformatics require specific software?

Bioinformatics analyses require the installation and use of various software packages. We have collected all tool installation related instructions on the following pages:

1. How to install everything
2. How to troubleshoot everything

# How to solve it

This section has been inspired by the book titled How to solve it by George Pólya describing methods of problem solving. His advice can be summarized via:

1. First, you have to understand the problem.
2. After understanding, then make a plan.
3. Carry out the plan.
4. Look back on your work. How could it be better?

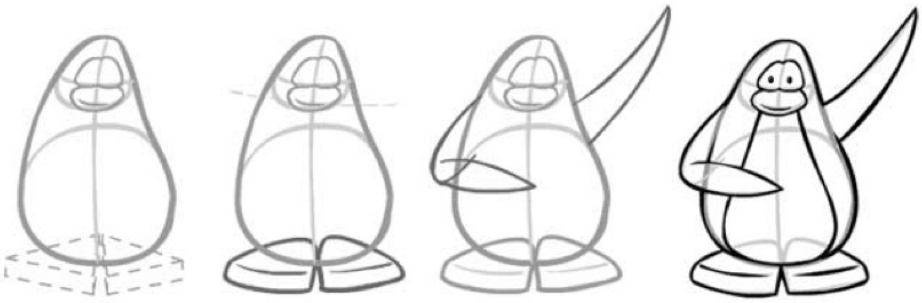
The advice is general and timeless, but how does it apply to Bioinformatics?

# What is a holistic data analysis?

In our opinion one of the most important characteristics of the analysis is to think about it "holistically" -- considering all steps that need to be performed.

The best analogy is drawing - usually artists will start with a rough draft. In time, they will add more detail and accuracy to their work. The same applies to any data analysis - do not get bogged down with the initial details - make it first work on a rough scale - so that you see all the moving components that will make up your final result. Only when you have all the parts together will you have a firm understanding of the strengths and weaknesses of the entire process.

Imagine and visualize what your result will need to look like and work toward that goal.



# How do I perform a holistic analysis?

The simplest approach is to isolate a small section of your data and build out your analysis around that. Don't start with processing hundreds of gigabytes of data against genomes with billions of bases.

Start small and stay in control. You will be able to build pipelines that run in minutes (or even seconds) and you are continually aware of what is taking place, what problems are cropping up, and what the results look like. You'll be able to identify very early if something is amiss. Your analysis will run quickly even on a small laptop.

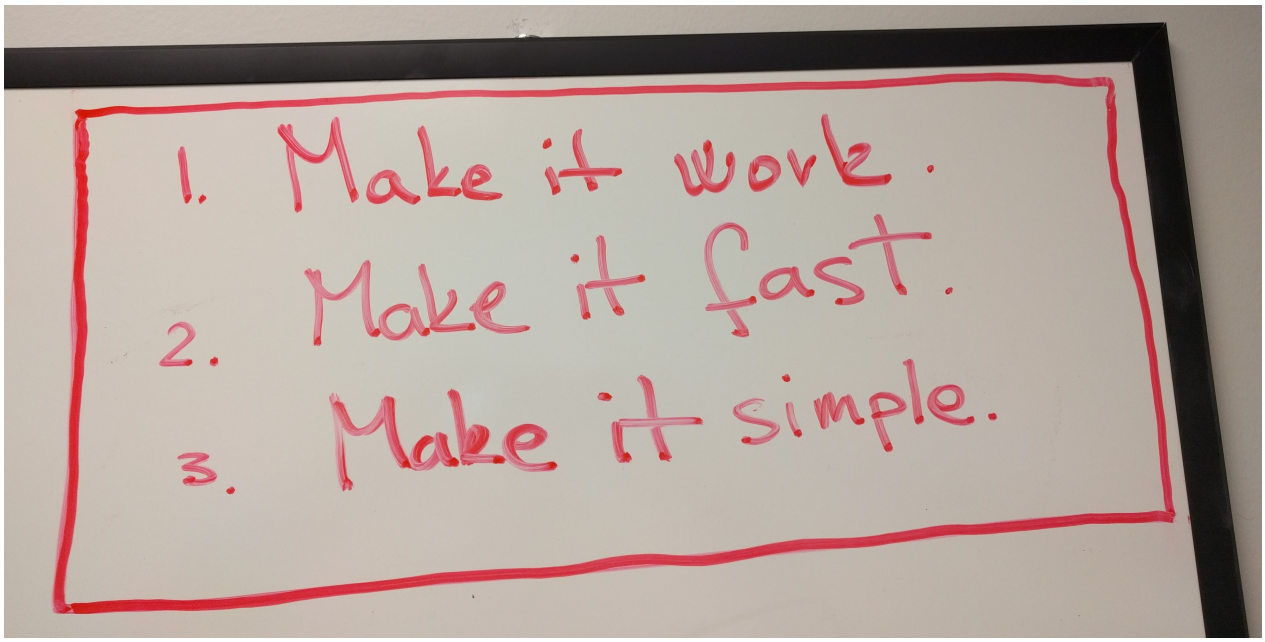
Do not try to optimize a process too early. Build your pipeline to be flexible and robust that you can rerun with other techniques if you need to. You will only really understand the problems that you need to solve when you get near the end.

How to select these subsets? Here are some examples we employ in this book:

1. Download just a subset of the data. For instance in all our examples that use `fastq-dump` we limit data to values between 10,000 or 1 million reads. You can tweak these numbers to find that balance where the results are quick and allow you to understand and iterate over the results.
2. Build random samples out of your data. Tools such as `seqtk`, `seqkit` and others will let you randomly select a much smaller subset of data.
3. Use a smaller segment of the target genome. If you study large genomes, use just a single chromosome of it. Build your pipeline and set everything up say over just human chromosome 22. Then swap out the reference file when you are ready to do it all.
4. Generate data with known properties. We have a chapter on simulating data - and it is, in fact, the best way to understand the properties and limits of the various processes that you encounter. When you analyze data with known properties, you can validate and understand how your methods work.

# What are the rules of a bioinformatics analysis?

In our center, written on the whiteboard is the motto and credo of how we believe progress is made in this field:




1. Make it work.
2. Make it fast.
3. Make it simple.

# Why is fast so important?

On the one hand doing it fast sort of make sense. Who wouldn't want to do things fast?

But there is more to it than just getting home by six. It is about the way we think about problems in the subconscious. When a process takes three days, we instinctively think up questions where we factor in the time it takes to get the answer. We try to ask more "difficult" questions that are "worth" the time. If a solution takes just two minutes, we can maintain mental focus on the problem, and we will investigate it with a very different approach.

So make your data small, analyze it quickly and ask quick and easy questions - but lots of them. You will get much further.

## How to deal with anxiety and stress?

It may feel strange to talk about anxiety in a data analysis book. But make no mistake about it - it is something many of us experience.

Bioinformaticians often work alone because other members of the group do not have sufficient expertise to contribute. In that case the data analyst alone needs to make dozens if not hundreds of decisions and they alone are responsible for applying these correctly.

Adding insult to injury most tools can be cryptic and terse, with features that you may know nothing about. When you save the commands that you type for posterity (and yes, you should), anyone at any time can review and criticise these, and any error you make will be visible and apparent. Being the sole decision maker of this analysis puts the burden on you alone.

With time the added pressure of this responsibility can wear you down psychologically. After a while you won't remember why you made certain decisions, and you might have doubts and fears. When many months later the paper is accepted and the publication comes around you may ask yourself: *Did I really do it right? What if now, that many people will look at it, they will instantly find and point out that trivial and glaring mistake that renders everything invalid?*.

One of the saddest, public, episodes of science are described in the article Doubts about Johns Hopkins research have gone unanswered, scientist says a story that follows what we believe a situation where the added pressure of being to sole data analyst of a major paper have led to a tragic outcome. Notably a recent article points out that Ph.D. students face significant mental health challenges.

# Doubts about Johns Hopkins research have gone unanswered, scientist says

For me the stories above are eerily familiar - I wish I had known about them sooner. You see, earlier in my career I've struggled with similar issues. I've rarely enjoyed the publication of what later turned out to be some of my most cited papers - in fact I've dreaded the publication date - I saw it as the date when everyone finds out that I made a mistake, it is all wrong and that I am incompetent.

Today I know better - and I tell you the same. If you do your due diligence, follow good practices, keep it simple, keep it straight, everything will be ok, it does not matter that much anyway.

And you know what? If you have messed up because you invoked the commands as

```
samtools view -f 357
```

instead of the "correct"

```
samtools view -f 257
```

it is not your fault. It is the stupidity of the tool that is to be blamed. A well-designed tool should never allow you to accidentally make errors of this type. As you will see later mistyping a single digit will radically alter the entire meaning of the command, yet there is no way to recognize that from just looking at it. The number that you see there needs to be interpreted as the binary bits encoded into that integer. Gee, thank you `samtools` (sarcasm)!

Psychology says that cursing helps: `samtools` you suck and worse, the `SAM` format is amusingly ill-designed. `GATK`? that tool is bonkers ... I won't go on, but the list is practically endless. Bioinformatics, unfortunately, represents all too well the disconnect of the Ivory Tower from reality, where the tool makers are "too smart for their own good" and create software that is fragile, brittle, difficult to use, easy to misuse, and frankly often just plain wrong.

So once you you got that off your chest you can go back to enjoy data analysis - and if you do mess up... well, rest assured many, many others did too. You may have been alone in the lab, but when it comes to people that published at least one incorrect analysis, you aren't.

It is a full house, actually.

# Data reproducibility

In this section we jump ahead to show you where we are going with this book and what to expect. You don't have to follow the commands that we demonstrate - but come back to these chapters once you know more.

The reproducibility of scientific analyses is (or rather should be) a cornerstone of science. The scientific method itself depends on other scientists' ability to verify statements made by others.

In reality, the reproducibility process is fraught with many pitfalls, and solutions are few and far between. The problems we face are in part due to rapid and radical transformations in different scientific domains, and this is particularly the case for the Life Sciences.

## What does reproducible research mean?

In what might be the most ironic of situations - scientists have a hard time agreeing what the words "reproducible research" mean.

The very definition of "*reproducible research*" itself seems not to be reproducible! There is a surprising amount of uncertainty and confusion even what the word ought to mean.

Here we present the ideas that scientists came up with in the order of least demands to most demands that they put on authors:

1. A brief description of the decisions that were made during the analysis.
2. A detailed description of the commands that were used to generate the report.
3. A list of all instructions and a snapshot of the intermediate data that the commands produced.
4. Publishing the analysis as a single document that one can easily re-run.
5. All the above with the added information of the exact computer setup, operating system configuration and software version that was used to generate the analysis.
6. All the above packaged into a single "virtual" machine that emulates the same setup the authors had at their disposal that can be re-run.

The problem with these latter demands is that they focus too much on blindly following another protocol instead of re-doing it in a different way. In our opinion only a different analysis, one that comes up with the same outcomes, would strengthen the case for a new discovery.

## Is it possible to re-analyze data?

As you will see it is usually a lot easier to re-analyze data from the beginning with your own methods than to reproduce and retrace the same steps that were published in a scientific paper.

In the majority of cases, you'd want to know how the data has been analyzed not to rerun the exact same commands, but to assess the strengths and weaknesses of the analysis and to identify any possible errors or inconsistencies in the process.

# Which research papers will be re-analyzed in this book?

As the book evolves we plan to cover a different scientific publication for each domain of application. The current list includes the following:

1. Genomic surveillance elucidates Ebola virus origin and transmission during the 2014 outbreak
2. Zika Virus Targets Human Cortical Neural Precursors and Attenuates Their Growth
3. GAGE: A critical evaluation of genome assemblies and assembly algorithms
4. Genome-Wide Mapping of Binding Sites Reveals Multiple Biological Functions of the Transcription Factor Cst6p in Saccharomyces cerevisiae.
5. Necrotizing enterocolitis a devastating complication of prematurity to microbial composition of the gut

# What is the best strategy to reproduce results?

As a bioinformatician, you may often need to reproduce published results. We must warn everyone that doing so by following the descriptions and explanations found within research papers is usually a tedious and frustrating process.

A typical scientific paper spends far more time trying to convince readers that its findings are significant rather than explaining the steps of how that analysis took place. This is a normal consequence of how scientific publications are judged. Novelty is the most important requirement.

But there is another and more devious complicating factor. The so-called *"scientific narrative"* is simplified and misleading - it depicts the bioinformatics analysis as a linear chain of decisions: *"first we did this, then we did that."* In this way it paints a picture that does not correspond to reality.

Most of the time the authors explored several, perhaps dozens of alternatives, ran into many dead ends, backtracked and tried their analyses in several different ways. After quite a bit of work, they've finally honed in on what they believe to be the correct approach for making the discovery. But within the scientific paper the long and branching chains of decisions and actions are described as straightforward choices, and the failed attempts are never mentioned.

Your aim should always be to re-analyze the data in a simpler way. "Re-doing it faster and better" is the approach that we champion in this book and it is the one that has served us well. With time methods improve and the algorithms change, then all scientific observations should be discoverable in different ways. Sticking to one protocol because that's what one always used before and that is what "my group is using" is not always a good choice. It may still be the optimal choice but you should ensure that you understand the alternatives. Once you know what type of insights are in your data there is almost always a simpler and more efficient method to find them.

# Are the challenges of data reproducibility recognized?

Most organizations funding science recognize that reproducible data and results are a growing problem in bioinformatics, and other areas of science. They implement what they believe to be a "carrot and stick" approach, policies of offering a combination of rewards and punishment to induce behavior that should

result in more reproducibility.

In our opinion these policies, at least those that we are aware of, rely mostly on "stick" and have less of the "carrot" component. In summary you should be aware that the problem is widely recognized even though the current mitigation efforts appear to be inadequate.

# Re-analyze: Genomic surveillance elucidates Ebola virus origin

## Which sections of the book re-analyze this data?

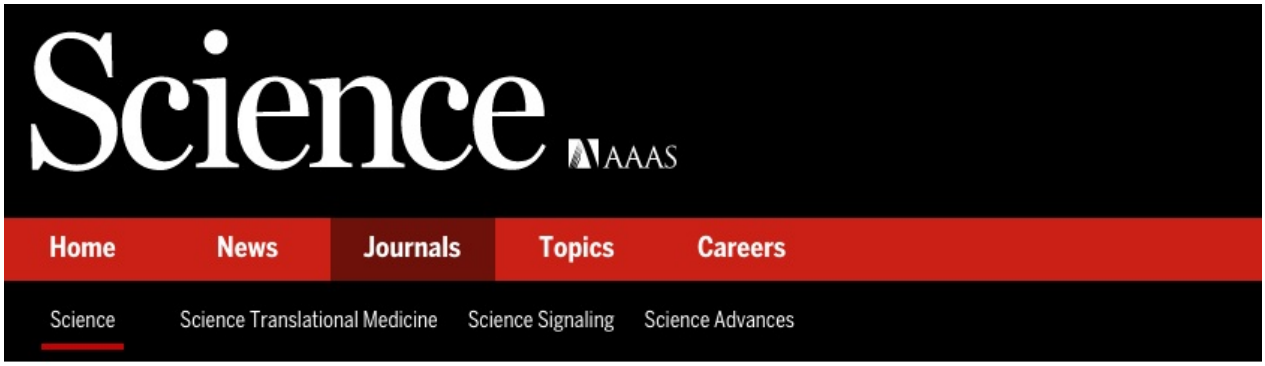
1. Using BLAST to analyze the Ebola data
2. Variant calling on the Ebola data
3. Multi sample variant calling

## Which scientific publication are we discussing?

We are interested in accessing the data for the study titled

- Genomic surveillance elucidates Ebola virus origin and transmission during the 2014 outbreak

published in 2014 in the Science research journal.




SHARE

REPORT

### Genomic surveillance elucidates Ebola virus origin and transmission during the 2014 outbreak

Stephen K. Gire[1,2,*], Augustine Goba[3,*,†], Kristian G. Andersen[1,2,*,†], Rachel S. G. Sealfon[2,4,*], Daniel J. Park[2,*], Lansana Kanneh[3], Simbirie Jalloh[3], Mambu Momoh[3,5], Mohamed Fullah[3,5,‡], Gytis Dudas[6], Shirlee Wohl[1,2,7], Lina M. Moses[8], Nathan L. Yozwiak[1,2], Sarah Winnicki[1,2], Christian B. Matranga[2], Christine M. Malboeuf[2], James Qu[2], Adrianne D. Gladden[2], Stephen F. Schaffner[1,2], Xiao Yang[2], Pan-Pan Jiang[1,2], Mahan Nekoui[1,2], Andres Colubri[1], Moinya Ruth Coomber[3], Mbalu Fonnie[3,‡], Alex Moigboi[3,‡], Michael Gbakie[3], Fatima K. Kamara[3], Veronica Tucker[3], Edwin Konuwa[3], Sidiki Saffa[3], Josephine Sellu[3], Abdul Azziz Jalloh[3], Alice Kovoma[3,‡], James Koninga[3], Ibrahim Mustapha[3], Kandeh Kargbo[3], Momoh Foday[3], Mohamed Yillah[3], Franklyn Kanneh[3], Willie Robert[3], James L. B. Massally[3], Sinéad B. Chapman[2], James Bochicchio[2], Cheryl Murphy[2], Chad Nusbaum[2], Sarah Young[2], Bruce W. Birren[2], Donald S. Grant[3], John S. Scheiffelin[8], Eric S. Lander[2,7,9], Christian Happi[10], Sahr M. Gevao[11], Andreas Gnirke[2,§], Andrew Rambaut[6,12,13,§], Robert F. Garry[8,§], S. Humarr Khan[3,†§], Pardis C. Sabeti[1,2,†§]

You could read this paper all the way through and not be able to find the information on where the data is located. If you don't believe us, see it for yourself.

# Is it possible to access the data for this analysis?

We were able to find the data mentioned only because we knew beforehand that data deposited in the Short Read Archive is typically designated with a bioproject code that starts with `PRJN`. A text search for this pattern leads to a tiny entry at the very end of the paper, past the references, in the acknowledgments - way out of sight – in the section that we could call a scientific "*no-mans land*." It is there where we finally found what many might consider one of the most important deliverables of this project:

> Sequence data are available at NCBI (NCBI BioGroup: **PRJNA257197**).

Make a note of this number `PRJNA257197`. It will be our primary means to get all the data for this project, not by clicking around on websites but in a systematic, automated and repeatable way.

# Where can we find out more about this dataset?

But now that we have the id number `PRJNA257197`, we're all set, right? Well, you still need to know where to look for it. Searching for this number at the NCBI website, we can find a Bioproject titled:

- Zaire ebolavirus sample sequencing from the 2014 outbreak in Sierra Leone, West Africa.

At the time of writing, the project summary included `891` experiments that produced `249` nucleotide and `2240` protein sequences.

***Project Data:***

| Resource Name | Number of Links |
|---|---|
| SEQUENCE DATA | |
| Nucleotide (Genomic RNA) | 249 |
| SRA Experiments | 891 |
| Protein Sequences | 2240 |
| PUBLICATIONS | |
| PubMed | 1 |
| PMC | 1 |
| OTHER DATASETS | |
| BioSample | 716 |

Once you are on the Bioproject site, it is no less shocking just how little information is disseminated on what exactly has been deposited or how these results were obtained.

- What are these `249` nucleotide and `2240` protein sequences?
- How reliable is the information?
- How are they different from what was known before?

- What methods and practices were used to derive them?

The rather lengthy Supplementary Information does not help much, either. This 29-page document has an extensive list of tools and techniques, but it severely lacks specificity. It contains statements such as:

> EBOV reads were extracted from the demultiplexed Fastq files using Lastal against a custom-made database containing all full-length EBOV genomes

This description is inadequate for reproducing the results: it only poses more questions.

- What is this custom made database?
- Why isn't this custom database deposited here as well?
- Which full-length EBOV genomes were included?
- What does it mean to be extracted with `Lastal` ?

Note another missing element here: who do we contact if we have a question? While papers do have corresponding authors, those are almost never the ones that know the intricate details of the analysis.

# How do we download results for this paper?

We will learn in later chapters of automated ways to get the sequencing data for this project but what about the results themselves? Can we get those in a flexible format? Here again, the standards vary widely. In this case the approach is borderline ridiculous. You need to visit the publisher's website, navigate to an arbitrary location and then individually download files with obscure file names, without knowing exactly what these files contain.

You can now get these to look at from the command line (via ridiculously named URLs) because we went through the tedious process of copy-pasting each and collected them for you below. Stop and think about how absurd these URLs look and how inefficient is to distribute data this way:

```
curl -OkL http://science.sciencemag.org/highwire/filestream/595764/field_highwire_adjunct_file
s/4/1259657_table_s1.zip
curl -OkL http://science.sciencemag.org/highwire/filestream/595764/field_highwire_adjunct_file
s/3/1259657_file_s4.zip
curl -OkL http://science.sciencemag.org/highwire/filestream/595764/field_highwire_adjunct_file
s/0/1259657_file_s1.zip
curl -OkL http://science.sciencemag.org/highwire/filestream/595764/field_highwire_adjunct_file
s/7/1259657_table_s4.zip
curl -OkL http://science.sciencemag.org/highwire/filestream/595764/field_highwire_adjunct_file
s/5/1259657_table_s2.zip
curl -OkL http://science.sciencemag.org/highwire/filestream/595764/field_highwire_adjunct_file
s/4/1259657_table_s1.zip
```

Unzip a file and see what it contains - note how uninformative the name of the file is:

```
unzip 1259657_table_s1.zip
```

As you will see later you can unzip all files in one command (and don't worry if you don't get this yet, we'll talk about this alter):

```
ls -1 *.zip | xargs -n 1 unzip
```

Some of the files are in PDF format. One such PDF contains nothing more than a table with four rows and five columns! Since it is distributed as a PDF and not as plain text or an Excel sheet that we could export into other formats we cannot automatically process the information contained in this file without manually transcribing it into a machine readable format!

Other files are in Excel but follow an arbitrary ad-hoc format unsuited for automated processing. Again those will need to be reformatted in a different way to compare to standard formats.

We think we made our point - today data distribution is an ad-hoc and superficially supervised process that demands potential users do substantial extra work that should not be required.

# Re-analyze: Zika Virus Targets Human Cortical Neural Precursors

## Which sections of the book re-analyze this data?

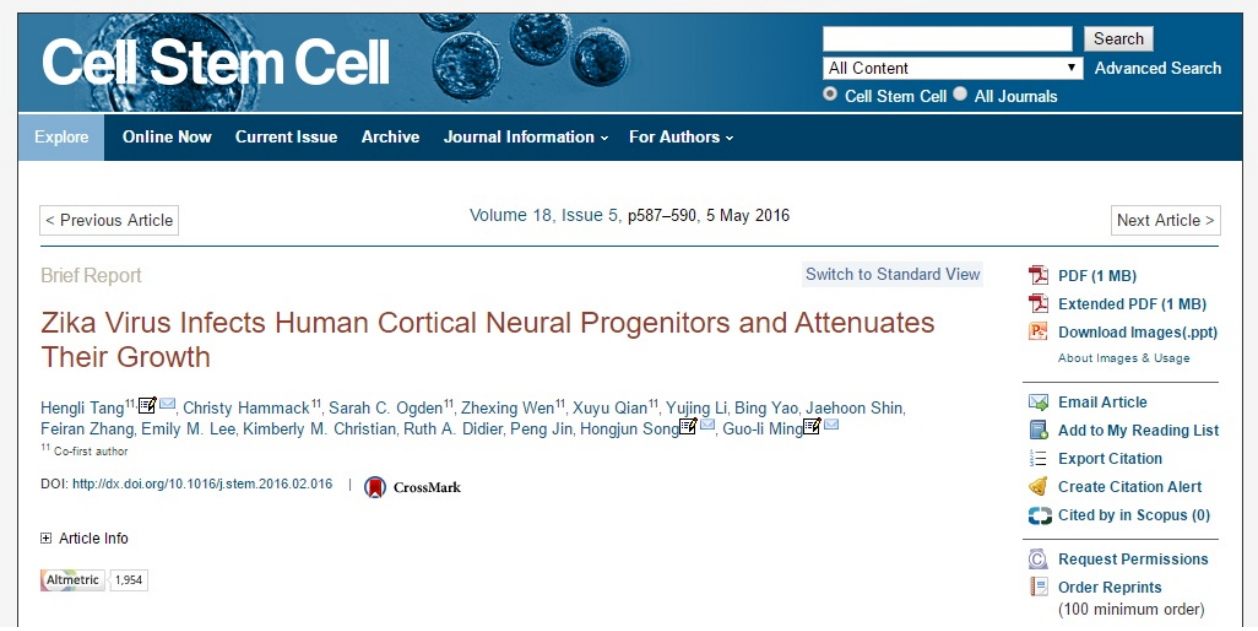
- RNA-Seq: Zika Data
- Zika RNA-Seq with alignment
- Zika RNA-Seq with kallisto

## Which scientific publications are we re-analyzing?

We are interested in accessing the data for the study titled

- Zika Virus Targets Human Cortical Neural Precursors and Attenuates Their Growth

published in the journal Cell Stem Cell in 2016.




## How do we find the data for this paper?

At the end of the paper we find the following statement in a more visible form (though again we actually found it by searching for the word GEO)

> The accession number for RNA-seq data reported in this paper is GEO: **GSE78711**

The GEO stands for the Gene Expression Omnibus and is a public functional genomics data repository that accepts array- and sequence-based data. Somewhat confusingly only the gene expression level results are actually stored in GEO, the sequence data is deposited into the Short Read Archive SRA. The

projects are then linked but it also means that there are two locations for the data.

The two locations for the data:

- SRA BioProject: PRJNA313294.
- The GEO series GSE78711.

Clearly we now have two numbers that we need to keep track of: `PRJNA313294` and `GSE78711`. In addition, the scientific journal that published the results also distributes some information from another location. This third location, has no programmatic access to allow searches. You have to click around the site and find the URLs with a manual process. If you've made it this far and you have the prior knowledge, you can then save those links as command line actions like so:

```
curl -OkL http://www.cell.com/cms/attachment/2055417636/2061019276/mmc2.xlsx
curl -OkL http://www.cell.com/cms/attachment/2055417636/2061019275/mmc1.pdf
```

Again, we think we have made our point - today data distribution is an ad-hoc and superficially supervised process that places an undo burden on any persons attempting to determine data reproducibility.

# Introduction to the command line

The terms **"Unix"** and **"Command Line"** are used interchangeably both in this Handbook and in the real world. Technically speaking, Unix is a class of operating system that originally used command line as the primary mode of interaction with the user. As computer technologies have progressed, the two terms have come to refer to the same thing.

## What is the command line?

People interact with computers via so-called **"user interfaces"**. Clicking an icon, scrolling with the mouse, and entering text into a window are all user interfaces by which we instruct the computer to perform a certain task.

When it comes to data analysis, one of the most "ancient" of interfaces—the text interface—is still the most powerful way of passing instructions to the computer.

In a nutshell, instead of clicking and pressing buttons you will be entering *words* and *symbols* that will act as direct commands and will instruct the computer to perform various processes.

## What does the command line look like?

For the uninitiated command line instructions may look like a magical incantation:

```
cat sraids.txt | parallel fastq-dump -O sra --split-files {}
```

In a way it is a modern magic - with just a few words we can unleash complex computations that would be impossible to perform in any other way. With the command line we can slay the dragons and still be home by six.

As for the above -- and don't worry if you can't follow this yet -- it is a way to download all the published sequencing data that corresponds to the ids stored in the `sraids.txt` file. The command as listed above invokes three other commands: a funny one named `cat` , a more logical sounding one `parallel` and a mysterious `fastq-dump` . This series of commands will read the content of the `sraids.txt` and will feed that line by line into the `parallel` program that in turn will launch as many `fastq-dump` programs as there are CPU cores on the machine. This parallel execution typically speeds up processing a great deal as most machines can execute more than one process at a time.

## What are the advantages of command line?

The command line presents a word by word representation of a series of actions that are explicit, shareable, repeatable and automatable. The ability to express actions with words allows us to organize these actions in the most appropriate way to solve every problem.

# What are the disadvantages of command line?

Running command line tools requires additional training and a more in depth understanding of how computers operate. Do note that this knowledge is quite valuable beyond the domain of bioinformatics.

# Is knowing the command line necessary?

Yes. While it is quite possible to perform many bioinformatics analyses without running a command line tool most analyses will benefit immensely from using a Unix like environment. Most importantly understanding Unix develops a way of thinking that is well suited to solving bioinformatics problems.

# Is the command line hard to learn?

That is not the right way to think about it.

Understand that learning the command line (and programming in general) is a not just a matter of learning a number of concepts. Instead it is primarily about learning to think a certain way - to decompose a problem into very simple steps each of which can be easily solved. Some people have an affinity to this -- they will pick it up faster, while others will need more practice. In our observation prior preparation/experience does not affect the speed at which people can learn the command line.

At the same time you have to realize that you can only succeed at changing how you think when you yourself perform these actions. In addition it also means that you can't just learn it all in day or even a week. So just keep at it for a little while longer building your skills. In the end it is really only a matter of time and practice.

Try to actively make use of command line tools by integrating them into your daily work. This is usually quite easy to do since Unix tools can be applied for just about any type of data analysis.

# Do other people also make many mistakes?

When you start out using the command line you will constantly make errors. That can be very frustrating. You could ask yourself, "Do other people make lots of mistakes too?"

Yes, we all do. We just correct them faster.

Your skill should be measured not just in making fewer mistakes but primarily in how quickly you fix the errors that you do make. So get on with making mistakes, that's how you know you are on the right path.

# How do I access the command line?

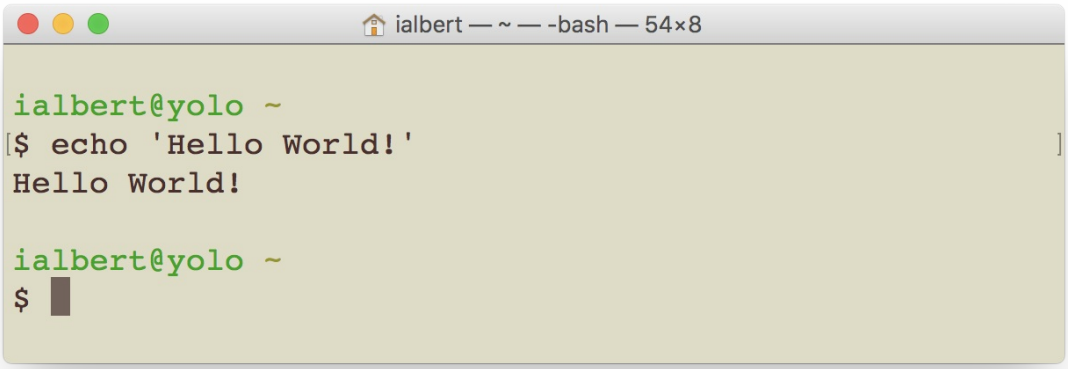
When using graphical user interfaces the command line shell is accessed via an application called "Terminal". When the "Terminal" runs it launches a so called "shell" that by default opens into a folder that is called your "home" directory.

# What is a shell?

The shell is an interface that interprets and executes the commands that are entered into it via words. There are different types of shells, the most commonly used shell is called `bash`. Below we run the `echo` command in `bash`

```
echo 'Hello World!'
```

and we see it executed in a Mac OSX Terminal below. The default terminal that opens up for you will probably look a bit different. There are options to change the fonts, background colors etc. In addition, our so called command prompt (the line where you enter the commands), is probably simpler than yours. See the Setup profile for details on how to change that.

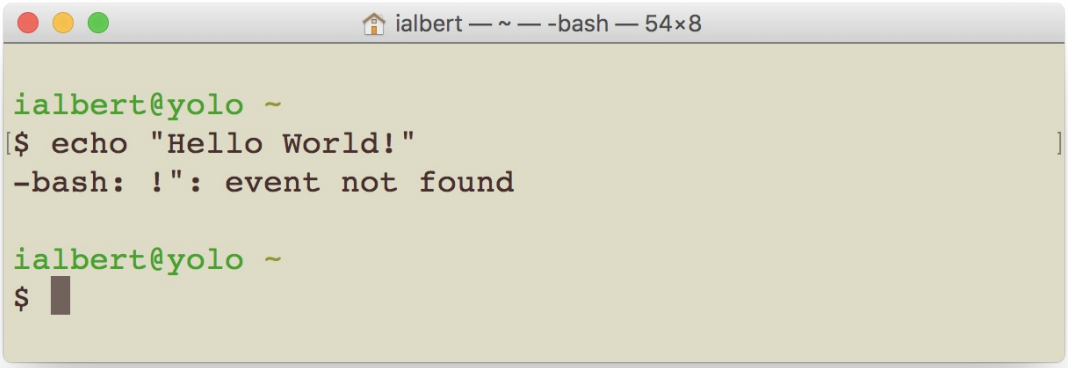



It is essential to recognize that the shell is not just a way to launch commands but a full programming language that provides powerful constructs to streamline its use.

Initially these powers tend to get in the way and cause some frustrations. For example the command above will fail when enclosed with double quotes

```
echo "Hello World!"
```

instead of echoing the output it will produce:




Maddeningly enough the error occurs only if the string contains the `!` character (or a few others). In many/most cases the single or double quotes work the same way, so for a long while you may be under the assumption that single and double quoting always works the same way.

As it turns out in `bash` strings things in double quotes are interpreted slightly differently than those enclosed in single quotes. Single quoted strings are passed exactly as they are, whereas in double quoted strings certain characters like the `!`, `$` and others may have special interpretation. This will allow you to build more powerful commands later.

> Gotcha's like these are numerous. Everyone gets tripped up at some time. Google is your friend.

# What is the best way to learn Unix?

We offer a number of tutorials right here in this book.

In addition there are excellent online tutorials, some free others at cost, each with a different take and philosophy on how to teach the concepts. We recommend that you try out a few and stick with the one that seems to work the best for you.

- CodeAcademy: Learn the command line
- Software Carpentry: The Unix shell
- Command line bootcamp
- Unix and Perl Primer for Biologists
- Learn Enough Command Line to Be Dangerous
- The Command Line Crash Course
- Learn Bash in Y minutes

And there are many other options.

# How much Unix do I need to know to be able to progress?

By the end of the your training you should have a firm grasp of the following:

1. Directory navigation: what the directory tree is, how to navigate and move around with `cd`
2. Absolute and relative paths: how to access files located in directories
3. What simple unix commands do: `ls`, `mv`, `rm`, `mkdir`, `cat`, `man`
4. Getting help: how to find out more on what a unix command does
5. What are "flags": how to customize typical unix programs `ls` vs `ls -l`
6. Shell redirection: what is the standard input and output, how to "pipe" or redirect the output of one program into the input of the other

If you don't know how to do something try Googling it. Put the relevant section of the error message into your search query. You will find that other people will have had the same problem and someone will have told them what the answer was. Google will likely direct you to one of these sites

- Ask Ubuntu
- StackOverflow

# Where can I learn more about the shell?

- Command Reference
- Explain Shell

# Unix Bootcamp

The following document is adapted from the Command-line Bootcamp by Keith Bradnam and is licensed via Creative Commons Attribution 4.0 International License. The original content has been abbreviated and simplified.

## Introduction

This 'bootcamp' is intended to provide the reader with a basic overview of essential Unix/Linux commands that will allow them to navigate a file system and move, copy, edit files. It will also introduce a brief overview of some 'power' commands in Unix.

## Why Unix?

The Unix operating system has been around since 1969. Back then, there was no such thing as a graphical user interface. You typed everything. It may seem archaic to use a keyboard to issue commands today, but it's much easier to automate keyboard tasks than mouse tasks. There are several variants of Unix (including Linux), though the differences do not matter much for most basic functions.

Increasingly, the raw output of biological research exists as *in silico* data, usually in the form of large text files. Unix is particularly suited to working with such files and has several powerful (and flexible) commands that can process your data for you. The real strength of learning Unix is that most of these commands can be combined in an almost unlimited fashion. So if you can learn just five Unix commands, you will be able to do a lot more than just five things.

## Typeset Conventions

Command-line examples that you are meant to type into a terminal window will be shown indented in a constant-width font, e.g.

```
ls -lrh
```

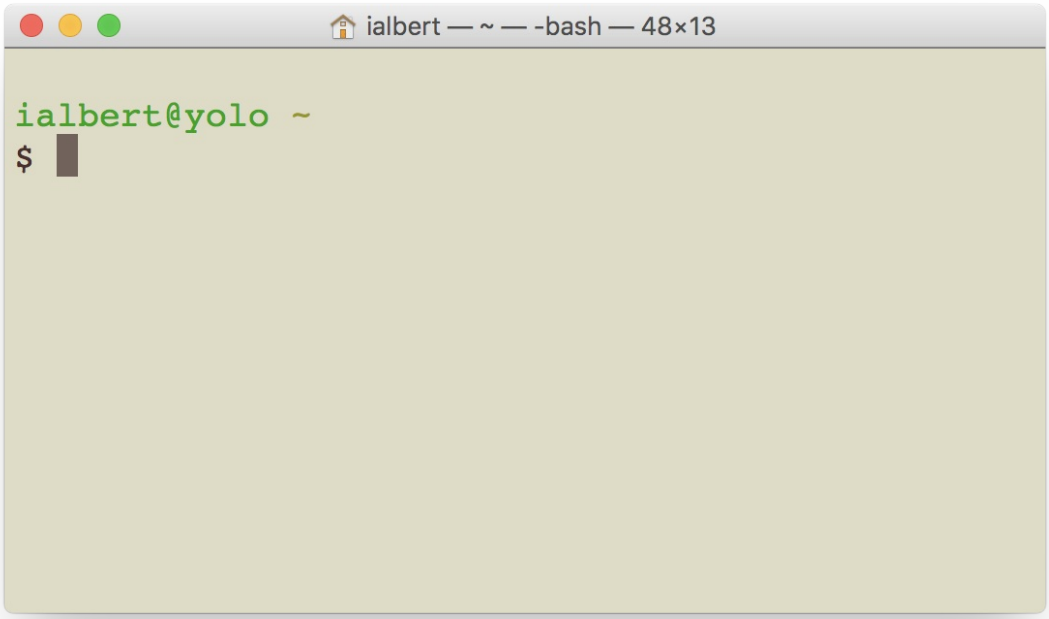
The lessons from this point onwards will assume very little apart from the following:

1. You have access to a Unix/Linux system
2. You know how to launch a terminal program on that system
3. You have a home directory where you can create/edit new files

## 1. The Terminal

A *terminal* is the common name for the program that does two main things. It allows you to type input to the computer (i.e. run programs, move/view files etc.) and it allows you to see output from those programs. All Unix machines will have a terminal program available.

Open the terminal application. You should now see something that looks like the following:




There will be many situations where it will be useful to have multiple terminals open and it will be a matter of preference as to whether you want to have multiple windows, or one window with multiple tabs (there are typically keyboard shortcuts for switching between windows, or moving between tabs).

# 2. Your first Unix command

It's important to note that you will always be *inside* a single directory when using the terminal. The default behavior is that when you open a new terminal you start in your own *home* directory (containing files and directories that only you can modify). To see what files and directories are in our home directory, we need to use the ls command. This command lists the contents of a directory. If we run the `ls` command we should see something like:

```
ls
```

prints (this depends on what computer you use):

```
Applications Desktop Documents Downloads
```

There are four things that you should note here:

1. You will probably see different output to what is shown here, it depends on your computer setup. Don't worry about that for now.
2. In your case, you may see a `$` symbol, `$ ls` and that text that you see is the Unix command prompt. Note that the command prompt might not look the same on different Unix systems. We do not include the command prompt in the examples so that you can copy-paste code. In this case, the `$` sign marks the end of the prompt.
3. The output of the `ls` command lists two things. In this case, they are directories, but they could also be files. We'll learn how to tell them apart later on. These directories were created as part of a specific course that used this bootcamp material. You will therefore probably see something very different on your own computer.
4. After the `ls` command finishes it produces a new command prompt, ready for you to type your next command.

The `ls` command is used to list the contents of *any* directory, not necessarily the one that you are currently in. Try the following:

```
ls /bin
```

it will print a lot of program names, among them,

```
bash pwd mv
```

# 3: The Unix tree

Looking at directories from within a Unix terminal can often seem confusing. But bear in mind that these directories are exactly the same type of folders that you can see if you use any graphical file browser. From the *root* level ( `/` ) there are usually a dozen or so directories. You can treat the root directory like any other, e.g. you can list its contents:

```
ls /
```

prints:

```
bin   dev   initrd.img      lib64       mnt   root  software  tmp  vmlinuz
boot  etc   initrd.img.old  lost+found  opt   run   srv       usr  vmlinuz.old
data  home  lib             media       proc  sbin  sys       var
```

You might notice some of these names appear in different colors. Many Unix systems will display files and directories differently by default. Other colors may be used for special types of files. When you log in to a computer, you are typically placed in your home directory, which is often inside a directory called 'Users' or 'home'.

# 4: Finding out where you are

There may be many hundreds of directories on any Unix machine, so how do you know which one you are in? The command pwd will print the working directory. That's pretty much all this command does:

```
pwd
```

prints:

```
/Users/ialbert
```

When you log in to a Unix computer, you are typically placed into your *home* directory. In this example, after we log in, we are placed in a directory called 'ialbert' which itself is a *subdirectory* of another directory called 'Users'. Conversely, 'Users' is the *parent* directory of 'ialbert'. The first forward slash that appears in a list of directory names always refers to the top level directory of the file system (known as the *root directory*). The remaining forward slash (between 'Users' and 'ialbert') delimits the various parts of the directory hierarchy. If you ever get 'lost' in Unix, remember the `pwd` command.

As you learn Unix you will frequently type commands that don't seem to work. Most of the time this will be because you are in the wrong directory, so it's a really good habit to get used to running the `pwd` command a lot.

# 5: Making new directories

If we want to make a new directory (e.g. to store some lecture related files), we can use the `mkdir` command:

```
mkdir edu
ls
```

shows:

```
edu
```

# 6: Getting from 'A' to 'B'

We are in the home directory on the computer but we want to to work in the new `edu` directory. To change directories in Unix, we use the `cd` command:

```
cd edu
pwd
```

will print:

```
/Users/ialbert/edu
```

Let's make two new subdirectories and navigate into them:

```
mkdir lecture
cd lecture
pwd
```

prints:

```
/Users/ialbert/edu/lecture
```

then make a data directory:

```
mkdir data
cd data/
pwd
```

prints:

```
/Users/ialbert/edu/lecture/data
```

We created the two directories in separate steps, but it is possible to use the `mkdir` command in way to do this all in one step.

Like most Unix commands, `mkdir` supports *command-line options* which let you alter its behavior and functionality. Command-like options are -- as the name suggests -- optional arguments that are placed after the command name. They often take the form of single letters (following a dash). If we had used the `-p` option of the `mkdir` command we could have done this in one step. E.g.

```
mkdir -p ~/edu/lecture/docs
```

***Note the spaces either side the `-p` !***

# 7: The root directory

Let's change directory to the root directory:

```
cd /
cd Users
cd ialbert
```

In this case, we may as well have just changed directory in one go:

```
cd /Users/ialbert
```

The leading `/` is incredibly important. The following two commands are very different:

```
cd /step1/step2
cd step1/step2/
```

The first command specifies a so called *absolute path* . It says go the root directory (folder) then the `step1` folder then the `step2` folder that opens from the `step1` directory. Importantly there can only be one `/step1/step2` directory on any Unix system.

The second command specifies a so called *relative path*. It says that from the *current location* go to the `step1` directory then the `step2` directory.

There can potentially be many `step1/step2` directories that open from different directories.

Learn and understand the difference between these two commands. Initially it is very easy to miss the leading `/` yet it is essential and fundamentally important. With a little time and practice your eyes will be trained and you will quickly notice the difference.

# 8: Navigating upwards in the Unix filesystem

Frequently, you will find that you want to go 'upwards' one level in the directory hierarchy. Two dots `..` are used in Unix to refer to the *parent* directory of wherever you are. Every directory has a parent except the root level of the computer. Let's go into the `lecture` directory and then navigate up two levels:

```
cd
cd edu
pwd
```

prints:

```
/Users/ialbert/edu
```

but now:

```
cd ..
pwd
```

prints:

```
/Users/ialbert
```

What if you wanted to navigate up *two* levels in the file system in one go? Use two sets of the `..` operator, separated by a forward slash:

```
cd ../..
```

# 9: Absolute and relative paths

Using `cd ..` allows us to change directory *relative* to where we are now. You can also always change to a directory based on its *absolute* location. E.g. if you are working in the `~/edu/lecture` directory and you then want to change to the `~/tmp` directory, then you could do either of the following:

```
cd
cd edu/lecture/data
cd ../../tmp
pwd
```

prints:

```
/Users/ialbert/
```

is identical to specifying the full path:

```
cd /Users/ialbert/edu/tmp
```

They both achieve the same thing, but the second example requires that you know about the full *path* from the root level of the computer to your directory of interest (the 'path' is an important concept in Unix). Sometimes it is quicker to change directories using the relative path, and other times it will be quicker to use the absolute path.

# 10: Finding your way back home

Remember that the command prompt shows you the name of the directory that you are currently in, and that when you are in your home directory it shows you a tilde character ( `~` ) instead? This is because Unix uses the tilde character as a short-hand way of [specifying a home directory][home directory].

See what happens when you try the following commands (use the `pwd` command after each one to confirm the results):

```
cd /
cd ~
cd
```

Hopefully, you should find that `cd` and `cd ~` do the same thing, i.e. they take you back to your home directory (from wherever you were). You will frequently want to jump straight back to your home directory, and typing `cd` is a very quick way to get there.

You can also use the `~` as a quick way of navigating into subdirectories of your home directory when your current directory is somewhere else. I.e. the quickest way of navigating from anywhere on the filesystem to your `edu/lecture` directory is as follows:

```
cd ~/edu/lecture
```

# 11: Making the `ls` command more useful

The `..` operator that we saw earlier can also be used with the `ls` command, e.g. you can list directories that are 'above' you:

```
cd ~/edu/lecture/
ls ../../
```

Time to learn another useful command-line option. If you add the letter 'l' to the `ls` command it will give you a longer output compared to the default:

```
ls -l ~
```

prints (system-dependent):

```
drwx------   5 ialbert  staff   170B May 13 15:24 Applications
drwx------+  7 ialbert  staff   238B Aug 20 05:51 Desktop
drwx------+ 30 ialbert  staff   1.0K Aug 12 13:12 Documents
drwx------+ 14 ialbert  staff   476B Aug 24 10:43 Downloads
```

For each file or directory we now see more information (including file ownership and modification times). The 'd' at the start of each line indicates that these are directories. There are many, many different options for the `ls` command. Try out the following (against any directory of your choice) to see how the output changes.

```
ls -l
ls -R
ls -l -t -r
ls -lh
```

Note that the last example combines multiple options but only uses one dash. This is a very common way of specifying multiple command-line options. You may be wondering what some of these options are doing. It's time to learn about Unix documentation....

# 12: Man pages

If every Unix command has so many options, you might be wondering how you find out what they are and what they do. Well, thankfully every Unix command has an associated 'manual' that you can access by using the `man` command. E.g.

```
man ls
man cd
man man # Yes, even the man command has a manual page
```

When you are using the man command, press `space` to scroll down a page, `b` to go back a page, or `q` to quit. You can also use the up and down arrows to scroll a line at a time. The man command is actually using another Unix program, a text viewer called `less`, which we'll come to later on.

# 13: Removing directories

We now have a few (empty) directories that we should remove. To do this use the [rmdir][] command. This will only remove empty directories, so it is quite safe to use. If you want to know more about this command (or any Unix command), then remember that you can just look at its man page.

```
cd ~/edu/lecture/
rmdir data
cd ..
rmdir lecture
ls
```

**Note, you have to be outside a directory before you can remove it with `rmdir`**

# 14: Using tab completion

Saving keystrokes may not seem important now, but the longer that you spend typing in a terminal window, the happier you will be if you can reduce the time you spend at the keyboard. Especially as prolonged typing is not good for your body. So the best Unix tip to learn early on is that you can [tab complete][] the names of files and programs on most Unix systems. Type enough letters to uniquely identify the name of a file, directory, or program and press tab -- Unix will do the rest. E.g. if you type 'tou' and then press tab, Unix should autocomplete the word to 'touch' (this is a command which we will learn more about in a minute). In this case, tab completion will occur because there are no other Unix commands that start with 'tou'. If pressing tab doesn't do anything, then you have not have typed enough unique characters. In this case pressing tab *twice* will show you all possible completions. This trick can save you a LOT of typing!

Navigate to your home directory, and then use the `cd` command to change to the `edu` directory. Use tab completion to complete directory name. If there are no other directories starting with 'e' in your home directory, then you should only need to type 'cd' + 'e' + 'tab'.

> **Tab completion will make your life easier and make you more productive!**

Another great time-saver is that Unix stores a list of all the commands that you have typed in each login session. You can access this list by using the `history` command or more simply by using the up and down arrows to access anything from your history. So if you type a long command but make a mistake, press the up arrow and then you can use the left and right arrows to move the cursor in order to make a change.

# 15: Creating empty files with the touch command

The following sections will deal with Unix commands that help us to work with files, i.e. copy files to/from places, move files, rename files, remove files, and most importantly, look at files. First, we need to have some files to play with. The Unix command `touch` will let us create a new, empty file. The touch command does other things too, but for now we just want a couple of files to work with.

```
cd edu
touch heaven.txt
touch earth.txt
ls
```

prints:

```
earth.txt  heaven.txt
```

# 16: Moving files

Now, let's assume that we want to move these files to a new directory ('temp'). We will do this using the Unix [mv][] (move) command. Remember to use tab completion:

```
mkdir temp
mv heaven.txt temp
mv earth.txt temp
ls temp
```

For the `mv` command, we always have to specify a source file (or directory) that we want to move, and then specify a target location. If we had wanted to, we could have moved both files in one go by typing any of the following commands:

```
mv *.txt temp
mv *t temp
mv *ea* temp
```

The asterisk `*` acts as a *wild-card character*, essentially meaning 'match anything'. The second example works because there are no other files or directories in the directory that end with the letter 't' (if there were, then they would be moved too). Likewise, the third example works because only those two files contain the letters 'ea' in their names. Using wild-card characters can save you a lot of typing.

The `?` character is also a wild-card but with a slightly different meaning. See if you can work out what it does.

# 17: Renaming files

In the earlier example, the destination for the `mv` command was a directory name (temp). So we moved a file from its source location to a target location, but note that the target could have also been a (different) file name, rather than a directory. E.g. let's make a new file and move it whilst renaming it at the same time:

```
touch rags
ls
mv rags temp/riches
ls temp/
```

prints:

```
earth.txt  heaven.txt  riches
```

In this example we create a new file ('rags') and move it to a new location and in the process change the name (to 'riches'). So `mv` can rename a file as well as move it. The logical extension of this is using `mv` to rename a file without moving it. You may also have access to a tool called `rename`, type `man rename` for more information.

```
mv temp/riches temp/rags
```

# 18: Moving directories

It is important to understand that as long as you have specified a 'source' and a 'target' location when you are moving a file, then it doesn't matter what your *current* directory is. You can move or copy things within the same directory or between different directories regardless of whether you are in any of those directories. Moving directories is just like moving files:

```
mv temp temp2
ls temp2
```

# 19: Removing files

You've seen how to remove a directory with the `rmdir` command, but `rmdir` won't remove directories if they contain any files. So how can we remove the files we have created (inside `temp`)? In order to do this, we will have to use the `rm` (remove) command.

> **Please read the next section VERY carefully. Misuse of the `rm` command can lead to needless death & destruction**

Potentially, `rm` is a very dangerous command; if you delete something with `rm`, you will not get it back! It is possible to delete everything in your home directory (all directories and subdirectories) with `rm`. That is why it is such a dangerous command.

Let me repeat that last part again. It is possible to delete EVERY file you have ever created with the `rm` command. Are you scared yet? You should be. Luckily there is a way of making `rm` a little bit safer. We can use it with the `-i` command-line option which will ask for confirmation before deleting anything (remember to use tab-completion):

```
cd temp
ls
rm -i earth.txt heaven.txt rags
```

will ask permission for each step:

```
rm: remove regular empty file 'earth.txt'? y
rm: remove regular empty file 'heaven.txt'? y
rm: remove regular empty file 'rags'? y
```

We could have simplified this step by using a wild-card (e.g. `rm -i *.txt`) or we could have made things more complex by removing each file with a separate `rm` command.

# 20: Copying files

Copying files with the `cp` (copy) command has a similar syntax as `mv`, but the file will remain at the source and be copied to the target location. Remember to always specify a source and a target location. Let's create a new file and make a copy of it:

```
touch file1
cp file1 file2
ls
```

What if we wanted to copy files from a different directory to our current directory? Let's put a file in our home directory (specified by `~`, remember) and copy it to the lecture directory (`~/edu/lecture`):

```
$ touch ~/edu/file3
$ cp ~/edu/file3 ~/edu/lecture/
```

In Unix, the current directory can be represented by a `.` (dot) character. You will often use for copying files to the directory that you are in. Compare the following:

```
ls
ls .
ls ./
```

In this case, using the dot is somewhat pointless because `ls` will already list the contents of the current directory by default. Also note how the trailing slash is optional.

# 21: Copying directories

The `cp` command also allows us (with the use of a command-line option) to copy entire directories. Use `man cp` to see how the `-R` or `-r` options let you copy a directory *recursively*.

# 22: Viewing files with less (or more)

So far we have covered listing the contents of directories and moving/copying/deleting either files or directories. Now we will quickly cover how you can look at files. The `more` or `less` commands let you view (but not edit) text files. We will use the `echo` command to put some text in a file and then view it:

```
echo "Call me Ishmael."
Call me Ishmael.
echo "Call me Ishmael." > opening_lines.txt
ls
```

prints:

```
opening_lines.txt
```

we can view the content of the file with:

```
more opening_lines.txt
```

On its own, `echo` isn't a very exciting Unix command. It just echoes text back to the screen. But we can redirect that text into an output file by using the `>` symbol. This allows for something called file *redirection*.

> **Careful when using file redirection (>), it will overwrite any existing file of the same name**

When you are using `more` or `less`, you can bring up a page of help commands by pressing `h`, scroll forward a page by pressing `space`, or go forward or backwards one line at a time by pressing `j` or `k`. To exit `more` or `less`, press `q` (for quit). The `more` and `less` programs also do about a million other useful things (including text searching).

# 23: Viewing files with cat

Let's add another line to the file:

```
echo "The primroses were over." >> opening_lines.txt
cat opening_lines.txt
```

prints:

```
Call me Ishmael.
The primroses were over.
```

Notice that we use `>>` and not just `>`. This operator will **append** to a file. If we only used `>`, we would end up overwriting the file. The `cat` command displays the contents of the file (or files) and then returns you to the command line. Unlike `less` you have no control on how you view that text (or what you do with it). It is a very simple, but sometimes useful, command. You can use `cat` to quickly combine multiple files or, if you wanted to, make a copy of an existing file:

```
cat opening_lines.txt > file_copy.txt
```

# 24: Counting characters in a file

```
$ ls
opening_lines.txt

$ ls -l
total 4
-rw-rw-r-- 1 ubuntu ubuntu 42 Jun 15 04:13 opening_lines.txt

$ wc opening_lines.txt
 2  7 42 opening_lines.txt

$ wc -l opening_lines.txt
2 opening_lines.txt
```

The `ls -l` option shows us a long listing, which includes the size of the file in bytes (in this case '42'). Another way of finding this out is by using Unix's `wc` command (word count). By default this tells you many lines, words, and characters are in a specified file (or files), but you can use command-line options to give you just one of those statistics (in this case we count lines with `wc -l`).

# 25: Editing small text files with nano

Nano is a lightweight editor installed on most Unix systems. There are many more powerful editors (such as 'emacs' and 'vi'), but these have steep learning curves. Nano is very simple. You can edit (or create) files by typing:

```
nano opening_lines.txt
```

You should see the following appear in your terminal:

The bottom of the nano window shows you a list of simple commands which are all accessible by typing 'Control' plus a letter. E.g. Control + X exits the program.

## 26: The `$PATH` environment variable

One other use of the `echo` command is for displaying the contents of something known as *environment variables*. These contain user-specific or system-wide values that either reflect simple pieces of information (your username), or lists of useful locations on the file system. Some examples:

```
$ echo $USER
ialbert
$ echo $HOME
/Users/ialbert
echo $PATH
/usr/local/sbin:/usr/local/bin:/Users/ialbert/bin
```

The last one shows the content of the `$PATH` environment variable, which displays a — colon separated — list of directories that are expected to contain programs that you can run. This includes all of the Unix commands that you have seen so far. These are files that live in directories which are run like programs (e.g. `ls` is just a special type of file in the `/bin` directory).

Knowing how to change your `$PATH` to include custom directories can be necessary sometimes (e.g. if you install some new bioinformatics software in a non-standard location).

## 27: Matching lines in files with grep

Use `nano` to add the following lines to `opening_lines.txt` :

```
Now is the winter of our discontent.
All children, except one, grow up.
The Galactic Empire was dying.
In a hole in the ground there lived a hobbit.
It was a pleasure to burn.
It was a bright, cold day in April, and the clocks were striking thirteen.
It was love at first sight.
I am an invisible man.
It was the day my grandmother exploded.
When he was nearly thirteen, my brother Jem got his arm badly broken at the elbow.
Marley was dead, to begin with.
```

You will often want to search files to find lines that match a certain pattern. The Unix command `grep` does this (and much more). The following examples show how you can use grep's command-line options to:

- show lines that match a specified pattern
- ignore case when matching ( `-i` )
- only match whole words ( `-w` )
- show lines that don't match a pattern ( `-v` )
- Use wildcard characters and other patterns to allow for alternatives ( `*` , `.` , and `[]` )

Show lines that match the word `was` :

```
$ grep was opening_lines.txt
The Galactic Empire was dying.
It was a pleasure to burn.
It was a bright, cold day in April, and the clocks were striking thirteen.
It was love at first sight.
It was the day my grandmother exploded.
When he was nearly thirteen, my brother Jem got his arm badly broken at the elbow.
Marley was dead, to begin with.
```

Use:

```
grep --color=AUTO was opening_lines.txt
```

to highlight the match.

Show lines that do not match the word `was` :

```
$ grep -v was opening_lines.txt
Now is the winter of our discontent.
All children, except one, grow up.
In a hole in the ground there lived a hobbit.
I am an invisible man.
```

`grep` has a great many options and applications.

# 28: Combining Unix commands with pipes

One of the most powerful features of Unix is that you can send the output from one command or program to any other command (as long as the second command accepts input of some sort). We do this by using what is known as a `pipe`. This is implemented using the '|' character (which is a character which always seems to be on different keys depending on the keyboard that you are using). Think of the pipe as simply connecting two Unix programs. Here's an example which introduces some new Unix commands:

```
grep was opening_lines.txt | wc -c
# 316

grep was opening_lines.txt | sort | head -n 3 | wc -c
# 130
```

The first use of `grep` searches the specified file for lines matching 'was'. It sends the lines that match through a pipe to the `wc` program. We use the `-c` option to just count characters in the matching lines (316).

The second example first sends the output of `grep` to the Unix `sort` command. This sorts a file alphanumerically by default. The sorted output is sent to the `head` command which by default shows the first 10 lines of a file. We use the `-n` option of this command to only show 3 lines. These 3 lines are then sent to the `wc` command as before.

> **Whenever making a long pipe, test each step as you build it!**

# Miscellaneous Unix power commands

The following examples introduce some other Unix commands, and show how they could be used to work on a fictional file called `file.txt`. Remember, you can always learn more about these Unix commands from their respective man pages with the `man` command. These are not all real world cases, but rather show the diversity of Unix command-line tools:

- View the penultimate (second-to-last) 10 lines of a file (by piping `head` and `tail` commands):

```
#tail -n 20 gives the last 20 lines of the file, and piping that to head will show the first 1
0 lines of the last 20.
tail -n 20 file.txt | head
```

- Show the lines of a file that begin with a start codon (ATG) (the `^` matches patterns at the start of a line):

```
grep "^ATG" file.txt
```

- Cut out the 3rd column of a tab-delimited text file and sort it to only show unique lines (i.e. remove duplicates):

```
cut -f 3 file.txt | sort -u
```

- Count how many lines in a file contain the words 'cat' or 'bat' ( `-c` option of `grep` counts lines):

```
grep -c '[bc]at' file.txt
```

- Turn lower-case text into upper-case (using `tr` command to 'transliterate'):

```
cat file.txt | tr 'a-z' 'A-Z'
```

- Change all occurrences of 'Chr1' to 'Chromosome 1' and write changed output to a new file (using `sed` command):

```
cat file.txt | sed 's/Chr1/Chromosome 1/' > file2.txt
```

# Data analytics with Unix

In this section we will use simple unix tools to obtain and investigate a genome data file that represents all the annotated features of the yeast genome.

This section is not meant to be a full introduction to all the utility of the Unix command line.

What we attempt to show you are the concepts, methods and the manner in which unix tools are typically employed to probe and poke at data.

## What directory should I use?

When you open a terminal it starts in a certain location (directory, folder) on your computer. You can of course move around and navigate to other locations, but every action that you take is always relative to the location from which you are giving the command.

For each project, lecture or task create a separate folder to work in.

Never start typing commands from your home directory. Tools create several ancillary files, and you do not want these mixed in with your other work related data.

Your two BFFs (best friends forever) are `pwd` and `ls`. These will tell you the most important information:

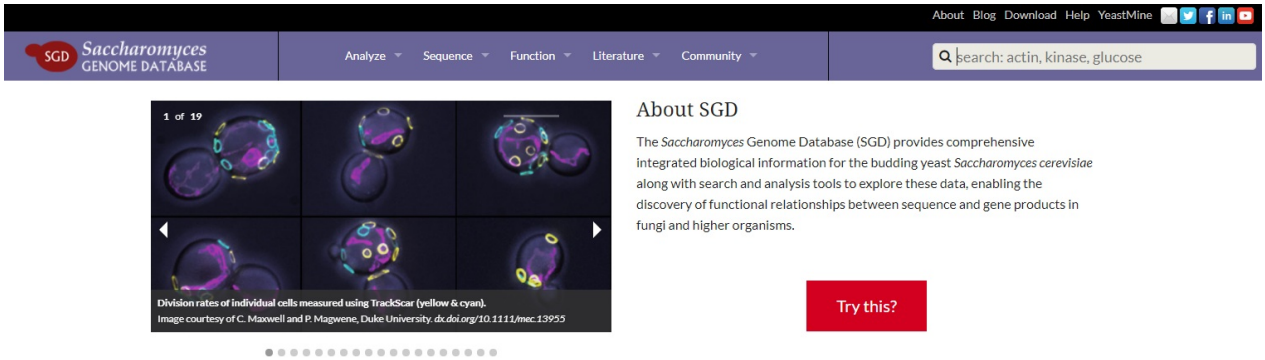
1. `pwd` will tell you *where* you are.
2. `ls` will tell you *what* you have there.

Especially at the beginning a lot of confusion is caused by not knowing these two pieces of information. When in doubt ask yourself:

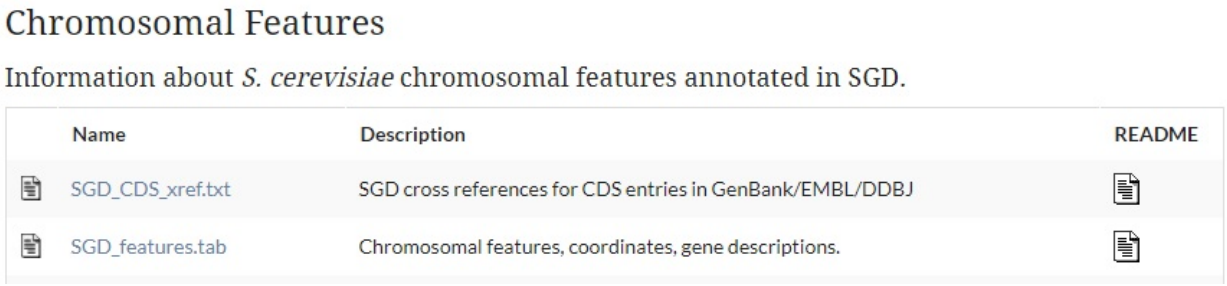
1. Where am I on my computer?
2. What do I have in the current folder?

## Where are we getting the data from?

We'll visit the Saccharomyces Genome Database (SGD) webpage. What you will notice that while clicking around "makes sense" it is not easy to properly describe where you would get a given dataset. All too often you will hear people say: *"I've downloaded data from the XYZ database"* as if there was only one data to be downloaded. Often you be left scratching your head as to what exactly did they download.




Visit the "Download" link then "Curated Data" then **"Chromosomal Features".** From that section we want to investigate the file called `SGD_features.tab` that is described as *"Chromosomal features, coordinates, gene descriptions."*. Let's get it into our terminal.




The first thing you need to do is obtain the URL of this file. While hovering over the link right click and select "Copy link address". You now have the location of the data:

```
http://downloads.yeastgenome.org/curation/chromosomal_feature/SGD_features.tab
```

# How do I obtain a data file that is online?

We can use either the `curl` or `wget` command line tool (see Curl vs wget) to download the content of URLs.

```
# Stow your work in a directory
mkdir lec03
cd lec03

# Get the data from SGD.
wget http://downloads.yeastgenome.org/curation/chromosomal_feature/SGD_features.tab

# Also get the README file.
wget http://downloads.yeastgenome.org/curation/chromosomal_feature/SGD_features.README

# Quick check. Where are you and what have you got?
pwd
ls
```

Simple Unix tools let us answer a wide variety of questions:

## How do I view the data one page at a time?

```
more SGD_features.README
```

press "space" to move forward, "b" to move backward, `q` or `ESC` to exit.

## How do I open a stream from a data?

The `cat` command concatenates one or more files and stars producing them on the output stream:

```
cat SGD_features.tab
```

This will print the contents of the file to the current output stream which happens to be your terminal window.

Tips:

- Use the manual `man` for more information on commands.
- If you want to rerun a command you can use the up-arrow key to recall the previous entry.
- You can press TAB to ask your shell to try to complete a file name. It is very handy! Always try to have the shell fill in the filename for you, that way you will never mistype these.

## How many lines does the file have?

You can pipe the output of you stream into another program rather than the screen. Use the `|` character to connect the programs. For example the `wc` program is the word counter.

```
cat SGD_features.tab | wc
```

prints the number of lines, words and characters in the stream:

```
16454  425719 3264490
```

You can customize what the `wc` program does

```
cat SGD_features.tab | wc -l
```

so that it will only print the number of lines:

```
16454
```

Now many programs also operate on files directly like so:

```
wc -l SGD_features.tab
```

but in general it is much clearer to open a stream with `cat` then pipe the stream into the next program. It is easier to design, build and understand more complicated pipelines when the stream is opened as a separate step.

## How does the file start?

```
cat SGD_features.tab | head
```

prints:

```
S000002143      ORF     Dubious YAL069W                 chromosome 1            1       335
  649     W               1996-07-31      1996-07-31      Dubious open reading frame; unlikely
 to encode a functional protein, based on available experimental and comparative sequence data
S000031098      CDS                                     YAL069W         1       335     649
  W               1996-07-31      1996-07-31
S000028594      ORF     Dubious YAL068W-A                       chromosome 1            1
  538     792     W               2003-07-29      2003-07-29      Dubious open reading frame;
unlikely to encode a functional protein, based on available experimental and comparative seque
nce data; identified by gene-trapping, microarray-based expression analysis, and genome-wide h
omology searching
S000031372      CDS                                     YAL068W-A               1       538
  792     W               2003-07-29      2003-07-29
```

## Which lines match a certain pattern?

Suppose you wanted to find information on gene `YAL060W`

```
cat SGD_features.tab  | grep YAL060W
```

## How can I color the pattern that is matched?

```
cat SGD_features.tab | grep YAL060W --color=always | head
```

## How can we tell how many lines DO NOT match a given pattern?

Adding the `-v` flag to `grep` reverses the action of `grep` it shows the lines that do not match.

```
cat SGD_features.tab | grep -v Dubious | wc -l
```

## How do I store the results in a new file?

The `>` character is the redirection.

```
cat SGD_features.tab | grep YAL060W > match.tab
```

now check how many files do you have:

```
match.tab
SGD_features.README
SGD_features.tab
```

## How many lines in the file match the word gene?

```
cat sc.tab | grep gene | wc -l
```

Important: note how the above matches the word gene anywhere on the line.

## How do I select genes?

It looks like this file uses the feature type (column 2) ORF for protein coding genes. You will need to cut the second field (by tabs).

```
cat SGD_features.tab | cut -f 2 | head
```

Build your commands one step at a time, always checking that you are on the right track:

```
cat SGD_features.tab | head
cat SGD_features.tab | cut -f 2 | head
cat SGD_features.tab | cut -f 2 | grep ORF | head
```

## How many genes are there?

```
cat SGD_features.tab | cut -f 2 | grep ORF | wc -l
```

## Can I select multiple columns?

```
cat SGD_features.tab | cut -f 2,3,4 | grep ORF | head
```

What does this do?

```
cat SGD_features.tab | cut -f 2,3,4 | grep ORF | grep -v Dubious | wc -l
```

# How many feature types are in this data?

We are going to use this data a lot so place it into a separate file for now.

```
cat SGD_features.tab | cut -f 2 > types.txt
```

Sorting places identical consecutive entries next to one another.

```
cat types.txt | sort | head
```

Find unique words. The `uniq` command collapses consecutive identical words into one.

```
cat types.txt | sort | uniq | head
```

Using `-c` flag to `uniq` will not only collapse consecutive entires it will print their counts.

```
cat types.txt | sort | uniq -c | head
```

prints:

```
352 ARS
196 ARS_consensus_sequence
  6 blocked_reading_frame
7074 CDS
 16 centromere
 16 centromere_DNA_Element_I
 16 centromere_DNA_Element_II
 16 centromere_DNA_Element_III
  8 external_transcribed_spacer_region
 24 five_prime_UTR_intron
```

Find out how many unique types of features are in this file:

```
cat SGD_features.tab | cut -f 2 | sort | uniq -c | wc -l
```

# Compressed files and directories

## What is a compressed format?

Compressed formats reduce file disk space requirements. A compressed file contains the same information but has now been optimized for size. On the downside, it needs to be decompressed to access the content.

Compression requires substantially more computational resources than decompression. It is always much faster (can be an order of magnitude faster) to restore a file from a compressed form than to create that compressed form.

Some compressed formats (like the .gz format) can be concatenated (or appended to) in compressed form. This can be handy as it avoids having to decompress the files, concatenate them and re-compress.

## What may be compressed?

You can compress single files or entire directories (these are called archives).

- A "compressed file" is a single file reduced in size. The name may end with `.gz`, `.bz`, `.bz2`, or `.zip`.
- A "compressed archive" is a folder containing multiple files combined, and then compressed. The name may end with `.tar.gz`, `.tar.bz2`, etc.

## What are some common compression formats?

- `ZIP`, extension `.zip`, program names are `zip/unzip`. Available on most platforms.
- `GZIP` extension `.gz`, program names are `gzip/gunzip`. The most common compression format on Unix-like systems.
- `BZIP2` extension `.bz/.bz2`, program names are `bzip2/bunzip2`.
- `XZ` extension `.xz`. A more recent invention. Technically bzip2 and (more so) xz are improvements over `gzip`, but gzip is still quite prevalent for historical reasons and because it requires less memory.

## Is there a bioinformatics-specific compression format?

Yes, the `bgzip` utility is installed when you install the `htslib` package. The utility implements BGZF (Blocked GNU Zip Format) that is a variant of the `GZIP` format but one that allows random access to the content of a compressed file. The BAM and BCF and the compressed VCF files typically need to be BGZIP compressed. A BGZIP file can be decompressed with `gzip` but only `bgzip` can create a BGZIP file.

```
# Get a sequence file.
efetch -db=nuccore -format=fasta -id=AF086833 > AF086833.fa
bgzip AF086833.fa
```

More about bgzip: http://www.htslib.org/doc/tabix.html

# How do I compress or uncompress a file?

```
# Get a sequence file.
efetch -db=nuccore -format=fasta -id=AF086833 > AF086833.fa

# Compress with gzip.
# Creates the file AF086833.fa.gz.
gzip AF086833.fa

# Some tools can operate on GZ files. If not you
# can also uncompress on "on the fly" without decompressing
# the entire file.
# Note: gzcat on macOS, zcat on Linux.
gzcat AF086833.fa.gz  | head

# Uncompress the file
# Creates the file AF086833.fa.
gunzip AF086833.fa.gz
```

# How do I compress or uncompress multiple files?

Zip-files may contain one or multiple files. This is a bit annoying in that you can't tell from the file name alone what the archive contains.

In the unix world the most commonly used approach is to create a so called "tape archive" even though it is not typically involving tape anymore. This is called a `tar` file (Tape Archive) file. The extension for gzip-compressed tar files is `.tar.gz`.

If we had the following two files:

```
efetch -db=nuccore -format=fasta -id=AF086833 > AF086833.fa
efetch -db=nuccore -format=gb -id=AF086833 > AF086833.gb
```

The command works by subcommands that are shortened from the actions it performs:

```
tar czfv  archive-name  list-of-files-that-go-into-the-archive
```

Means that we want to create `c`, a compressed `z`, file `f`, in verbose `v` mode in a file called `sequences.tar.gz` with the following command:

```
tar czfv sequences.tar.gz AF086833.fa AF086833.gb
```

The first name listed on the command `sequences.tar.gz` is the file to be created, the other two are the files that should be added, equivalent to:

```
tar czvf sequences.tar.gz AF086833.*
```

If you accidentally put a file that intended to compress first in the line it will be overwritten (destroyed) as tar tries to create that file, so watch out for that!

Unless we have very few files, this is not a good way to package files -- upon unpacking we may not remember which files we have unpacked.

It is best if we put all files to be packed into a subdirectory (this also helps with the overwriting problem above).

```
mkdir sequences
mv AF086833.* sequences/
```

then compress that entire directory like so:

```
tar czvf sequences.tar.gz sequences/*
```

# What is an rsyncable archive?

As we seen before `rsync` is a tool that can synchronize files by sending only the differences between existing files. When the files are compressed `rsync` needs extra information -- otherwise the whole file will be synchronized on any change. Linux versions of `gzip` have the `--rsyncable` flag that can make gzip files that can be synced much faster. You can compress your archives in an rsync-friendly like this:

```
tar -c sequences/* | gzip --rsyncable > file.tar.gz
```

# What is a tarbomb?

A tarbomb is a tar file created by an unwitting user that contains a very large number of files.

Here is an example:

```
curl -O http://data.biostarhandbook.com/data/tarbomb.tar.gz
```

Upon unpacking it "explodes" all over your directory.

Disarming a tarbomb:

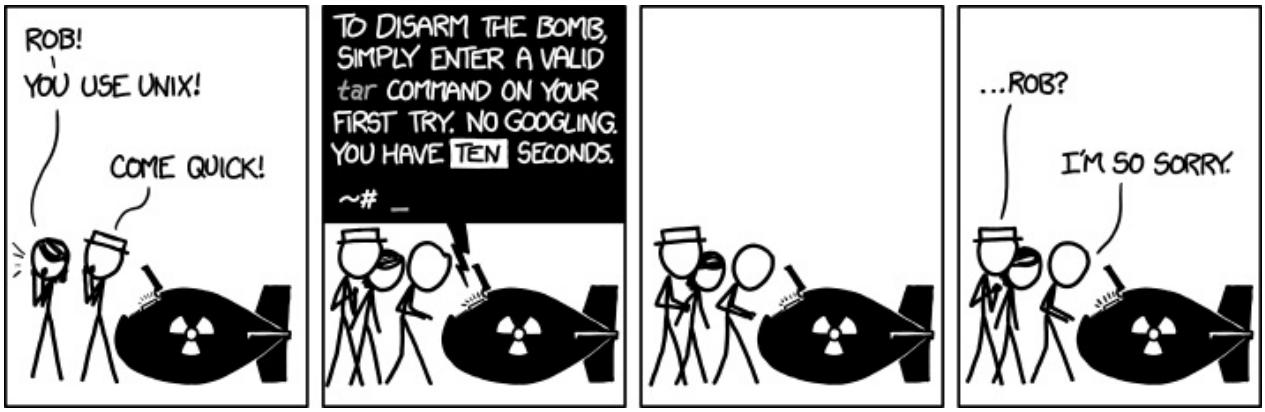
1. View the content of an archive:

```
tar tzvf tarbomb.tar.gz
```

1. Move it to a separate directory and unpack it there.

## How do we use tar again?

You may forget which `tar` flags you need. You're not alone. See http://xkcd.com/1168/ for a different kind of tarbomb:



# Ontologies

## What do words mean?

Computational concepts and algorithms require exact definitions. It is exceedingly important that all scientists use the same words to describe the same information.

An *ontology* is a controlled vocabulary of *terms* or *concepts* and a restricted set of *relationships* between those terms. The most commonly used ontologies for bioinformatics data are:

- The Sequence Ontology ( `SO` ): a vocabulary for information related to sequence features.
- The Gene Ontology ( `GO` ), a vocabulary for information related to gene functions.

## Why is the ontology necessary?

In the paper titled The Sequence Ontology: a tool for the unification of genome annotations, the authors state:

> Unfortunately, biological terminology is notoriously ambiguous; the same word is often used to describe more than one thing and there are many dialects. For example, does a coding sequence (CDS) contain the stop codon or is the stop codon part of the 3'-untranslated region (3' UTR)?
>
> There really is no right or wrong answer to such questions, but consistency is crucial when attempting to compare annotations from different sources, or even when comparing annotations performed by the same group over an extended period of time.

It is essential to recognize that consistency matters more than "correctness" - there is no universally "correct" answer in biology.

## Are there other ontologies?

Yes. Possibly too many. See a list at the The Open Bioinformatic Ontologies (OBO) Foundry.

Creating an ontology is a difficult, thankless job. Scientists often underestimate the difficulty of coming up with logically consistent definitions. For that reason, they often embark on overly ambitious ontology building adventures. Hence, there are many biomedical ontologies left in various states of abandon and disrepair. See the sorry situation of the RNA Ontology for example.

## Who names the genes?

A few different organizations have taken upon themselves the obligation to maintain a consistent naming scheme.

- For example HUGO Gene Nomenclature Committee (HGNC) available via

> http://www.genenames.org/ is the only worldwide authority that assigns standardized nomenclature to human genes.

Other model and non-model organisms might have their consortia that may follow similar or very different naming conventions to label the genes for that organism.

Whereas the gene names are standardized other names, such transcript names, regulatory elements, etc may have competing labeling. Converting between these across different organizations and their data releases continues to be one of most annoying and unsolved problems of bioinformatics.

# What will our data tell us?

In general, most bioinformatics-oriented analysis results fall into two categories (and combinations thereof):

1. **What a piece of DNA is**: annotations or classifications.
2. **What a piece of DNA does**: functional analyses.

For example, a **de-novo genome assembly** attempts to reconstruct a genome that was measured after being broken into many small fragments. Once assembled some part of the genome can be annotated with terms such as the capacity to encode for proteins or the ability to regulate RNA expression. The ideal result of this analysis would be a genome where every base carries annotations that describe its role.

In contrast a typical **RNA-Seq** study, which aims to explain phenotypes by identifying the transcripts that show variations in their expression levels. Since transcripts have functional roles associated with them, the change in the abundance of a transcript is assumed to affect its functional role, and hence to influence the phenotype of the organism. The ideal outcome of an RNA-Seq experiment is the identification of the mechanisms by which the DNA functions and those by which it produces an observed phenotype.

# Sequence Ontology

## What is the Sequence Ontology (SO)?

The Sequence Ontology (SO) at http://www.sequenceontology.org/ is a collaborative ontology project for the definition of sequence features used in biological sequence annotation.

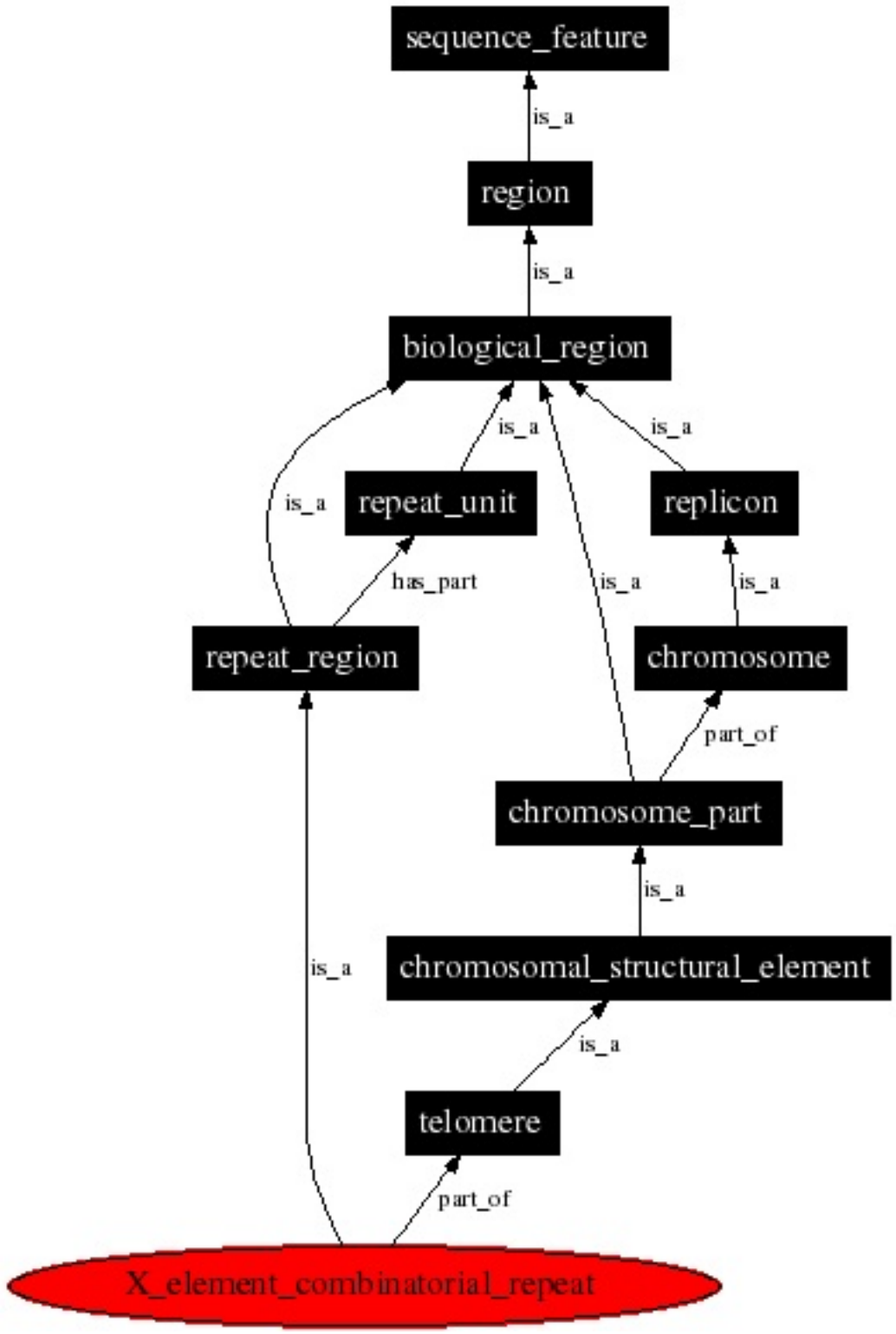
## How to find term definitions?

Search the Sequence Ontology Browser for the words that need to be defined.

For example, to define the term `X_element_combinatorial_repeat` within a sequencing data context, we can visit the Sequence Ontology Browser and find its definition to be:

> **X element combinatorial repeat**
>
> An X element combinatorial repeat is a repeat region located between the X element and the telomere or adjacent Y' element.

A graphical display shows the relationship of the terms to its parents:




Note how the description above makes use of the word `X Element` that in turn can also be found via another search in the browser to be:

> **X element**
>
> The X element is a conserved region, of the telomere, of ~475 bp that contains an ARS sequence and in most cases an Abf1p binding site.

And so on. One may now need the word `telomere` to define. Keep searching until you find all term definitions. Click the words on the image to jump to another term - this makes navigating much easier.

# Does all sequencing data obey the rules of SO?

It should but sometimes it does not. Adopt a defensive mindset and verify your data. A standard error, for example, is when data sources do not include the sequence for the `stop codon` in a sequence labeled as `CDS` although based on the definition they should:

> **CDS**
>
> A contiguous sequence which begins with, and includes a start codon and ends with, and includes, a stop codon.

# How are the SO relationships defined?

Whereas SO terminology is precise, the rationale for the SO relationships between terms are more complicated to understand - and they feel more subject to interpretation. For that and other reasons the majority of life scientists do not make use of the SO relationships and use the SO terms only.

For details we recommend consulting the publication: The Sequence Ontology: A tool for the unification of genome annotations. Genome Biology, 2005.

# How can I investigate the SO data?

While during regular use we don't need to access the original SO data it is worth spending a bit a time to understand how it is organized and what it contains.

Finding the SO data is somewhat difficult. There is a GitHub repository at https://github.com/The-Sequence-Ontology/SO-Ontologies. But even on that site there is little information on how this data is formatted.

```
URL=https://raw.githubusercontent.com/The-Sequence-Ontology/SO-Ontologies/master/so-simple.obo
wget $URL
```

The `obo` format is plain text and is NOT column oriented. Hence most Unix tools don't quite work well with them. Instead, the data is "record" based where a multiline record lists the term and its parent via the `is_a` relationship. Here is an example record:

```
[Term]
id: SO:0000004
name: interior_coding_exon
subset: SOFA
synonym: "interior coding exon" EXACT []
is_a: SO:0000195 ! coding_exon
```

# How many Sequence Ontology terms are there?

We can easily count how many `SO` terms are there:

```
cat so-simple.obo | grep 'Term' | wc -l
```

The entire Sequence Ontology is just `2359` terms. Is that a lot? Is that little? Are we close to be finished or not with naming things? It is hard to tell. Let's just say that based on our current understanding of the genome we have `2359` terms to associate with sequences.

# How can I quickly search the Sequence Ontology?

Grepping the raw data is probably the fastest way to find what you are looking for.

How many record definitions match the word `gene`:

```
cat so-simple.obo | grep 'name: gene'
```

prints:

```
name: gene_sensu_your_favorite_organism
name: gene_class
name: gene_part
name: gene_by_transcript_attribute
name: gene_by_polyadenylation_attribute
name: gene_to_gene_feature
name: gene_array_member
```

What we really want is the name that matches `gene` only. We can specifiy that as `gene` followed by the end of line. In pattern matching (and you will see details of this later) the *end of line* character may be specified as the `$` character.

```
cat so-simple.obo | grep 'name: gene$'
```

prints only one entry:

```
name: gene
```

The `-B` and `-A` flags make `grep` print the lines before and after a match. This can be very handy.

```
cat so-simple.obo | grep 'name: gene$' -B 1 -A 6
```

will print:

```
id: SO:0000704
name: gene
def: "A region (or regions) that includes all of the sequence elements necessary to encode a f
unctional transcript. A gene may include regulatory regions, transcribed regions and/or other
functional sequence regions." [SO:immuno_workshop]
comment: This term is mapped to MGED. Do not obsolete without consulting MGED ontology. A gene
 may be considered as a unit of inheritance.
subset: SOFA
xref: http://en.wikipedia.org/wiki/Gene "wiki"
is_a: SO:0001411 ! biological_region
relationship: member_of SO:0005855 ! gene_group
```

# How to search for other information?

While the website shows better formatted images you can mine the Sequence ontology at the command line for all kinds of content. Which records match the word `PCR` :

```
cat so-simple.obo | grep 'PCR' -B 2 -A 2
```

We get information of the form:

```
...
--
id: SO:0000345
name: EST
def: "A tag produced from a single sequencing read from a cDNA clone or PCR product; typically
 a few hundred base pairs long." [SO:ke]
comment: This term is mapped to MGED. Do not obsolete without consulting MGED ontology.
subset: SOFA
--
id: SO:0001481
name: RAPD
def: "RAPD is a 'PCR product' where a sequence variant is identified through the use of PCR wi
th random primers." [ZFIN:mh]
synonym: "Random Amplification Polymorphic DNA" EXACT []
is_a: SO:0000006 ! PCR_product
created_by: kareneilbeck
creation_date: 2009-09-09T05:26:10Z
...
```

# Gene ontology

## What is the Gene Ontology (GO)?

The Gene Ontology (GO) is a controlled vocabulary that connects each gene to one or more functions.

As it happens, the name "Gene Ontology" is somewhat misleading. The ontology is intended to categorize **gene products** rather than the genes themselves. Different products of the same gene may play very different roles, and labelling and treating all of these functions under the same gene name may (and often does) lead to confusion.

## How is the GO designed?

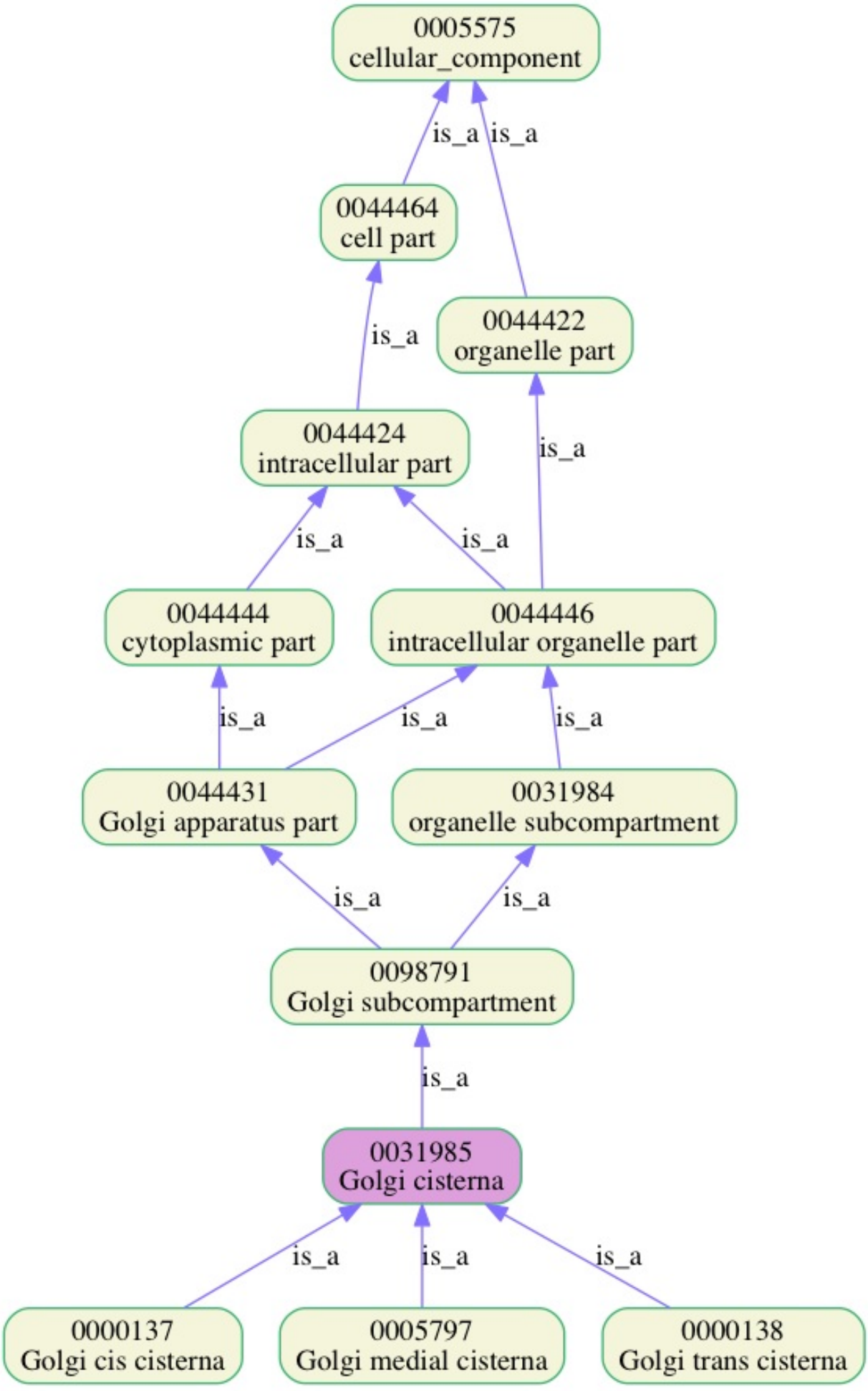
The GO project has developed three structured, independent sub-ontologies that describe gene products in a species-independent manner. The sub-ontologies are as follows:

- Cellular component (CC). **Where** does the product exhibit its effect? This ontology relates the gene product to a component of a cell, that is, part of a larger object, for example: *cell*, *nucleus*, *Golgi membrane*, *SAGA complex*
- Molecular function (MF). **How** does it work? Which biochemical mechanism characterizes the product? These terms describe activities that occur at the molecular level: *lactase activity*, *actin binding*
- Biological process (BP). **What** is the purpose of the gene product? The general rule to assist in distinguishing between a biological process and a molecular function is that a process must have **more than one** distinct step: *cholesterol efflux*, *transport*, *mitotic prophase*

## How are GO terms organized?

The GO ontology is structured as a directed, acyclic graph where each term has defined relationships to one or more other terms in the same domain, and sometimes to other domains. The GO vocabulary is designed to be species-agnostic, and includes terms applicable to prokaryotes and eukaryotes, and to single and multicellular organisms.

Here is the definition of the Golgi Cisterna ( `GO:0031985` ) visualized with goatools by Haibao Tang et al.

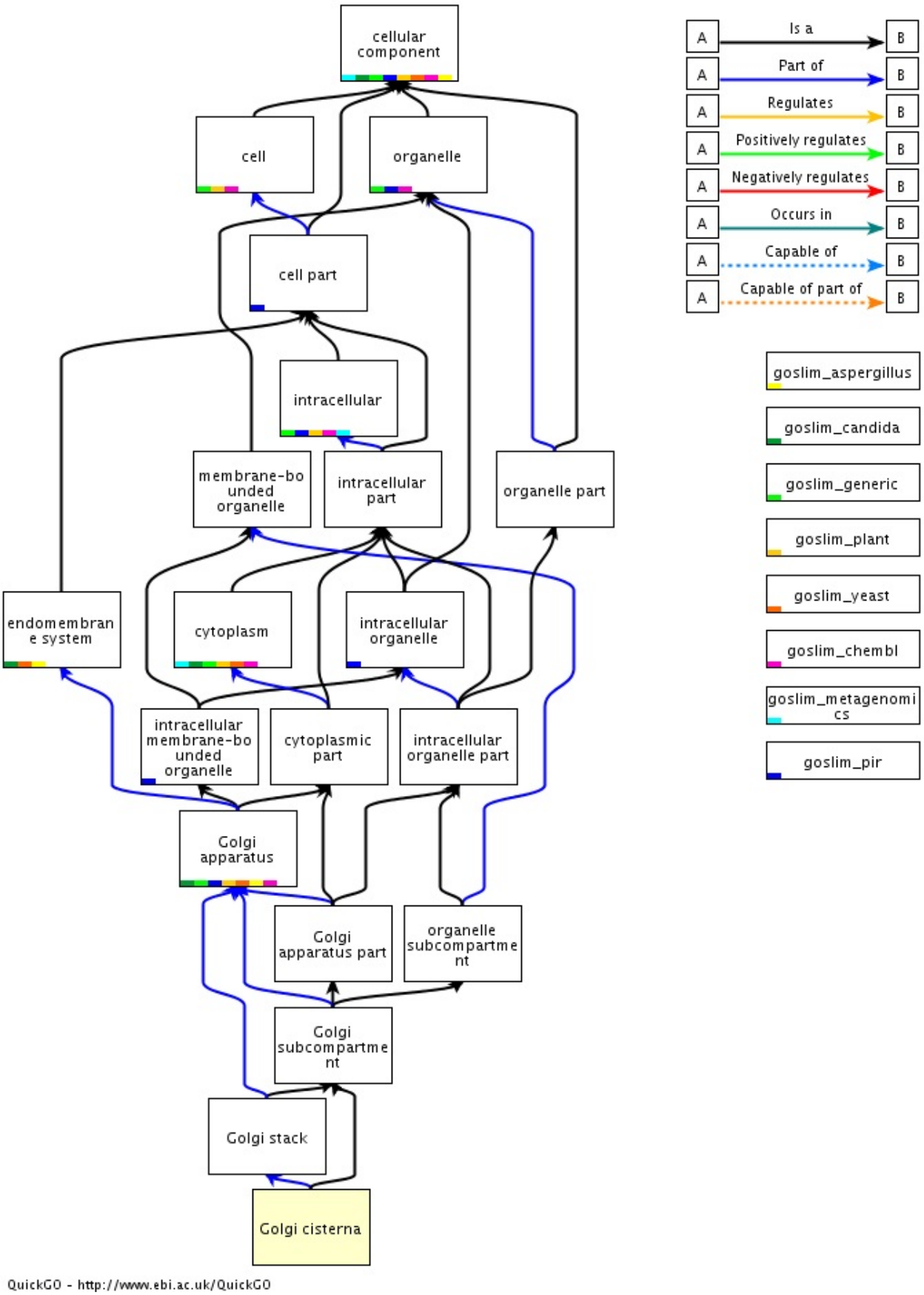

0005575
cellular_component
is_a
is_a
0044464
cell part
0044422
organelle part
is_a
is_a
0044424
intracellular part
is_a
is_a
is_a
0044444
cytoplasmic part
0044446
intracellular organelle part
is_a
is_a
is_a
0044431
Golgi apparatus part
0031984
organelle subcompartment
is_a
is_a
0098791
Golgi subcompartment
is_a
0031985
Golgi cisterna
is_a
is_a
is_a
0000137
Golgi cis cisterna
0005797
Golgi medial cisterna
0000138
Golgi trans cisterna

# Where can the Gene Ontology be viewed?

The Gene Ontology website is the authoritative source for definitions, but is not particularly well suited for data interpretation.

The Quick GO service from the European Bioinformatics Institute offers a web interface with more user-friendly functionality.

The same GO term Golgi Cisterna ( `GO:0031985` ) viewed in QuickGO shows




When compared to the other visualization, the images are similar, but they obviously do not represent identical information. This is a common theme when working with Gene Ontology. It is often not easy to identify how an annotation is being generated/filtered or expanded.

There are so called reduced (slim) vocabularies that simplify the data to broader terms at the cost of losing some of its lower level granularity.

# Understanding the GO data

While during regular use we may not need to access the original GO data it is worth spending just a bit a time to understand what it contains and how it is laid out. It will give you a different perspective and a deeper understanding on what is possible to infer in biology based on the way chose to store the data.

## What format is the GO data in?

GO data are available for each organism from the GO Download page. The underlying data for the gene ontology consists of two files:

1. Gene ontology definition file.
2. A gene association file.

These two pieces of information connect the ontology terms to gene product names. The GO files are surprisingly small and can be easily downloaded:

```
wget http://purl.obolibrary.org/obo/go.obo
```

## What does a GO term file contain?

The content of the core `go.obo` file is constructed of records in the form:

```
[Term]
id: GO:0000002
name: mitochondrial genome maintenance
namespace: biological_process
def: "The maintenance of the structure and integrity of
  the mitochondrial genome; includes replication and
  segregation of the mitochondrial chromosome." [GOC:ai, GOC:vw]
is_a: GO:0007005 ! mitochondrion organization
```

## What is a GO association file?

The 7 digit long GO ids have the form of `GO:xxxxxxx` . Then the gene association file for human gene products can be obtained as:

```
# Get the file.
wget http://geneontology.org/gene-associations/goa_human.gaf.gz

# Unzip the compressed file.
gunzip goa_human.gaf.gz
```

The file names may occasionally change (sometimes suddenly and with little prior notice) If the link is nonfunctional please visit the Gene Ontology download page and find and substitute the correct location.

# What format is the GO association file in?

It is the so called GAF format that is defined at: http://geneontology.org/page/go-annotation-file-gaf-format-21 that states:

### Annotation File Fields

The annotation flat file format is comprised of 17 tab-delimited fields.

| Column | Content | Required? | Cardinality | Example |
|---|---|---|---|---|
| 1 | DB | required | 1 | UniProtKB |
| 2 | DB Object ID | required | 1 | P12345 |
| 3 | DB Object Symbol | required | 1 | PHO3 |
| 4 | Qualifier | optional | 0 or greater | NOT |
| 5 | GO ID | required | 1 | GO:0003993 |
| 6 | DB:Reference (\|DB:Reference) | required | 1 or greater | PMID:2676709 |
| 7 | Evidence Code | required | 1 | IMP |
| 8 | With (or) From | optional | 0 or greater | GO:0000346 |
| 9 | Aspect | required | 1 | F |
| 10 | DB Object Name | optional | 0 or 1 | Toll-like receptor 4 |
| 11 | DB Object Synonym (\|Synonym) | optional | 0 or greater | hToll\|Tollbooth |
| 12 | DB Object Type | required | 1 | protein |
| 13 | Taxon(\|taxon) | required | 1 or 2 | taxon:9606 |
| 14 | Date | required | 1 | 20090118 |
| 15 | Assigned By | required | 1 | SGD |
| 16 | Annotation Extension | optional | 0 or greater | part_of(CL:0000576) |
| 17 | Gene Product Form ID | optional | 0 or 1 | UniProtKB:P12345-2 |

Comments are specified with `!` the rest are tab separated and column oriented data of the form:

```
UniProtKB   A0A024QZP7   CDC2  GO:0004672  GO_REF:0000002 ...
```

Here the first few columns indicate the gene product name, then its corresponding GO association is listed. Since GO terms have a hierarchical representation, a GO term for a gene name implies that the gene is annotated with all the parent nodes as well. The parent nodes typically describe the similar characteristics but in a broader sense.

In the gene association file, each gene product may be associated with one or more GO terms in each category.

# Do the association files represent most biological knowledge?

To some extent yes. While life sciences is much richer in general and the domain expertise of individuals can be lot deeper, the GO associations represent the "automatically searchable" dimension of this knowledge.

The `go.obo` file measures `33Mb` whereas the compressed gene association file is a mere `7Mb`. It is nothing short of mind-boggling to realize that these files are a road map to and product of an entire scientific field.

The GO files represent the accumulated and distilled results of the work of tens of thousands of scientists over many decades! Every automated, genome-based diagnosis, data interpretation, functional characterization, or genetic test relies on the content of this data. The life science data interpretation relies on Gene Ontology being correct, well annotated, and available to all.

# What kind of properties does the GO data have?

We will investigate the Gene Ontology.

```
# Get the GO data and their associations.
wget http://purl.obolibrary.org/obo/go.obo
wget http://geneontology.org/gene-associations/goa_human.gaf.gz

# How big are these files
ls -lh

# Uncompress the GO file.
gunzip goa_human.gaf.gz

# Remove the lines starting with ! and simplfy the file name.
cat goa_human.gaf | grep -v '!' > assoc.txt

# How many gene to association mappings?
cat assoc.txt | wc -l
# 477692 (this number changes when the data is updated)

# Gene names are in column 3.
cat assoc.txt | cut -f 3 | head -20

# Gene names are in column 3. Find the unique ones.
cat assoc.txt | cut -f 3 | sort | uniq -c | head -20

# How many unique genes are present?
cat assoc.txt | cut -f 3 | sort | uniq -c | wc -l
# 19713 (this number changes when the data is updated)
```

# What are the most annotated human genes and proteins?

The association file contains both gene and protein ids.

```
# What are the ten most highly annotated proteins in the GO dataset?
cat assoc.txt | cut -f 2 | sort | uniq -c | sort -k1,1nr > prot_counts.txt

# What are the ten most highly annotated gene names in the GO dataset?
cat assoc.txt | cut -f 3 | sort | uniq -c | sort -k1,1nr > gene_counts.txt
```

A cool little tool called `datamash` lets you do data analytics at command line.

```
datamash --help
```

Unfortunately the `uniq -c` command pads numbers with a variable number of spaces. We need to `squeeze` those into a single space.

`tr -s` can do that.

```
cat gene_counts.txt | tr -s ' '
```

Average number of annotations

```
# cat gene_counts.txt | tr -s ' ' | datamash -t ' ' mean 2
# 21.928170537048
```

The mean, minimum, maximum and standard deviation:

```
cat gene_counts.txt | tr -s ' ' | datamash -t ' ' mean 2  min 2 max 2 sstdev 2
# 21.928170537048 1 724 31.285153079175
```

# What are the ten most highly annotated genes in the GO dataset?

```
cat gene_counts.txt | head
```

Produces:

```
724 TP53
669 GRB2
637 EGFR
637 UBC
580 RPS27A
570 UBB
565 UBA52
511 CTNNB1
422 SRC
```

Interestingly in December 2016, when we first wrote this chapter the list was the following:

```
1337 HLA-B
 918 HLA-A
 819 HLA-DRB1
 789 TP53
 ...
```

Today the `HLA-B` gene produces a radically different output:

```
cat gene_counts.txt | grep HLA-B
# 44 HLA-B
```

# Do the GO annotations change?

The latest GO data download produces a very different report than the above. Notably HLA genes are not in the top ten anymore. It reinforces the ephemeral nature of annotations - as new information or different filtering paramters are applied the information content can drastically shift. Why did the HLA genes dissapear? Was that an error. Probably... but what does it mean for the publications that we researched during that time that the error was present? Was anyone ever publicly notified about this? We discovered this discrepancy by accident... There it goes to show how not everything is all and well in the GO world...

*The following is a description to a discovery process that may not apply anymore but is educational to follow it through.*

As we can see, the most annotated protein is `P04637`, a cellular tumor antigen p53, which acts as a tumor suppressor in many tumor types. It corresponds to the gene `TP53`.

The most annotated gene was `HLA-B`. As it happens, this is a gene that we were unable to find right away when searching for it on the official gene name nomenclature (HGNC).

It is one of those `&$!*&@!@` moments that every bioinformatician will have to learn to deal with:

> The most annotated gene in the GO seemingly cannot be found when searching for it in HGNC, the repository that is supposed to be the authoritative resource for the domain!

The cause is both simple and infuriating: the search for `HLA-B` finds over a thousand entries related to `HLA-B` but the gene that is actually named `HLA-B` turns up only on the second page of hits, buried among entries distantly related to it.

Situations like this, where seemingly mundane tasks suddenly require problem solving, are very common in bioinformatics. Keep that in mind the next time something completely surreal seems to happen.

An alternative and much better maintained resource GeneCards allows you to locate the gene by name right away.

# How complete is the GO?

We may wonder: how complete are these GO annotations? Would it be possible to estimate what percent of all functional annotations have been discovered so far?

While that question is too abstract for our liking, we could tabulate the dates assigned to each evidence and see how many pieces of evidence are produced per year.

It is "reasonably simple" to extract and tabulate the date information from the two files that we downloaded to get the rate of information growth in the GO:

```
# Don't worry about this command if you don't get it yet.
# We'll cover it later but we just want to demonstrate some results.
cat assoc.txt \
| cut -f 14 \
| awk '{ print substr($1, 1, 4) }' \
| sort \
| uniq -c \
| sort -k 2,2 -n \
| awk '{ print  $2 "\t" $1 }'
```

Produces the number of annotations verified in a given year.

```
...
2008    9682
2009    13322
2010    15026
2011    23490
2012    16428
2013    60555
2014    34925
2015    33096
2016    235077
```

This result is quite surprising, and we don't quite know what to make of it. Is seems that most evidence is assigned to the latest year - `2016`. Or maybe there is some sort of quirk in the system that assigns the latest date to every evidence that is being re-observed? We don't know.

There is a also a suprising dip in the number of annotations in `2012`, then a subsequent bump in `2013`.

# Functional analysis of bioinformatics data

The content of this chapter follow the ideas presented in Ten Years of Pathway Analysis: Current Approaches and Outstanding Challenges from 2012, Plos Computational Biology.

For the majority of people, the reason for studying bioinformatics is to make sense of the data. We all want to gain insights into biological phenomena.

Surprisingly enough, whereas significant resources are spent on gathering data, scientists are unexpectedly short-sighted when it comes to funding the organization and categorization of the information that has been collected. It seems that in their minds this is a solved problem: "just put it in a database." In reality organizing existing knowledge is difficult, thankless and unappreciated effort - that everyone needs dearly, but few can understand the magnitude and complexity of it. It is a manifestation of the so called tragedy of commons where: "individual users acting independently according to their self-interest behave contrary to the common good."

As an example, the Kyoto Encyclopedia of Genes and Genomes (KEGG) database had to change its licensing and started to charge thousands of dollars for bulk data access as they were unable to fund their service otherwise.

Companies have recognized the bottleneck and the commercial potential of the inefficiencies of the process. Many offer services that solve the increasingly painful process of summarizing data. Of course, the services that they offer come with eye popping pricing and are practically inaccessible when it comes to students, publicly funded groups or citizen scientists. This is not to say that the services might not be worth the price when it comes to large scale organizations. We want to make it clear that we are not against commercial services of any kind. The concerning issue is the increasingly difficult access to data that should be public.

# What is an Over-Representation Analysis (ORA)?

An Over-Representation Analysis (ORA) attempts to find representative functions of a list of genes. It does this by comparing the number of times a function is observed to a baseline.

The publication that we mention above summarizes `ORA` as:

> ... over-representation analysis (ORA), which statistically evaluates the fraction of genes in a particular pathway found among the set of genes showing changes in expression. It is also referred to as “2×2 table method” in the literature. ORA uses one or more variations of the following strategy: first, an input list is created using a certain threshold or criteria. For example, a researcher may choose genes that are differentially over- or under-expressed in a given condition at a false discovery rate (FDR) of 5%. Then, for each pathway, input genes that are part of the pathway are counted. This process is repeated for an appropriate background list of genes (e.g., all genes measured on a microarray).

In a nutshell, and in a somewhat simplified explanation, an ORA analysis computes four numbers:

1. How many functions do all genes have in total?

2. How many should I observe by random chance if I were to select a subset (of the same size as the current subset)?
3. How many functions do my selected (subset) genes have?
4. How many functions did I observe in my subset?

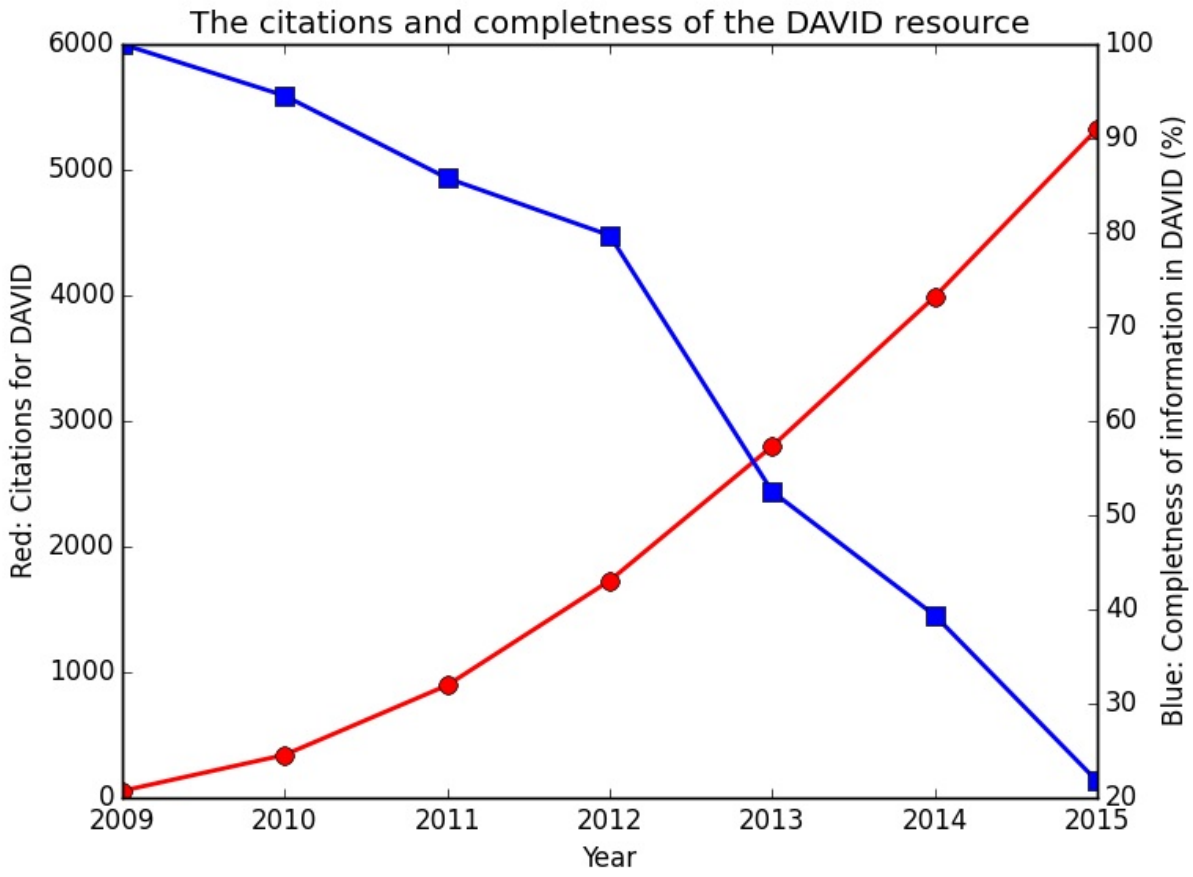
An `ORA` analysis will compute how likely is that the fourth number could be obtained by random chance knowing the other three.

# Are there different ways to compute ORA analyses?

If you followed the section on Gene Ontology, you know that the GO files have a relatively simple format to them. Moreover, the data structure is that of a (network) tree. Building a program that assigns counts to each node and can traverse the tree is a standard problem that requires only moderately advanced programming skill.

At the same time, data interpretation is such an acute problem there always seems to be a shortage of tools that can perform the ORA analysis in one specific way. Hence a cottage industry of ORA tools exists, with hundreds of such tools - the vast majority abandoned, failing or even worse giving the wrong answer tacitly.

Moreover, the accuracy of a method will critically depend on it integrating the most up to date information. As a tale of caution, we note that the DAVID: Functional Annotation Tool was not updated from 2010 to 2016! Over this time it was unclear to users whether the data that the server operates on includes the most up to date information. New functions and annotations are added on an almost weekly basis, over the many years this difference increased considerably. We believe that by the end of the 2016 DAVID operated on a mere 20% of the total information t known in biology. Nevertheless, it had gained thousands of citations every single year.




As you can see above in 2015 alone, DAVID collected over 5000 citations, yet at that time it contained only 20% of the known annotations. The "good" news is that there was an update in 2016 so for at least a little bit it is not as far behind as it used to.

The sorry state of DAVID has been a sore point for many. The "update" might have been in response to growing chorus of complaints:

- Impact of outdated gene annotations on pathway enrichment analysis Nature Methods, 2016
- Several Biostar posts: Is the DAVID knowledgebase not updated anymore?, then GO analysis: DAVID vs GREAT vs GOrilla, and Finally the much awaited DAVID update has taken place

# What are problems of the ORA analysis?

The shortcomings of the overlap analysis are that:

1. ORA analysis does not account for the magnitude of expression levels. The gene is either in the list or not.
2. ORA typically uses only a subset of genes - the cutoffs are "arbitrary" in the sense that they are based on convention rather than an objective measure.
3. Genes and functions are all considered independent of one another. For statistical assumptions to work, this is an essential requirement. If the independence constraint does not hold then the mathematical basis for the test does not hold either. As we all know many functions in the cell are strongly interdependent.

# Why do we still use the ORA analysis?

Clearly, as you can see above the method has many pitfalls. Why do we still use it?

I think the reason for this is that the concept is simple, quick to compute, and most scientists understand how it is calculated. For many other techniques the internal workings are so complicated and "black-box"-like that often you will be left with nothing more than a belief that that box probably does what it claims to.

In our opinion, ORA analyses should be usd as a hypothesis generation and testing framework and not as a method that provides the final answer to a problem.

# What is a Functional Class Scoring (FCS)?

Let's see how the paper summarizes it:

1. "A gene-level statistic is computed using the molecular measurements from an experiment."
2. "The gene-level statistics for all genes in a pathway are aggregated into a single pathway-level statistic."
3. "The final step in FCS is assessing the statistical significance of the pathway-level statistic."

> FCS methods use this information to detect coordinated changes in the expression of genes in the same pathway. Finally, by considering the coordinated changes in gene expression, FCS methods account for dependence between genes in a pathway, which ORA does not.

Now you can see another reason of why ORA analysis is still used. At least we understand what the ORA does - whereas the text above could be applied broadly and generically to cover a great variety of techniques - most of which could be complicated and nearly impossible to understand for a life scientist. When people use an FCS tool they become mere "passengers" of it - they run the tool then hope for the best that it takes them where they want to go.

There are many implementations of each method and approach some more explicit than others - do refer to the paper for a list.

# Gene set enrichment

## What is a gene set enrichment analysis?

"Gene set enrichment analysis" refers to the process of discovering the common characteristics potentially present in a list of genes. When these characteristics are GO terms, the process is called "functional enrichment".

There are a few variants of gene set enrichment analysis. One of the most commonly used is *over-representation analysis (ORA)*. An ORA examines the genes in a list, summarizes the GO annotations for each, then determines whether there any annotations are statistically over-represented (compared to an expectation) in that list.

## What tools are used to perform enrichment analysis?

Of all bioinformatics tools developed to-date, the GO enrichment tools are in the worst shape. Many are in various states of abandon. Unlike other software that, once implemented correctly, keeps working over the years, a GO tool's analysis depends, critically, on its using the most up-to-date GO annotations. Hence, GO tools require ongoing maintenance and effort, which their owners are often unable to provide.

Most enrichment tools seem to incorporate the word "GO" into various "humorous" tool names, like `GOOSE`, `GOrilla`, `AmiGO`, `BINGO`, `QuickGO`. But the joke gets old very quickly, as most of these tools are surprisingly hard to use and/or produce inconsistent results. We recommend lots of caution when using GO-based tools, as all are notoriously difficult to evaluate for correctness.

Over the years, we have grown to trust the following tools:

- AgriGO Web-based GO Analysis Toolkit and Database for Agricultural Community.
- DAVID This is the GO tool biologists love. It is the "most generous" of them all, as it produces copious amounts of output. DAVID was nearly abandoned for more than 6 years, but was finally updated in 2016. In the meantime (between 2009 and 2016), thousands of publications used it to produce GO enrichment analyses on increasingly outdated GO data.
- Panther is offered directly from the GO website. It produces very limited information and no visualization.
- goatools a command line Python scripts.
- ermineJ Standalone tool with easy to use interface. It has detailed documentation.
- GOrilla Web-based; good visualization; downloadable results (as Excel files); easily see which genes contribute to which enriched terms; results pages indicate date of last update GO database (often within the last week).
- ToppFun - a suite of tools for human genomes that integrates a suprising array of data sources.

## Will different tools produce different results?

Biostar Quote of the Day: Why does each GO enrichment method give different results?

> I'm new to GO terms. In the beginning it was fun, as long as I stuck to one algorithm. But then I found that there are many out there, each with its own advantages and caveats (the quality of graphic representation, for instance).
>
> [...] As a biologist, what should I trust? Deciding on this or that algorithm may change the whole story/article!"

This is a ongoing problem with no clear solution. Unlike most other computational approaches where there is an objective way to validate the quality of a result, in most GO analyses we don't know how to tell when it worked correctly.

Overall GO enrichment is surprisingly subjective and different methods will produce different results. Note that two different results could be both correct. Or not...

# How do I perform a gene set enrichment analysis?

Most tools follow the same principles:

1. Enter the group of gene names to be studied.
2. Enter the group of genes that are the "background" (perhaps all genes for this organism)
3. Select a statistical method for comparison.

For example, let's find genes that match `HLA` (see the chapter on GO on how to get `assoc.txt` )

If we input

```
cat assoc.txt | cut -f 3 | grep HLA |sort | uniq | head
```

it produces this list:

```
HHLA1
HHLA2
HHLA3
HLA-A
HLA-B
HLA-C
HLA-DMA
HLA-DMB
HLA-DOA
HLA-DOB
```

Enter this list into a GO enrichment tool shown here for Panther via the main GO website:

# Enrichment analysis

HHLA1
HHLA2
HHLA3
HLA-A
HLA-B
HLA-C
HLA-DMA
HLA-DMB
HLA-DOA
HLA-DOB

biological process

Homo sapiens

Submit

Displaying only results with P<0.05; click here to display all results

| | Homo sapiens (REF) | upload 1 (▼ Hierarchy NEW!) | | | | |
|---|---|---|---|---|---|---|
| GO biological process complete | # | # | expected | Fold Enrichment | +/- | P value |
| negative regulation of antigen processing and presentation of peptide antigen via MHC class II | 2 | 2 | .00 | > 100 | + | 3.33E-03 |
| └regulation of antigen processing and presentation of peptide antigen via MHC class II | 5 | 2 | .00 | > 100 | + | 2.08E-02 |
| └regulation of immune system process | 1397 | 7 | .67 | 10.51 | + | 4.76E-03 |
| └regulation of antigen processing and presentation of peptide or polysaccharide antigen via MHC class II | 6 | 2 | .00 | > 100 | + | 3.00E-02 |
| └negative regulation of antigen processing and presentation of peptide or polysaccharide antigen via MHC class II | 3 | 2 | .00 | > 100 | + | 7.50E-03 |
| └negative regulation of antigen processing and presentation | 6 | 2 | .00 | > 100 | + | 3.00E-02 |
| └negative regulation of antigen processing and presentation of peptide antigen | 4 | 2 | .00 | > 100 | + | 1.33E-02 |
| antigen processing and presentation of endogenous peptide antigen via MHC class I via ER pathway, TAP-independent | 3 | 3 | .00 | > 100 | + | 2.86E-06 |
| └antigen processing and presentation of endogenous peptide antigen via MHC class I via ER pathway | 10 | 3 | .00 | > 100 | + | 1.06E-04 |
| └antigen processing and presentation of endogenous peptide antigen via MHC class I | 15 | 3 | .01 | > 100 | + | 3.57E-04 |
| └antigen processing and presentation of peptide antigen | 179 | 7 | .09 | 82.01 | + | 3.16E-09 |
| └antigen processing and presentation | 216 | 7 | .10 | 67.96 | + | 1.17E-08 |
| └immune system process | 2013 | 8 | .96 | 8.33 | + | 2.21E-03 |
| └antigen processing and presentation of endogenous peptide antigen | 16 | 3 | .01 | > 100 | + | 4.33E-04 |
| └antigen processing and presentation of endogenous antigen | 19 | 3 | .01 | > 100 | + | 7.24E-04 |
| peptide antigen assembly with MHC class II protein complex | 4 | 2 | .00 | > 100 | + | 1.33E-02 |
| └peptide antigen assembly with MHC protein complex | 5 | 2 | .00 | > 100 | + | 2.08E-02 |
| └MHC protein complex assembly | 6 | 2 | .00 | > 100 | + | 3.00E-02 |
| └antigen processing and presentation of peptide antigen via MHC class II | 94 | 4 | .04 | 89.24 | + | 6.76E-04 |
| └antigen processing and presentation of peptide or polysaccharide antigen via MHC class II | 98 | 6 | .05 | > 100 | + | 1.75E-08 |
| └MHC class II protein complex assembly | 4 | 2 | .00 | > 100 | + | 1.33E-02 |
| antigen processing and presentation of exogenous peptide antigen via MHC class I, TAP-independent | 9 | 3 | .00 | > 100 | + | 7.71E-05 |
| └antigen processing and presentation of exogenous peptide antigen via MHC class I | 66 | 3 | .03 | 95.33 | + | 3.00E-02 |
| └antigen processing and presentation of exogenous peptide antigen | 163 | 7 | .08 | 90.06 | + | 1.64E-09 |
| └antigen processing and presentation of exogenous antigen | 170 | 7 | .08 | 86.36 | + | 2.20E-09 |
| protection from natural killer cell mediated cytotoxicity | 6 | 2 | .00 | > 100 | + | 3.00E-02 |
| type I interferon signaling pathway | 63 | 3 | .03 | 99.87 | + | 2.61E-02 |
| └cellular response to type I interferon | 63 | 3 | .03 | 99.87 | + | 2.61E-02 |
| └response to type I interferon | 67 | 3 | .03 | 93.90 | + | 3.14E-02 |
| └immune response | 1105 | 7 | .53 | 13.29 | + | 9.57E-04 |
| antigen processing and presentation of exogenous peptide antigen via MHC class I, TAP-dependent | 63 | 3 | .03 | 99.87 | + | 2.61E-02 |
| antigen processing and presentation of exogenous peptide antigen via MHC class II | 92 | 4 | .04 | 91.18 | + | 6.21E-04 |

# Using the g:Profiler

## What is the g:Profiler?

A useful website sporting a counterintuitive and somewhat confusing naming scheme, the `g:Profiler` is a suite of several tools.

For the full details we recommend consulting the publication g:Profiler -- a web server for functional interpretation of gene lists Nucleic Acids Research 2016;

The service can be accessed at:

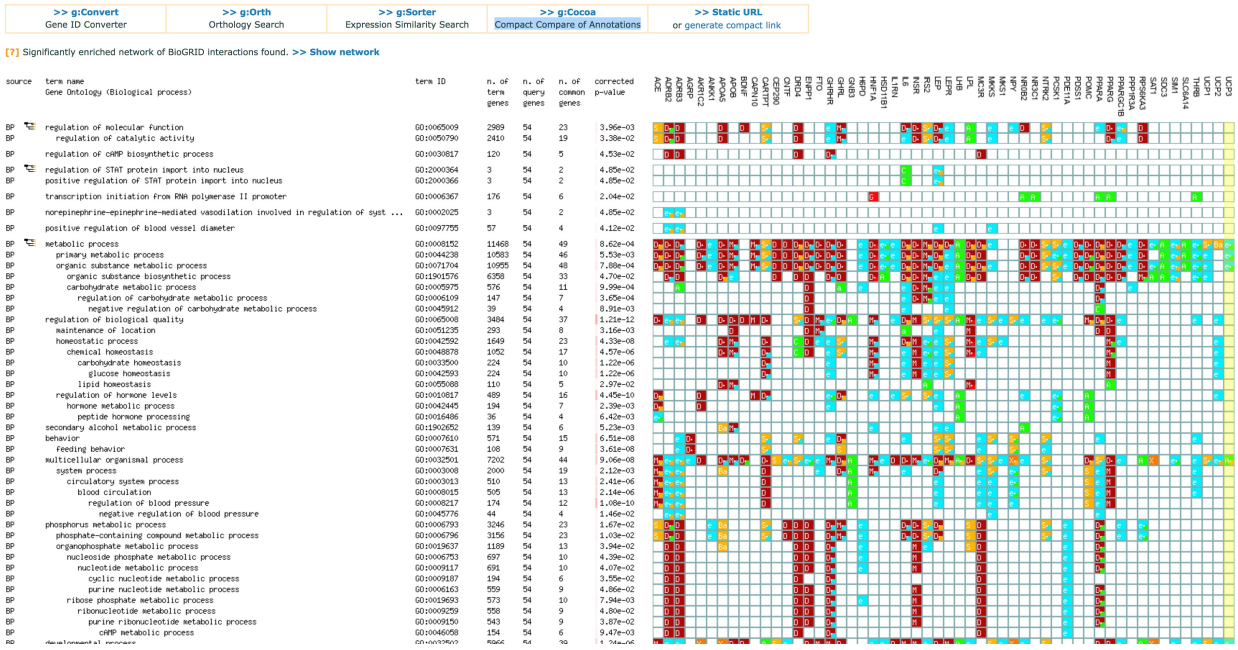
- http://biit.cs.ut.ee/gprofiler

The `g:Profiler` can perform: "functional enrichment analysis and mine additional information. These tools analyse flat or ranked gene lists for enriched features (g:GOSt; g:Cocoa), convert gene identifiers of different classes (g:Convert), map genes to orthologous genes in related species (g:Orth) and find similarly expressed genes from public microarray datasets (g:Sorter). An additional tool g:SNPense maps human single nucleotide polymorphisms (SNP) to gene names, chromosomal locations and variant consequence terms from Sequence Ontology"

## What is a standout feature of the g:Profiler?

In our the `g:Profiler` visualizes the evidence code for GO terms in the clearest way.

As an example load the following gene-list.txt into the g:Profiler and view the results.




## What functionality does the g:Profile have?

## g:GOSt

Performs functional enrichment analysis:

> **g:GOSt** performs pathway enrichment analysis and is the central tool of the g:Profiler web server. It maps a user-provided gene list to various sources of functional information and determines significantly enriched pathways, processes, and other annotations.

## g:COCOA

COCOA (Compact Compare of Annotations) offers simultaneous enrichment analysis of multiple gene lists:

> **g:Cocoa** provides means to analyse several gene lists at the same time and compare their characteristic enriched terms. This is useful in scenarios where an experimental design involves many comparisons of samples or individuals, or when one wants to compare directly different clusters of genes arising from the analysis. Each gene list is analysed for functional enrichments similarly to g:GOSt and resulting terms are aligned vertically into a matrix highlighting strongest findings for every gene list.

## g:Convert

Allows for automatic conversion of gene identifiers.

> **g:Convert** provides a convenient service to translate identifiers (IDs) of genes, proteins, microarray probesets and many other types of namespaces. The seamless translation process works on a mixed set of diverse identifiers and maps these through Ensembl gene identifiers (ENSG) as reference. In cases of multiple identifiers, all relevant combinations are highlighted. At least 13 types of IDs are supported for all of the 213 species available in g:Profiler, and at least 40 types of IDs for more than 50 species.

## g:Orth

Maps related genes across species

> **g:Orth** allows the user to map a list of genes of interest to homologous genes in another related organism. Many experiments are conducted in model organisms and knowledge from such experiments is transferred to other organisms to compare or complement previous findings. g:Orth uses g:Convert to map gene IDs to Ensembl ENSG identifiers. Further mapping to orthologous genes in other organisms is also based on Ensembl data. We provide cross-references between all organisms in g:Profiler. Queries are limited to tiers according to classes of species (animals, plants, fungi).

## g:Sorter

Findins similar genes in transcriptomics data

> **g:Sorter** is a tool for finding lists of co-expressed genes from public transcriptomics datasets. Thousands of microarray and RNA-seq experiments have been conducted in the past decades. The majority of published studies have been accumulated in databases like ArrayExpress and Gene Expression Omnibus. We have downloaded 7878 datasets for 18 species from ArrayExpress and provide gene co-expression similarity searches using six most common statistical methods. The datasets can be searched rapidly with keywords. The input of g:Sorter is a single gene and a dataset of interest and the result is a sorted list of genes that are similarly expressed with the gene of interest. These lists can be integrated into functional enrichment analysis. For comprehensive global gene expression similarity queries, as well as support for more species and platforms we suggest to use Multi Experiment Matrix (MEM) tool

## g:SNPense

SNP identifier mapping

> With the rapid growth of whole genome sequencing technology, researchers are uncovering extensive genetic variation and large collections of known SNP are available for human and other species. In order to easily map SNP identifiers (e.g. `rs4244285`) to gene names, chromosomal coordinates and retrieve their functional consequences we now provide a new service called g:SNPense. Information about genome variants is retrieved from dbSNP and mapped to NCBI Genes. Potential functional consequences of the variants are retrieved from Ensembl Variation data and grouped into 35 Sequence Ontology terms of various severity. g:SNPense is a potential entry point to g:GOSt and its functional annotation pipeline and enrichment analysis.

# The Database for Annotation, Visualization and Integrated Discovery (DAVID)

Authors: Aswathy Sebastian, Istvan Albert

## What is DAVID?

DAVID is a functional enrichment analysis tool that can be used to understand the biological relevance of a list of genes.

The acronym DAVID stands for *The Database for Annotation, Visualization, and Integrated Discovery* (DAVID). The tool allows users to:

- Identify functionally important genes in a gene list.
- Get the enriched biological terms in a gene list.
- Obtain the biological processes that a set of up-regulated and down-regulated genes are involved in.

The tools is a valuable resource - but as many other scientific software, it suffers from very confusing interface elements and redundant windows that pop over others. It is very easy to get disoriented while using it.

## What does DAVID do ?

When presented a gene list like the one shown below

```
Tmem132a
Myl3
Myl4
Tnnt2
Hspb7
Tnni3
Actc1
...
```

DAVID will generate functional interpretations:

- 14 genes (My13, My14, Tnni3,...) from this list are involved in *Cardiac Muscle Contraction*
- 4 genes (Actc1, Myh6, Myh7, Tnnt2) have the role of *ATPase Activity*

and so on. The value added by the tools is that it helps find groups of genes that share certain properties that might be interesting to a life scientist.

## What are the different steps in a DAVID analysis?

A typical gene enrichment and functional annotation workflow in DAVID has the following steps:

1. Load a gene list into the site.
2. Explore details through the annotation table, chart or clustering reports.
3. Export and save the results.

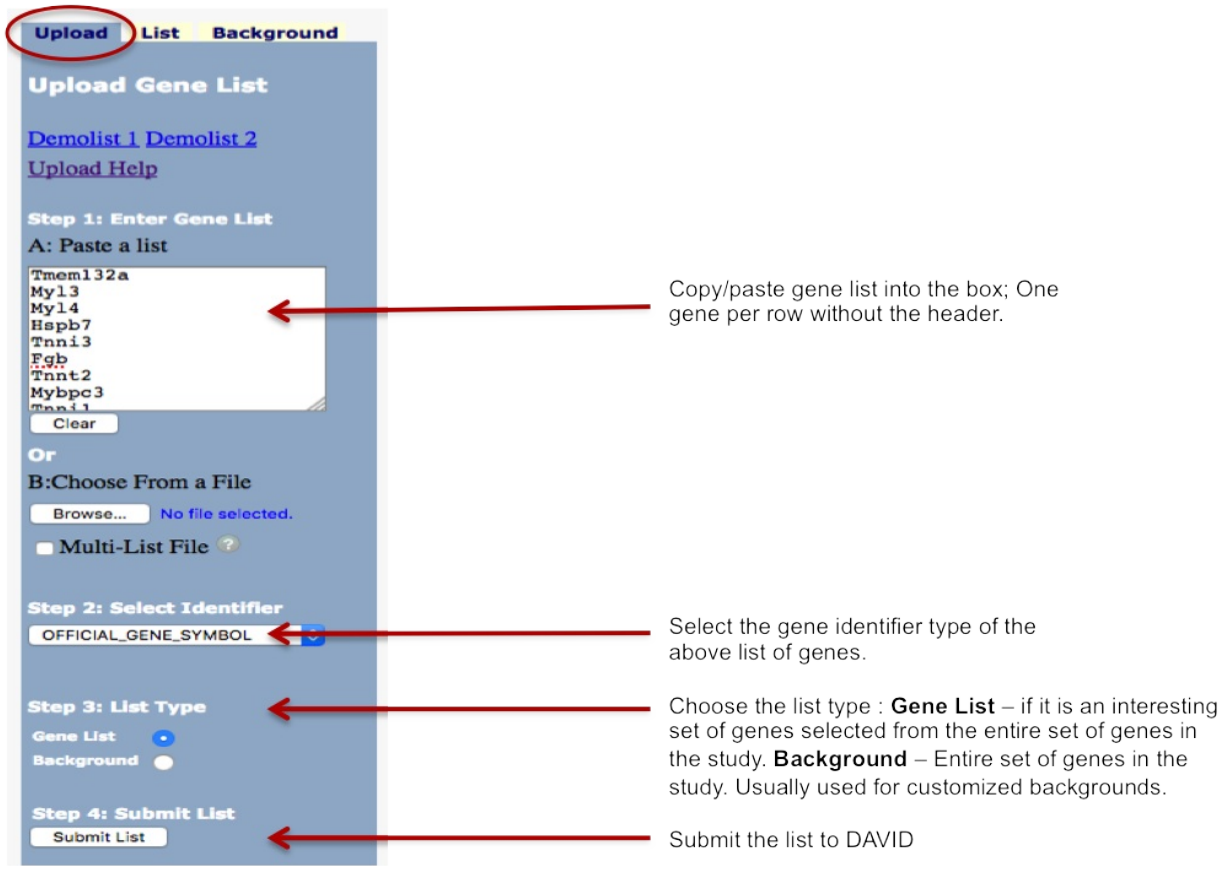
# How do I start an analysis with DAVID ?

Every analysis in DAVID starts with the submission of a gene list.

A gene list is a selected set of genes from an experiment that you might be interested in learning more about. This gene list will be evaluated against a so-called "background" list to detect attributes that are only present in the gene list.

Often, the background is taken as all genes of an organism, but, in some cases, it is more appropriate to use a different gene set as background. For example, if the list of interesting genes was taken from a microarray experiment, the background should be only those genes queried by the microarray. Gene lists from sequencing assays generally can use all the organism's genes as the background.

To start an analysis, users can either click on the 'Start Analysis' tab or choose one of the tools from 'Shortcut to DAVID Tools' tab. Both will take you to the upload wizard:




An example gene list can be obtained using the commands below. Here a mouse genelist is selected for DAVID analysis by using an expression p-value cut off of 0.05.

```
curl -O http://data.biostarhandbook.com/rnaseq/mouse-gene-expression.txt
cat mouse-gene-expression.txt |  awk '$2 <0.05 {print $1}' >genelist.txt
```

Once the list is submitted, a selection of the input species might be needed in case there are ambiguities and the input gene names may refer to multiple species.

A note on what is happening behind the scenes when a gene list is submitted:

- Each input gene id gets converted into a so-called *DAVID Gene ID*, a unique identifier specific to DAVID. All the subsequent analysis will be applied to the DAVID IDs in the selected gene list.
- The DAVID Gene ID connects the input genes across all other databases in DAVID. DAVID supports nearly 30 different input id types and a broad set of of annotation categories.

Once the gene list is submitted, you will need to explore the submitted gene list with DAVID tools and their results are named somewhat redundantly and confusingly (at least we have a hard time remembering which one does what):

- Functional Annotation.
- Gene Functional Classification.

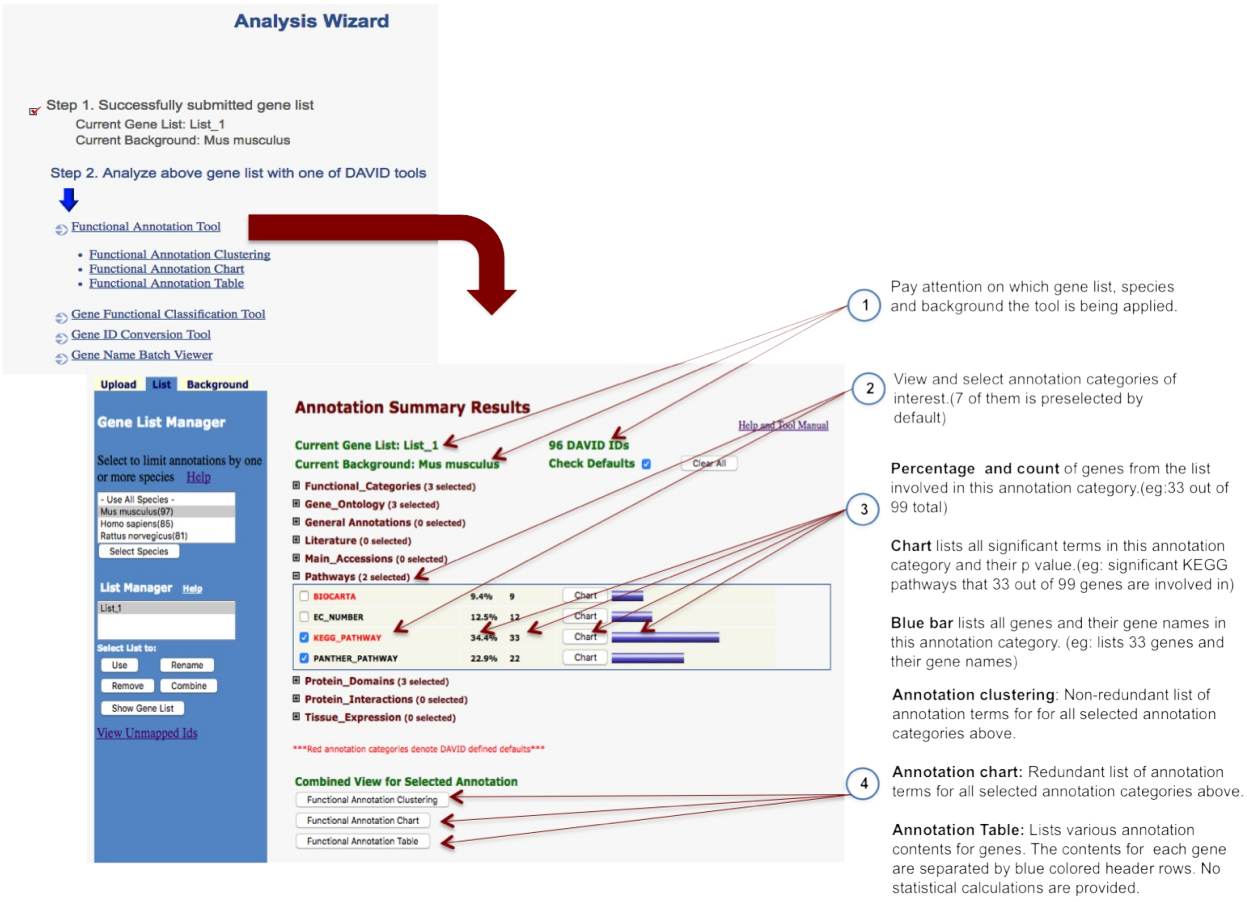
# What is the Functional Annotation Tool?

These tools assist users in the biological interpretation of their data. It maps the genes to annotation content from different databases (that can be selected by users). DAVID computes the enriched functionality of the gene list.

The results of a Functional Annotation process is a Functional Annotation Summary where the user has options to further select annotation categories of interest and explore them in detail.

# What is the Functional Annotation Summary?

The selections of the Functional Annotation Tool and Annotation Summary page are shown below:




Expanding on each selected category gives the annotation details of the genes involved in it.

The combined view allows you to explore the combined annotation content of all selected categories.

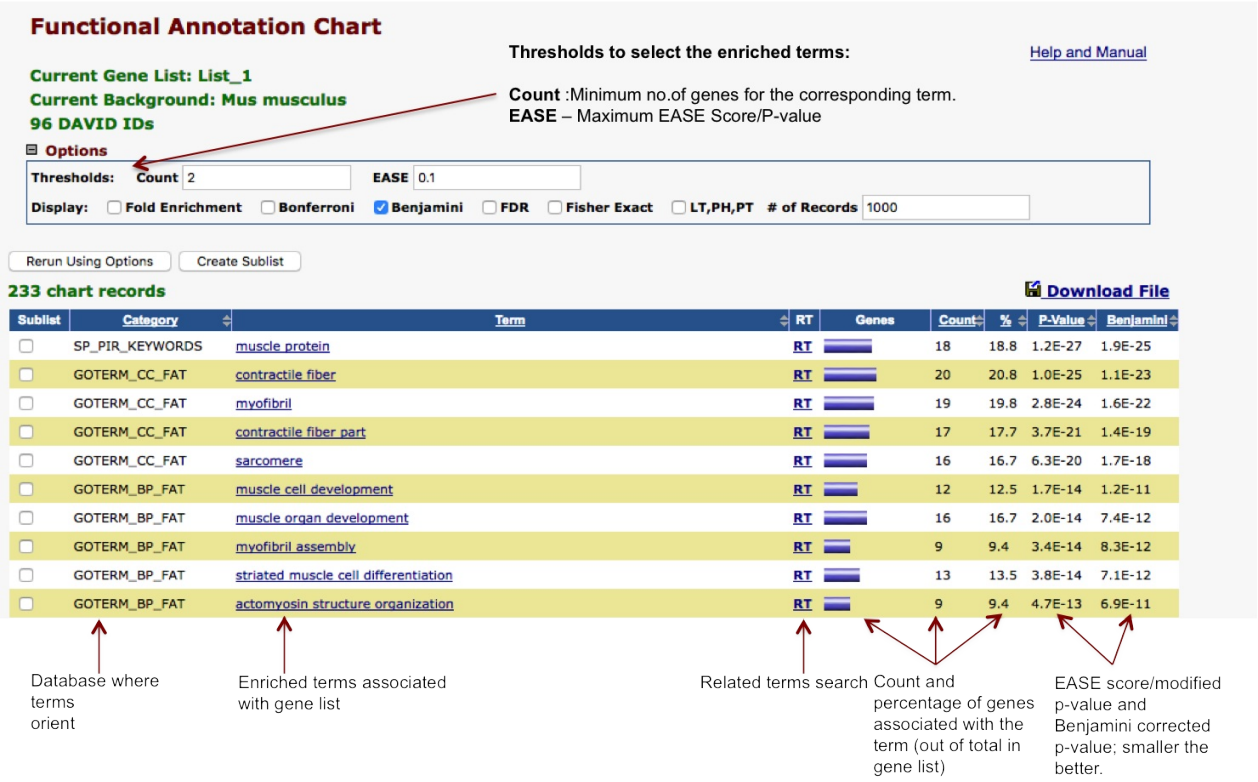
# How do I explore the Functional Annotation Summary?

The selected functional categories of interest in the Functional Annotation Summary page can be analyzed through any of the following reports:

1. **Functional Annotation Chart:** A view focused on annotation terms that lists all enriched annotation terms and their associated genes.
2. **Functional Annotation Clustering:** Groups similar annotation terms into clusters based on their similarity and the genes they share in common.
3. **Functional Annotation Table:** Gene-centric view that lists all genes and their associated annotation terms (selected only). No statistics are provided here.

# What is a Functional Annotation Chart ?

The functional annotation chart is a tabular form of all enriched annotation terms associated with a gene list. The enriched annotation terms presented here can be redundant. The enrichment P-value is the so-called *Ease Score*, calculated via Fisher's Exact test.

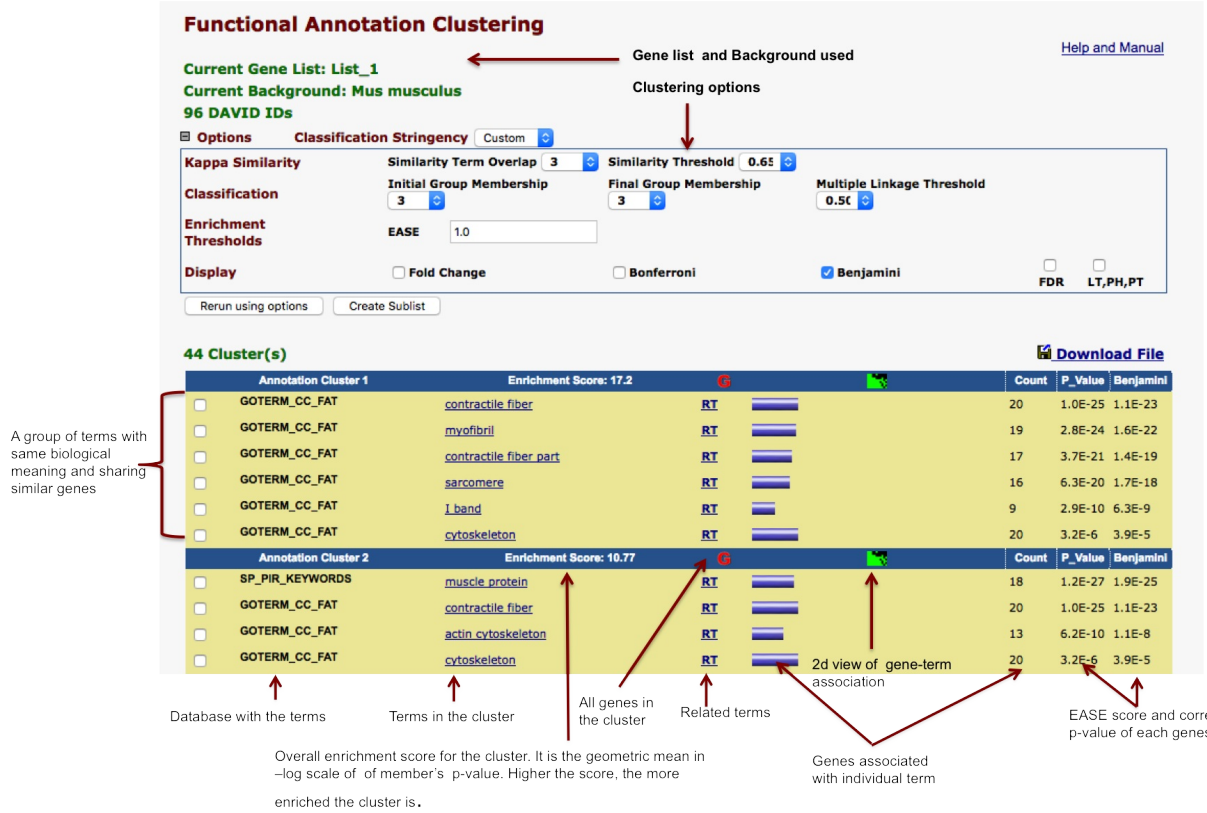



In the example shown here, 233 annotation terms are associated with 96 input genes. 18 of these 96 genes are involved in muscle protein functionality. Clicking on the blue bar lists these genes. The related terms (RT) search shows other annotation terms that these 18 genes are associated with. In other words, RT lists all annotation terms above a certain similarity threshold that are shared by these genes.

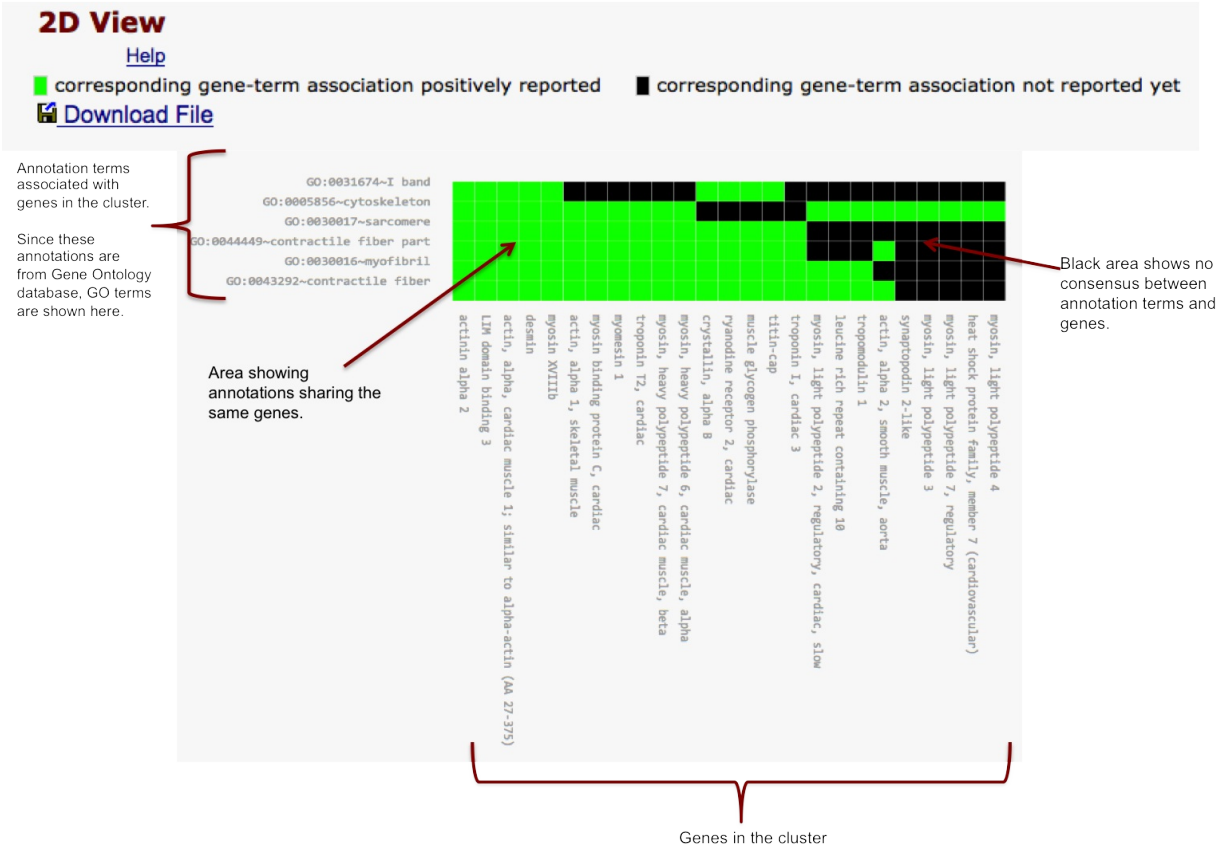
# What is Functional Annotation Clustering?

Annotation terms associated with genes can be redundant. DAVID measures the degree of similarity between these terms using kappa statistics based on the number of genes they share. All the terms above a certain similarity threshold are then grouped together using a clustering algorithm. Thus each annotation cluster groups together terms with similar biological meaning.

The clustering report below shows that 233 enriched annotation terms associated with 96 DAVID input ids are clustered into 44 groups.

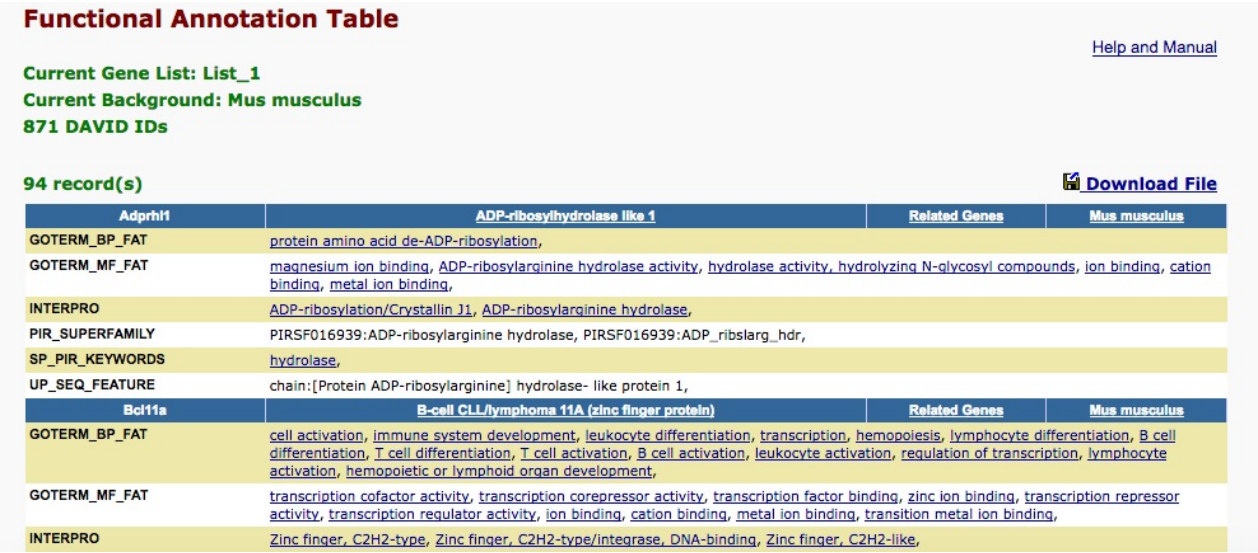



Cluster1 in the example shown here has 6 annotation terms that share the associated genes. The relationship between the genes and these annotation terms is shown in the image below. The green area represents the sharing of genes by these annotation terms. The black area denotes that there is no association between the terms and the genes. A large green area means that more genes are shared, implying a greater similarity between these terms. The terms are thus grouped together in a single cluster.



# What is a Functional Annotation Table?

The Functional Annotation Table is a third form of reporting the annotation content (selected in annotation summary) of a gene list. It provides a gene-centric view by listing the genes and their associated terms. No statistics are reported in this page. Each gene is separated by a blue header row.



**Functional Annotation Table**

Help and Manual

**Current Gene List: List_1**
**Current Background: Mus musculus**
**871 DAVID IDs**

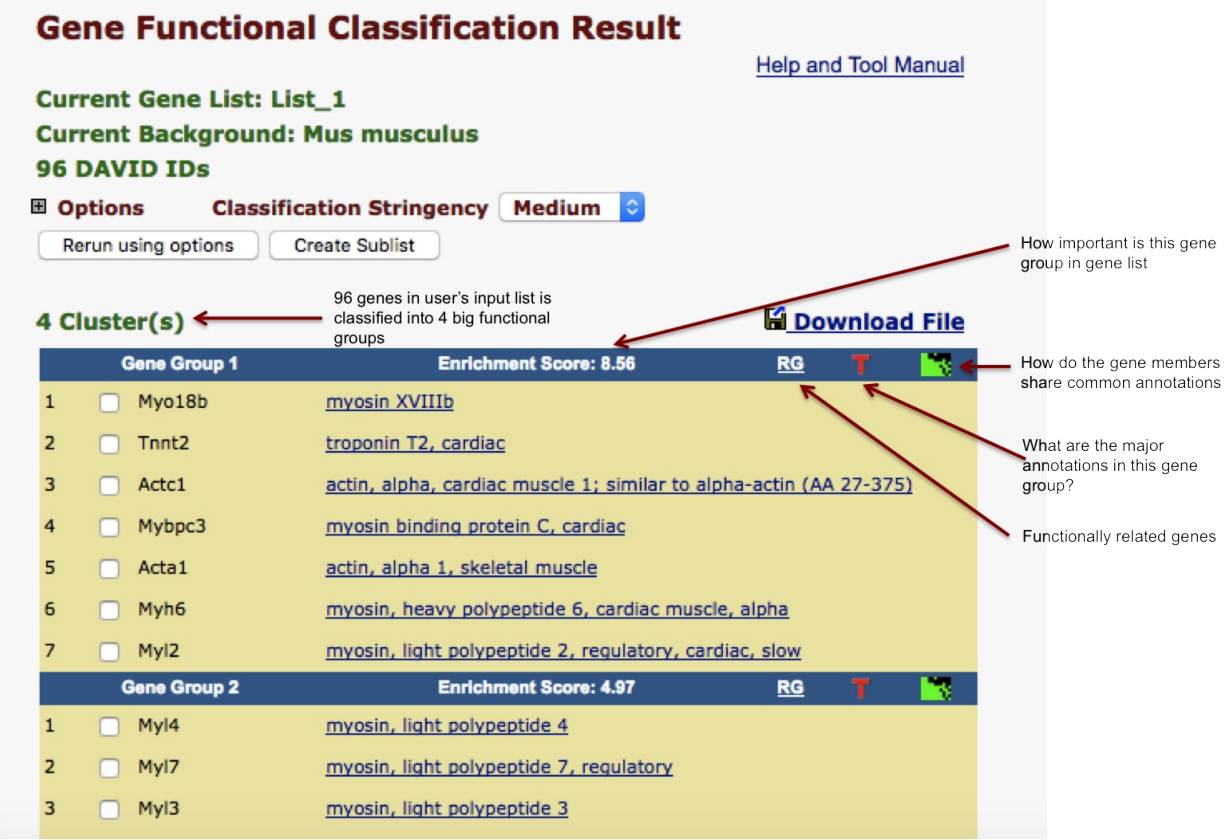
**94 record(s)** Download File

| Adprhl1 | ADP-ribosylhydrolase like 1 | Related Genes | Mus musculus |
|---|---|---|---|
| GOTERM_BP_FAT | protein amino acid de-ADP-ribosylation, | | |
| GOTERM_MF_FAT | magnesium ion binding, ADP-ribosylarginine hydrolase activity, hydrolase activity, hydrolyzing N-glycosyl compounds, ion binding, cation binding, metal ion binding, | | |
| INTERPRO | ADP-ribosylation/Crystallin J1, ADP-ribosylarginine hydrolase, | | |
| PIR_SUPERFAMILY | PIRSF016939:ADP-ribosylarginine hydrolase, PIRSF016939:ADP_ribslarg_hdr, | | |
| SP_PIR_KEYWORDS | hydrolase, | | |
| UP_SEQ_FEATURE | chain:[Protein ADP-ribosylarginine] hydrolase- like protein 1, | | |
| **Bcl11a** | **B-cell CLL/lymphoma 11A (zinc finger protein)** | **Related Genes** | **Mus musculus** |
| GOTERM_BP_FAT | cell activation, immune system development, leukocyte differentiation, transcription, hemopoiesis, lymphocyte differentiation, B cell differentiation, T cell differentiation, T cell activation, B cell activation, leukocyte activation, regulation of transcription, lymphocyte activation, hemopoietic or lymphoid organ development, | | |
| GOTERM_MF_FAT | transcription cofactor activity, transcription corepressor activity, transcription factor binding, zinc ion binding, transcription repressor activity, transcription regulator activity, ion binding, cation binding, metal ion binding, transition metal ion binding, | | |
| INTERPRO | Zinc finger, C2H2-type, Zinc finger, C2H2-type/integrase, DNA-binding, Zinc finger, C2H2-like, | | |

# What is Gene Functional Classification Tool?

DAVID has yet another tool to help the user interpret the biological meaning of their data: the Gene Functional Classification Tool. This tool provides a view similar to annotation clustering, except that the genes are clustered instead of the annotation terms.

The Gene Functional Classification tool reports all the functionally related genes in the user's gene list. Only genes not mapped to any of the functional categories are omitted from the table.




# What is an EASE Score?

In the DAVID annotation system, Fisher's Exact Test is used to measure gene-enrichment in annotation terms. The Ease score is a slightly modified Fisher's Exact p-value. The smaller the value, the more enriched the associated term. Consider,

- `a` = number of genes in the user's list that belong to a particular pathway, `Z`.
- `b` = total number of genes in the user's list.
- `A` = number of genes in the background that belong to pathway `Z`
- `B` = total number of genes in the background.

Fisher's Exact Test would compare `a/b` genes in the user's list with `A/B` genes in the background to see if the genes in the user's list belong to pathway 'Z' by anything more than random chance.

DAVID compares `(a-1)/b` with the `A/B` in Fisher's Exact test to check significance. The p-value obtained is reported as the EASE score.

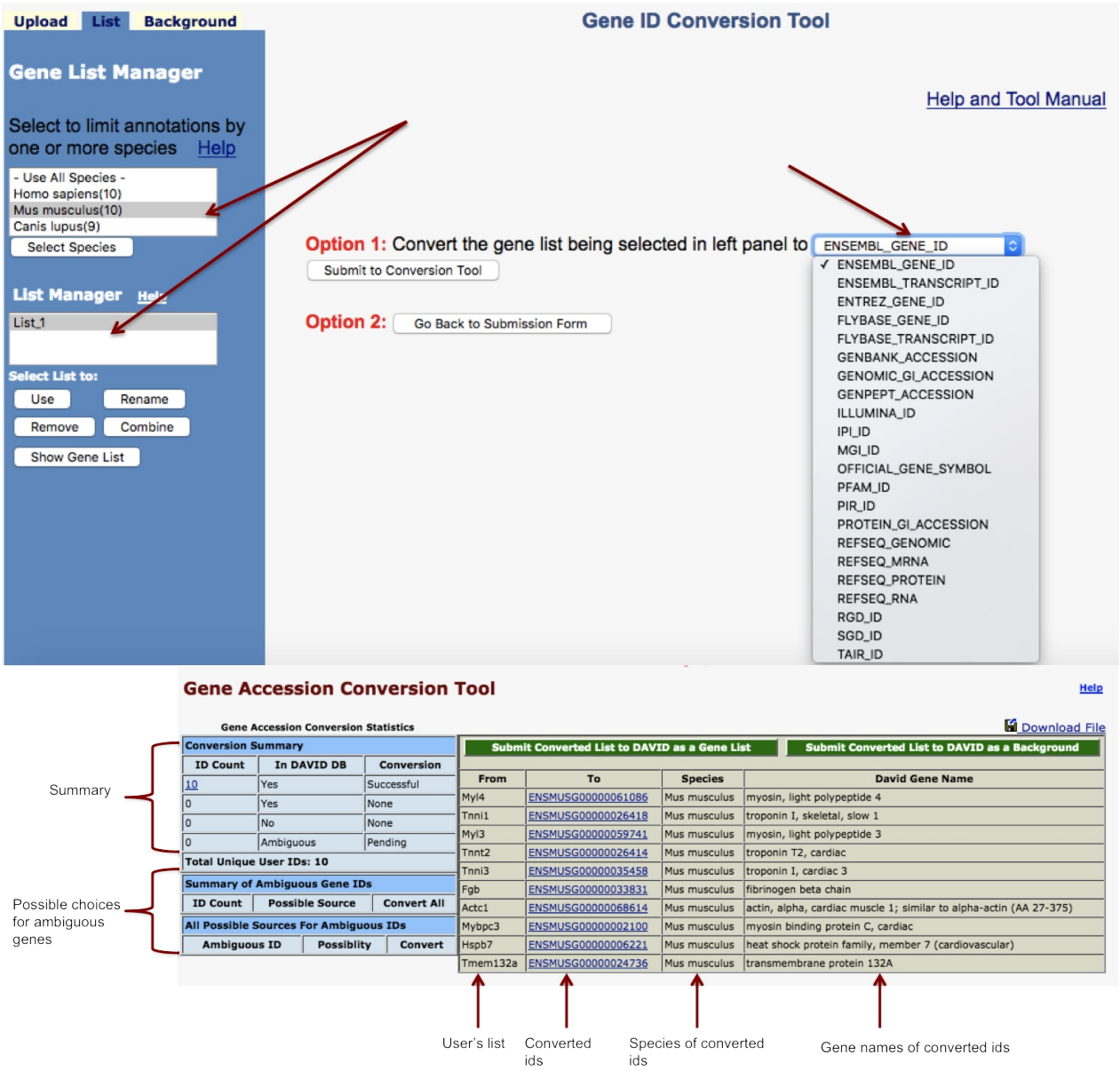
# What is the Kappa statistic?

The clustering algorithm in DAVID is based on the hypothesis that similar annotations should have similar gene members. It uses the Kappa statistic to measure the degree of common genes between two annotations. This is followed by heuristic clustering to group similar annotations according to Kappa values.

Cohen's kappa coefficient (k) measures the degree of agreement between qualitative items. DAVID uses kappa values to measure the functional relationship between gene pairs. A higher Kappa value in DAVID means that genes are more similar in their annotation content. A zero or a negative value for Kappa indicates that genes share no functionality.

# What does the Gene ID Conversion Tool do?

This tool enables the conversion of an accession numbers from one specification to another. There are around 30 different supported id types, which makes this tool quite useful. The gene id conversion page and the results page are shown below.




# What are some pros and cons of DAVID?

## Pros

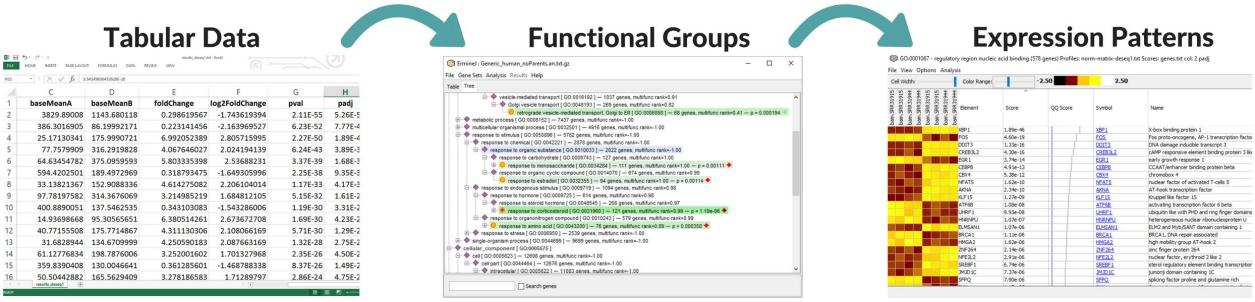
- The database in DAVID provides the user with "many interesting" results. DAVID is very well suited for exploratory analyses as it does not filter the results.
- Since it is an online resource, no installation is required.
- The statistical model is simple and straightforward.
- Every tool can be used independently of another. In other words, you don't need to follow a set series of actions to get the results.

## Cons

- All results are in "flat" tabular form. No visualizations are provided.
- It's hard to gain a higher level overview of the relationship between the terms that are enriched.
- Every click in DAVID opens a new window. You can feel lost after just a few clicks.
- DAVID presents the user with many different views of the same result with the aim to help better interpret the data. However, this can be very confusing; sometimes it feels like running in circles.

# ErmineJ : Gene Set Analysis Software

ErmineJ is a functional analysis tool that can be used to determine the functional categories or biological pathways that are enriched in a list of genes generated by an experiment. The gene list will be tested with a reference set of Gene Ontology (GO) terms to identify interesting biological pathways in the experiment.




In the simplest of terms, you start with a text file that contains genes on each row and measurements in each column. ErmineJ's purpose is to find the common functions across the genes that are above a certain score.

## What type of analyses does ErmineJ offer?

ErmineJ offers different ways to analyze the gene sets. These include,

- **ORA** - Over-Representation Analysis (finds enriched functions).
- **GSR** - Examines the distribution of gene scores in each set.
- **CORR** - Uses correlation of expression profiles.

## How is ErmineJ different from other options?

- ErmineJ runs on Java, hence it is easy to install and run.
- It runs on a desktop ( not browser based ).
- It has a simple, easy-to-use graphical user interface.
- ErmineJ provides data visualization.
- It computes a multi-functionality score.

## What are the input files in ErmineJ?

The main input files for ErmineJ are

1. The annotation file. These specify the gene-to-function information.
2. The gene score file. This specifies which gene was assigned what score.
3. The optional expression data. This assigns numbers across multiple conditions for each gene.

# What are the annotation files?

The annotations represent the names and relationships among GO terms. ErmineJ comes with a GO XML file containing the names and relationships among GO terms. The ErmineJ website offers options to download annotations for many commonly studied organisms ( see details below ).

- Download ErmineJ annotation files

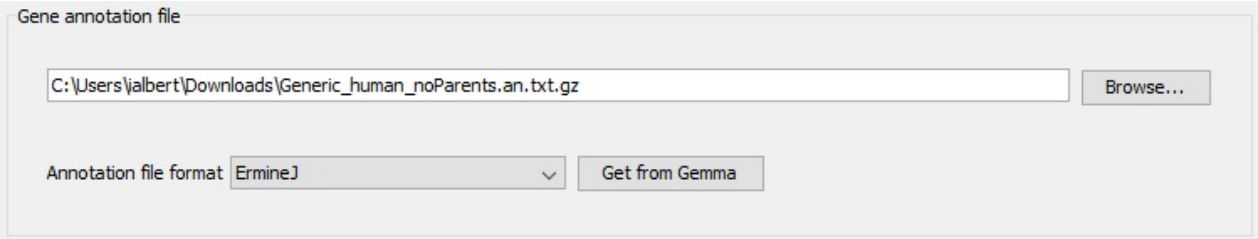
There quite the number of files there. Our examples work on with the generic human and generic mouse files, those can be downloaded from the command line with:

```
curl -O http://chibi.ubc.ca/microannots/Generic_human_noParents.an.txt.gz
```

and

```
curl -O http://chibi.ubc.ca/microannots/Generic_mouse_noParents.an.txt.gz
```

When you start ErmineJ you will need to load one of these files into the system. Only one annotation can be active at any given moment though.




# What is a gene score file?

Gene scores are used to select genes, then based on the selection the tool determines which functions are enriched. It is a tab delimited file with unique gene identifiers as the first column and score values as the second column.

For example if you had an differential expression files (we'll show you how to make these in the RNA-Seq section) that looked like this then selecting and saving the first and last columns of would create a gene score file.

A score may be any value applied to a gene that represents some measure of interest, frequently that is a p-value or fold-change. Non-numeric values such as 'NaN' or '#NUM!' will be interpreted as zero. The first line is considered as a header line.

```
id      padj
WSB1    5.26199960214757e-51
TPBG    7.77064023248874e-48
XBP1    1.88906929176218e-46
ASNS    3.89107811001364e-39
SLC7A5  1.68112640483516e-35
```

You can make this gene score yourself with:

```
curl -O http://data.biostarhandbook.com/ontology/deseq-output.txt
cat deseq-output.txt |  cut -f 1,8 > gene_scores.txt
```

# What is the expression data?

Expression data contain the measurements (e.g. read counts) associated with each gene and condition in a tab-delimited text file format. It is not used to determine functional enrichment. Instead it is displayed when groups of genes are visualized.

The first row is the header line. Missing values must be indicated by blank, not by 'NA' or any other non-numerical values.

An example raw data file may look like this:

```
gene       C1     C2      C3      M1      M2      M3
Myl3    1207.16 1345.04 1247.69 2222.91 3041.39 2819.01
Tnnt2   2401.01 2542.16 2712.17 3648.55 5133.96 4505.65
Mybpc3  1689.82 2026.43 2173.50 3070.45 4439.41 3540.34
...
```

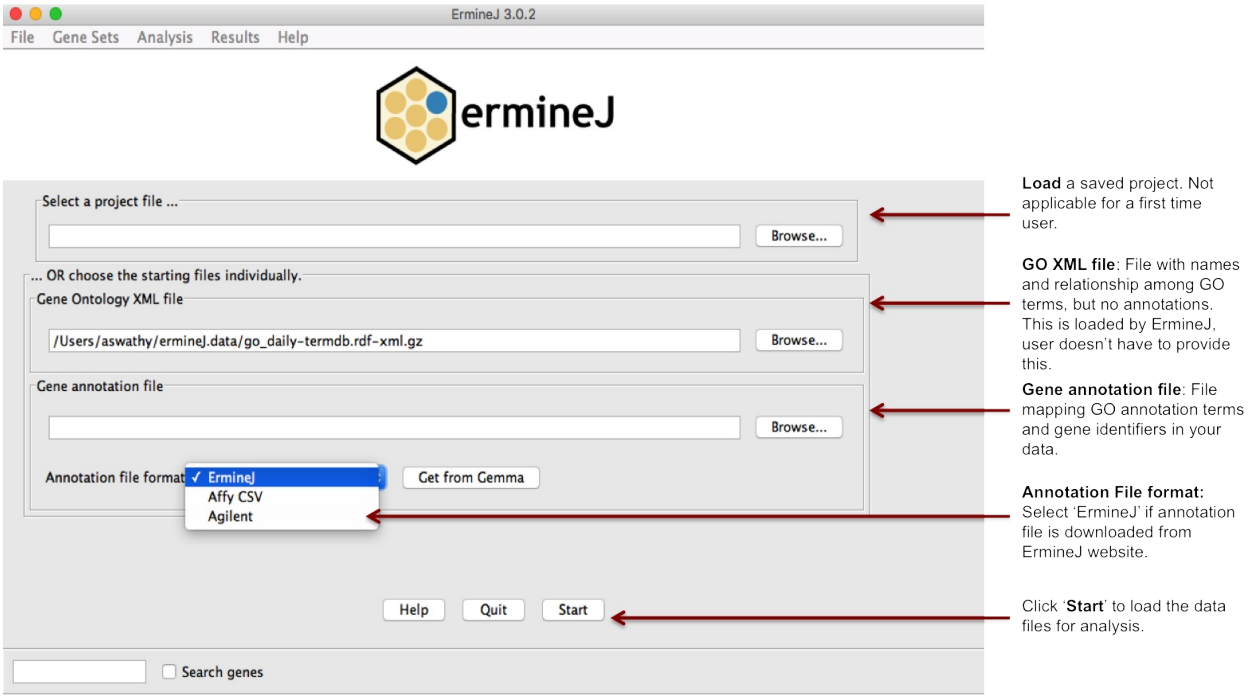
To get our example expression data from the command line do:

```
    curl -O http://data.biostarhandbook.com/ontology/normalized-matrix.txt
```

You will specify this expression data at the same time that we enter the gene score file.

# How do I start an analysis in ErmineJ?

When we start up ErmineJ, the following page will be presented:




**GO XML File :** ErmineJ will automatically download and store this file in the user's ermineJ.data directory. This file is required even if you are not using GO.

**Annotation File** The annotation file appropriate to the experiment can be *downloaded from ErmineJ website using this link.* It is recommended to use 'No parents' versions of annotation file ( parent GO terms are not explicitly *listed in the file* ), as it will result in faster start up time. If it is *downloaded from here,* then select the 'ErmineJ' format.

*Get From Gemma* used to be another option to get the annotation file. At the time of writing this document, the option appears to be non-functional.

Upon clicking 'start,' these files will be loaded and displayed in a table view. The different columns include the name and description of a group (e.g., GO ids and description), the number of elements in the group, and a score denoting the multi-functionality bias of the group. More columns will be added to this as you run an analysis. The details of the table view and tree view are explained below ( see *Results of ORA analysis* ).

Once these files are loaded, choose an analysis to run from the 'Analysis' tab.

# What is an Over-Representation Analysis (ORA)?

Over-representation Analysis (ORA) is a method to identify statistically significant functional groups (e.g. Gene Ontology terms) among high-scoring genes in a gene list.

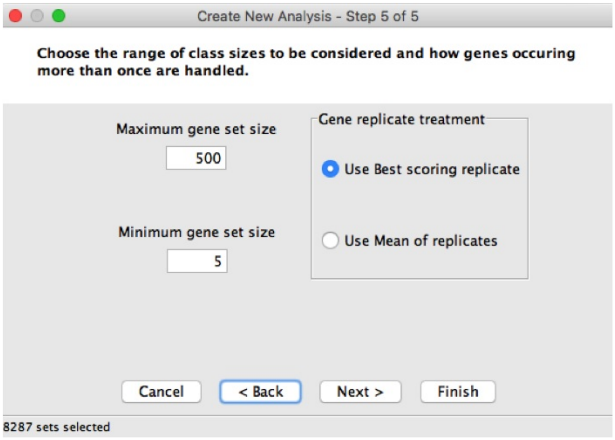
This method requires a gene-score file (see *input files* section) of all genes in the experiment. The user needs to set a threshold of gene scores to 'select' the genes in the gene list to be examined for functional enrichment. The null hypothesis is that genes in the functional groups are randomly distributed between 'selected' and 'non-selected' genes. It uses hyper-geometric distribution to identify the functional groups that are significant among the genes passing the threshold.

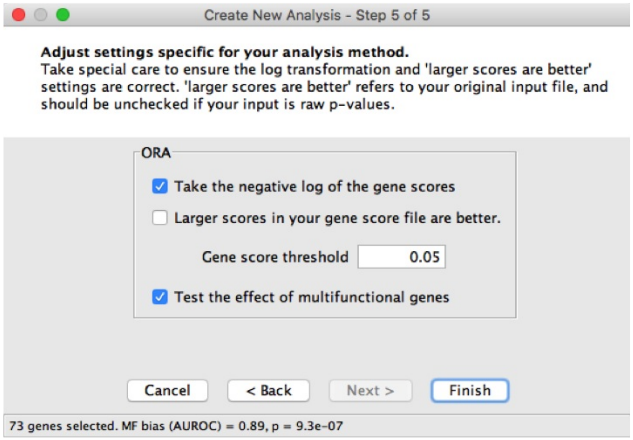
# How do I run an ORA?

To start an ORA analysis, choose ORA from Analysis ->Run Analysis tab.

**ORA requires a complete list of gene scores, not just the selected gene list**. Once a complete list of gene scores is uploaded, the user needs to specify a threshold to determine the genes to be selected for analysis. Raw gene expression data file can also be provided, though this is optional. The following parameters are set by the user:

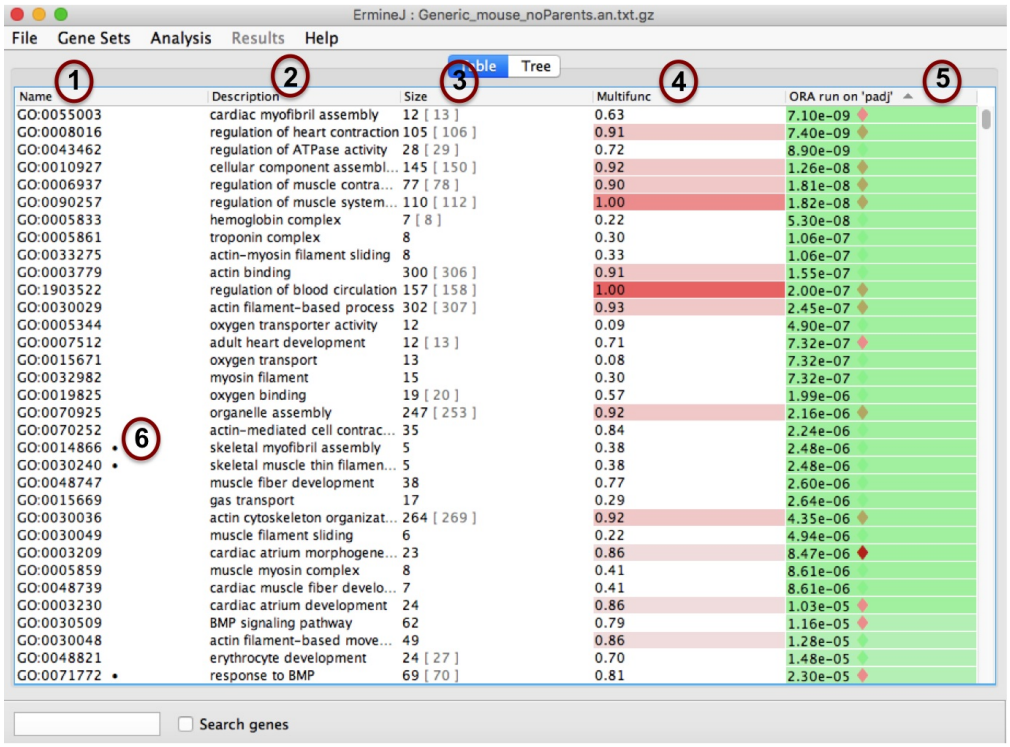
- **Maximum and minimum gene set sizes** - This refers to the number of genes in the Gene Ontology groups to be considered for analysis. It is recommended not to use very small or very large numbers here. Having a large number of functional groups to examine can worsen the multiple-testing issues.
- **ORA-specific settings** - The ORA-specific settings dialog box is shown below. Depending on the gene scores in the input file, the user needs to select the appropriate choice. For example if the scores are p-values, the "take the negative log of the gene scores" box should be checked, and "larger scores in your score file are better" should be unchecked. **Gene score threshold** refers to the score value in the input file and this determines how genes are selected. For example, if the input gene list has p-values, then the threshold might be 0.05 or 0.001, but not 4. The threshold refers to the values *prior* to log-transforming.




# What are the results of an ORA analysis?

The ORA analysis displays the functional groups that are over-represented in the gene list and the associated p-value. The results can be viewed either in a table format (see below) or in a tree format.



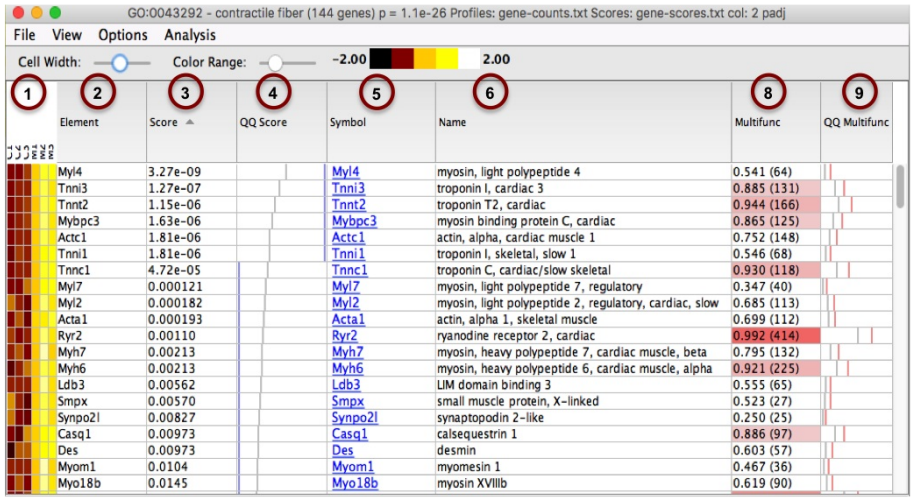

| Name | Description | Size | Multifunc | ORA run on 'padj' |
|---|---|---|---|---|
| GO:0055003 | cardiac myofibril assembly | 12 [ 13 ] | 0.63 | 7.10e-09 |
| GO:0008016 | regulation of heart contraction | 105 [ 106 ] | 0.91 | 7.40e-09 |
| GO:0043462 | regulation of ATPase activity | 28 [ 29 ] | 0.72 | 8.90e-09 |
| GO:0010927 | cellular component assembl... | 145 [ 150 ] | 0.92 | 1.26e-08 |
| GO:0006937 | regulation of muscle contra... | 77 [ 78 ] | 0.90 | 1.81e-08 |
| GO:0090257 | regulation of muscle system... | 110 [ 112 ] | 1.00 | 1.82e-08 |
| GO:0005833 | hemoglobin complex | 7 [ 8 ] | 0.22 | 5.30e-08 |
| GO:0005861 | troponin complex | 8 | 0.30 | 1.06e-07 |
| GO:0033275 | actin-myosin filament sliding | 8 | 0.33 | 1.06e-07 |
| GO:0003779 | actin binding | 300 [ 306 ] | 0.91 | 1.55e-07 |
| GO:1903522 | regulation of blood circulation | 157 [ 158 ] | 1.00 | 2.00e-07 |
| GO:0030029 | actin filament-based process | 302 [ 307 ] | 0.93 | 2.45e-07 |
| GO:0005344 | oxygen transporter activity | 12 | 0.09 | 4.90e-07 |
| GO:0007512 | adult heart development | 12 [ 13 ] | 0.71 | 7.32e-07 |
| GO:0015671 | oxygen transport | 13 | 0.08 | 7.32e-07 |
| GO:0032982 | myosin filament | 15 | 0.30 | 7.32e-07 |
| GO:0019825 | oxygen binding | 19 [ 20 ] | 0.57 | 1.99e-06 |
| GO:0070925 | organelle assembly | 247 [ 253 ] | 0.92 | 2.16e-06 |
| GO:0070252 | actin-mediated cell contrac... | 35 | 0.84 | 2.24e-06 |
| GO:0014866 • | skeletal myofibril assembly | 5 | 0.38 | 2.48e-06 |
| GO:0030240 • | skeletal muscle thin filamen... | 5 | 0.38 | 2.48e-06 |
| GO:0048747 | muscle fiber development | 38 | 0.77 | 2.60e-06 |
| GO:0015669 | gas transport | 17 | 0.29 | 2.64e-06 |
| GO:0030036 | actin cytoskeleton organizat... | 264 [ 269 ] | 0.92 | 4.35e-06 |
| GO:0030049 | muscle filament sliding | 6 | 0.22 | 4.94e-06 |
| GO:0003209 | cardiac atrium morphogene... | 23 | 0.86 | 8.47e-06 |
| GO:0005859 | muscle myosin complex | 8 | 0.41 | 8.61e-06 |
| GO:0048739 | cardiac muscle fiber develo... | 7 | 0.41 | 8.61e-06 |
| GO:0003230 | cardiac atrium development | 24 | 0.86 | 1.03e-05 |
| GO:0030509 | BMP signaling pathway | 62 | 0.79 | 1.16e-05 |
| GO:0030048 | actin filament-based move... | 49 | 0.86 | 1.28e-05 |
| GO:0048821 | erythrocyte development | 24 [ 27 ] | 0.70 | 1.48e-05 |
| GO:0071772 • | response to BMP | 69 [ 70 ] | 0.81 | 2.30e-05 |

1. Name/ID of the functional group.
2. Description of functional group.
3. No. of elements in the functional group.
4. Multi-functionality score ranging from 0 to 1. It indicates the degree to which the group is biased towards multifunctional genes. 1 is the highest.
5. P-value associated with functional group.
6. A '•' it means the functional group has the same exact members as one or more groups.

The first four columns are standard to an ErmineJ table view. As each analysis is run, the results from each run will be added to the standard table as additional columns.

The p-values associated with each functional group, as obtained from the ORA run, are given in the 5th column. Benjamini-Hochberg correction is used to correct the p-values for multiple testing. Different shades of green color denote the strength of enrichment. A green diamond next to the p-value indicates a low effect of multi-functionality correction (low sensitivity) and a red diamond indicates that the group is sensitive to multi-functionality correction (see below for details).

Additional information on each functional group is obtained by double clicking on the corresponding p-value. This displays the details of all genes in the functional group. Expression data is optional, however if it was loaded, the expression values will be displayed as a heatmap on the left-hand side as shown below.




| Element | Score | QQ Score | Symbol | Name | Multifunc | QQ Multifunc |
|---|---|---|---|---|---|---|
| Myl4 | 3.27e-09 | | Myl4 | myosin, light polypeptide 4 | 0.541 (64) | |
| Tnni3 | 1.27e-07 | | Tnni3 | troponin I, cardiac 3 | 0.885 (131) | |
| Tnnt2 | 1.15e-06 | | Tnnt2 | troponin T2, cardiac | 0.944 (166) | |
| Mybpc3 | 1.63e-06 | | Mybpc3 | myosin binding protein C, cardiac | 0.865 (125) | |
| Actc1 | 1.81e-06 | | Actc1 | actin, alpha, cardiac muscle 1 | 0.752 (148) | |
| Tnni1 | 1.81e-06 | | Tnni1 | troponin I, skeletal, slow 1 | 0.546 (68) | |
| Tnnc1 | 4.72e-05 | | Tnnc1 | troponin C, cardiac/slow skeletal | 0.930 (118) | |
| Myl7 | 0.000121 | | Myl7 | myosin, light polypeptide 7, regulatory | 0.347 (40) | |
| Myl2 | 0.000182 | | Myl2 | myosin, light polypeptide 2, regulatory, cardiac, slow | 0.685 (113) | |
| Acta1 | 0.000193 | | Acta1 | actin, alpha 1, skeletal muscle | 0.699 (112) | |
| Ryr2 | 0.00110 | | Ryr2 | ryanodine receptor 2, cardiac | 0.992 (414) | |
| Myh7 | 0.00213 | | Myh7 | myosin, heavy polypeptide 7, cardiac muscle, beta | 0.795 (132) | |
| Myh6 | 0.00213 | | Myh6 | myosin, heavy polypeptide 6, cardiac muscle, alpha | 0.921 (225) | |
| Ldb3 | 0.00562 | | Ldb3 | LIM domain binding 3 | 0.555 (65) | |
| Smpx | 0.00570 | | Smpx | small muscle protein, X-linked | 0.523 (27) | |
| Synpo2l | 0.00827 | | Synpo2l | synaptopodin 2-like | 0.250 (25) | |
| Casq1 | 0.00973 | | Casq1 | calsequestrin 1 | 0.886 (97) | |
| Des | 0.00973 | | Des | desmin | 0.603 (57) | |
| Myom1 | 0.0104 | | Myom1 | myomesin 1 | 0.467 (36) | |
| Myo18b | 0.0145 | | Myo18b | myosin XVIIIb | 0.619 (90) | |

1. Expression profile of the gene
2. Gene name
3. User- defined gene score (from gene-score file).
4. Gene score shown graphically. Blue line denotes the score. Grey line denotes the expected distribution. *
5. Gene symbol
6. Gene Name as in annotation file.
7. Multi-functional score of the gene (1 being highly multi-functional). Values is the parenthesis is the no. of annotations gene has. It roughly correlates to the multi-functional score.
8. Expected and observed multi-functionality score.*

* In both column 3 and 9, if observed value is to the right of the expected, it means the scores in the gene set are generally better than expected by chance.

An example expression data file can be obtained using the commands below. This file contains mouse gene expression raw data for 6 samples (3 controls - C1,C2,C3 and 3 mutants - M1,M2,M3).

```
curl -O http://data.biostarhandbook.com/rnaseq/mouse-gene-expression.txt
cat mouse-gene-expression.txt |  cut -f 1,3,4,5,6,7,8 >gene_counts.txt
```

# What is a multi-functionality score?

Multi-functionality refers to the fact that a gene can have more than one function. In ErmineJ, function refers to the annotations for a gene. Some genes will have more functions (and hence annotations) than others but we must recognize that this information is incomplete. Moreover, we can't even assume that the missing annotations are evenly distributed. Some genes are more "popular" and better-studied than others. Consequently, there is more information available about "popular" genes than "less popular" genes.

ErmineJ calculates a multi-functionality score for each gene. The equation takes into account the size of the functional group. In general, however, the multi-functionality score of a gene correlates highly with the number of annotations it has.

The fourth column in the ErmineJ table format is the multi-functionality score of a functional group. It is a normalized rank that shows the degree to which a group has multi-functional genes. The higher the score, the more generic the functional group is. Details on how the multi-functionality score for a functional group is calculated can be found here.

# Is multi-functionality good or bad?

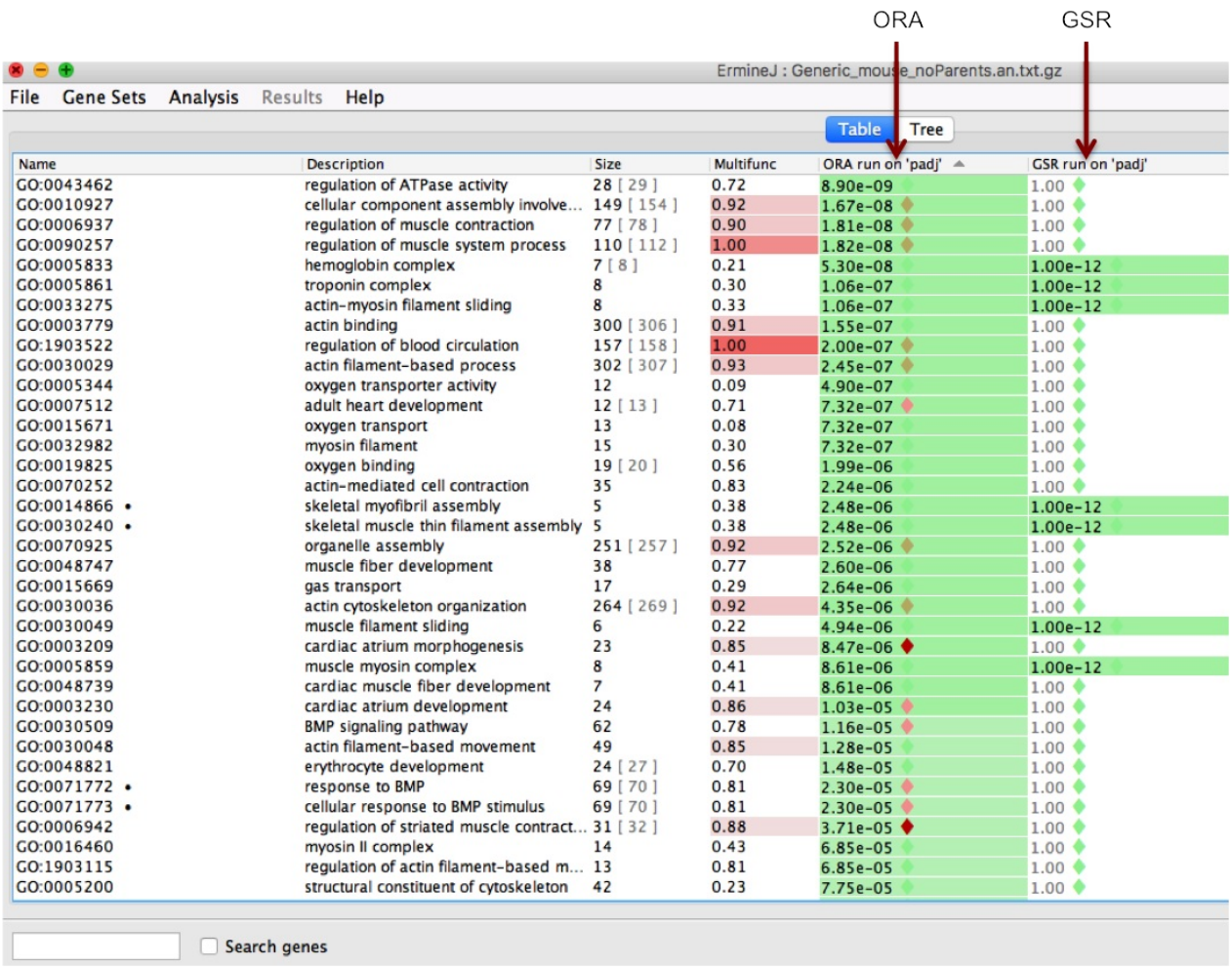
From solely a data interpretation perspective, multifunctional genes are more likely to be false positives. We can think of this as a multiple testing problem. When a gene is multi-functional, it has more opportunities to be shown as enriched just by chance.

Depending on the analysis method used, ErmineJ employs different approaches to correct for multi-functionality. In ORA, genes are iteratively removed in the order of their multi-functionality to test the effect this removal has on the enriched groups. The idea here is that if any functional group is enriched merely due to the multi-functionality of the genes, then such groups will fall out as you remove those genes.

# What is Gene Score Resampling (GSR)?

Gene Score Resampling (GSR) is another method in ErmineJ (similar to ORA) to identify significant functional groups in a gene list. GSR is different from ORA in that it does not require a threshold. Instead, GSR computes an aggregate score (e.g., a geometric mean of gene scores) for a functional group and calculates its significance by resampling the data.

The results of both ORA and GSR methods applied on the same input data sets are shown below. GSR gave fewer results compared to ORA in this analysis.




| Name | Description | Size | Multifunc | ORA run on 'padj' | GSR run on 'padj' |
|---|---|---|---|---|---|
| GO:0043462 | regulation of ATPase activity | 28 [ 29 ] | 0.72 | 8.90e-09 | 1.00 |
| GO:0010927 | cellular component assembly involve... | 149 [ 154 ] | 0.92 | 1.67e-08 | 1.00 |
| GO:0006937 | regulation of muscle contraction | 77 [ 78 ] | 0.90 | 1.81e-08 | 1.00 |
| GO:0090257 | regulation of muscle system process | 110 [ 112 ] | 1.00 | 1.82e-08 | 1.00 |
| GO:0005833 | hemoglobin complex | 7 [ 8 ] | 0.21 | 5.30e-08 | 1.00e-12 |
| GO:0005861 | troponin complex | 8 | 0.30 | 1.06e-07 | 1.00e-12 |
| GO:0033275 | actin-myosin filament sliding | 8 | 0.33 | 1.06e-07 | 1.00e-12 |
| GO:0003779 | actin binding | 300 [ 306 ] | 0.91 | 1.55e-07 | 1.00 |
| GO:1903522 | regulation of blood circulation | 157 [ 158 ] | 1.00 | 2.00e-07 | 1.00 |
| GO:0030029 | actin filament-based process | 302 [ 307 ] | 0.93 | 2.45e-07 | 1.00 |
| GO:0005344 | oxygen transporter activity | 12 | 0.09 | 4.90e-07 | 1.00 |
| GO:0007512 | adult heart development | 12 [ 13 ] | 0.71 | 7.32e-07 | 1.00 |
| GO:0015671 | oxygen transport | 13 | 0.08 | 7.32e-07 | 1.00 |
| GO:0032982 | myosin filament | 15 | 0.30 | 7.32e-07 | 1.00 |
| GO:0019825 | oxygen binding | 19 [ 20 ] | 0.56 | 1.99e-06 | 1.00 |
| GO:0070252 | actin-mediated cell contraction | 35 | 0.83 | 2.24e-06 | 1.00 |
| GO:0014866 • | skeletal myofibril assembly | 5 | 0.38 | 2.48e-06 | 1.00e-12 |
| GO:0030240 • | skeletal muscle thin filament assembly | 5 | 0.38 | 2.48e-06 | 1.00e-12 |
| GO:0070925 | organelle assembly | 251 [ 257 ] | 0.92 | 2.52e-06 | 1.00 |
| GO:0048747 | muscle fiber development | 38 | 0.77 | 2.60e-06 | 1.00 |
| GO:0015669 | gas transport | 17 | 0.29 | 2.64e-06 | 1.00 |
| GO:0030036 | actin cytoskeleton organization | 264 [ 269 ] | 0.92 | 4.35e-06 | 1.00 |
| GO:0030049 | muscle filament sliding | 6 | 0.22 | 4.94e-06 | 1.00e-12 |
| GO:0003209 | cardiac atrium morphogenesis | 23 | 0.85 | 8.47e-06 | 1.00 |
| GO:0005859 | muscle myosin complex | 8 | 0.41 | 8.61e-06 | 1.00e-12 |
| GO:0048739 | cardiac muscle fiber development | 7 | 0.41 | 8.61e-06 | 1.00 |
| GO:0003230 | cardiac atrium development | 24 | 0.86 | 1.03e-05 | 1.00 |
| GO:0030509 | BMP signaling pathway | 62 | 0.78 | 1.16e-05 | 1.00 |
| GO:0030048 | actin filament-based movement | 49 | 0.85 | 1.28e-05 | 1.00 |
| GO:0048821 | erythrocyte development | 24 [ 27 ] | 0.70 | 1.48e-05 | 1.00 |
| GO:0071772 • | response to BMP | 69 [ 70 ] | 0.81 | 2.30e-05 | 1.00 |
| GO:0071773 • | cellular response to BMP stimulus | 69 [ 70 ] | 0.81 | 2.30e-05 | 1.00 |
| GO:0006942 | regulation of striated muscle contract... | 31 [ 32 ] | 0.88 | 3.71e-05 | 1.00 |
| GO:0016460 | myosin II complex | 14 | 0.43 | 6.85e-05 | 1.00 |
| GO:1903115 | regulation of actin filament-based m... | 13 | 0.81 | 6.85e-05 | 1.00 |
| GO:0005200 | structural constituent of cytoskeleton | 42 | 0.23 | 7.75e-05 | 1.00 |

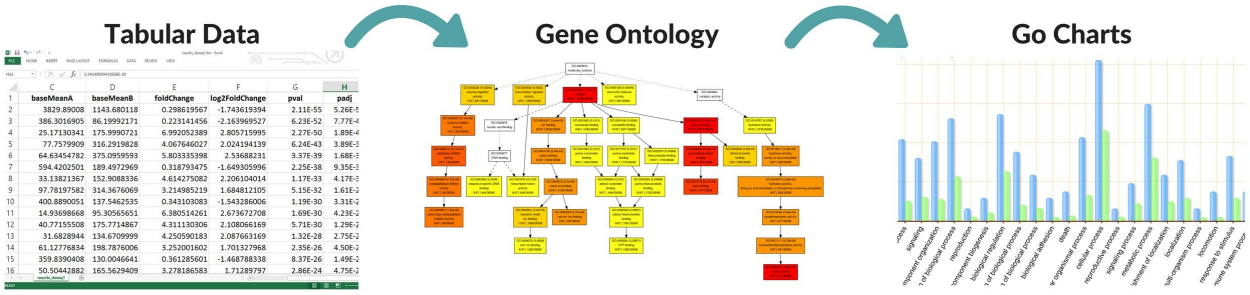
# What is Correlation Analysis in ErmineJ?

The correlation analysis (CORR) is differs from other ErmineJ methods by assessing how well the genes in a functional group are clustered together. This method does not use gene scores. Instead, it uses the expression profiles of the genes themselves. CORR checks if the correlation among the members of a functional group is more than expected by chance. The raw score for a functional group is the absolute mean of the pairwise correlation of its members.

Unlike other methods in ErmineJ, the CORR analysis is computationally intensive. Hence, one needs to be careful while setting the parameters when running this analysis.

# AGRIGO - Go Analysis Toolkit and Database for Agricultural Community

AgriGo is a web based tool for gene ontology analysis. It is mainly focused on the agricultural species. However a a good number of the widely studied genome like mouse, chicken etc are also included.




Strenghts of AgriGO are:

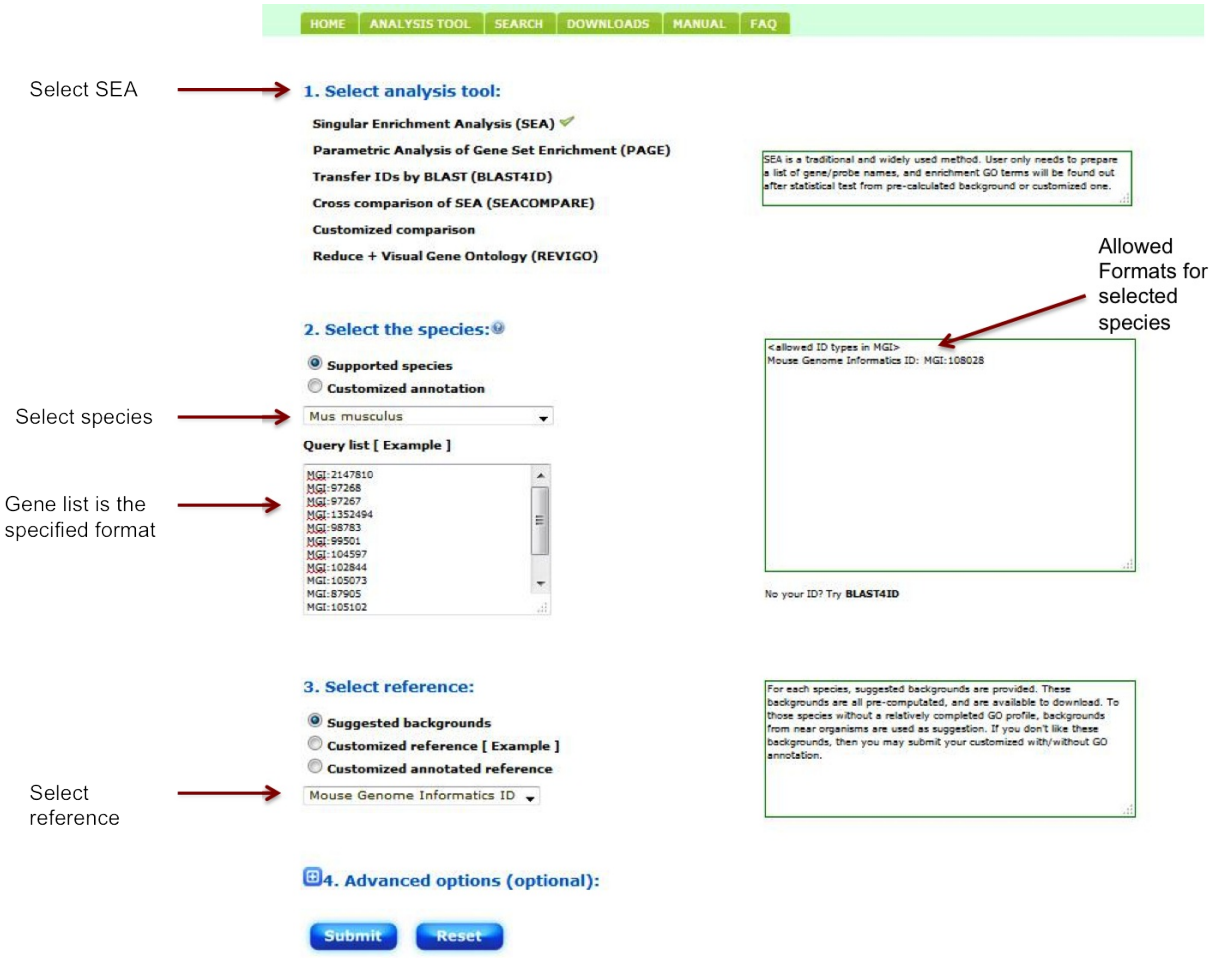
- Usable and simple interface.
- Great graphical visualizations.

## What is Singular Enrichment Analysis (SEA) in AgriGo?

Singular Enrichment Analysis (SEA)in AgriGo can be used to find the enriched Gene Ontology terms in user's list. User only needs to prepare a list of genes. After performing statistical tests with pre-computed or customized backgrounds, user will be provided with enriched GO terms. These results can then be visualized in a GO hierarchical tree structure along with its significance level.

## How to use SEA?

When SEA as selected as the analysis tool, the user is required to specify the species and the gene list in the required format. When species is selected, allowed formats for gene list will be displayed on the right hand side box. For example, if `Mus musculus` is selected, then gene list should be submitted.




The data file used here can be obtained using the commands below. Here a mouse genelist is selected for AgriGO analysis by using an expression p-value cut off of 0.05.

```
curl -O http://data.biostarhandbook.com/rnaseq/mouse-gene-expression.txt
cat mouse-gene-expression.txt |  awk '$2 <0.05 {print $1}' > genelist.txt
```

Since AgriGo requires mouse gene list to be in 'MGI' format, submit this to the `Batch Query` in Mouse Genome Informatics website.

The MGI ids obtained from here can be pasted in the `query list` box. Once the reference is chosen, submit the list.

# What are the outputs of AgriGO?

A brief summary report and a table of enriched GO terms along with their statistical significance will be displayed once the gene list is processed. In addition, user has many options to generate graphical visualizations.

**Analysis Brief Summary**

**Job ID:** 656410826 [Useful within 7 days]
**Job Name:** 656410826
**Species:** *Mus musculus*
**GO type:** Completed GO
**Background/Reference:** Mouse genome informatics ID
**Annotated number in query list:** 67 [ Download ]
**Annotated number in background/reference:** 35008
**Significant GO terms:** 204 [ Details ]

**Graphical Results**

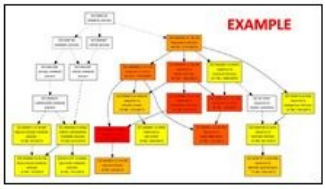
**Select Category**

Biological Process  Cellular Component  Molecular Function

**Advanced Parameter Settings**

Graphic result format: PNG PDF JPEG GIF SVG
Graph rank direction: Top to Bottom Left to Right Bottom to Top Right to Left
Graph font size (pt): 7 8 9 10 11 12

Generate Image




**GO flash Chart**

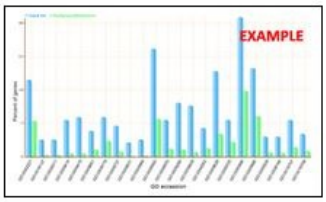
**Select Category**

Biological Process  Cellular Component  Molecular Function

**Advanced Parameter Settings**

Bar style: Glass Bar Filled Bar 3D Bar Cylinder Bar
Query bar color: #1A8CFF Bg/Ref bar color: #66FF33 [HEX format only] [ default ]
X legend content: GO annotation GO accession font: 14
X legend rotation: 300 [270 to 315 is suggested]

Generate Chart

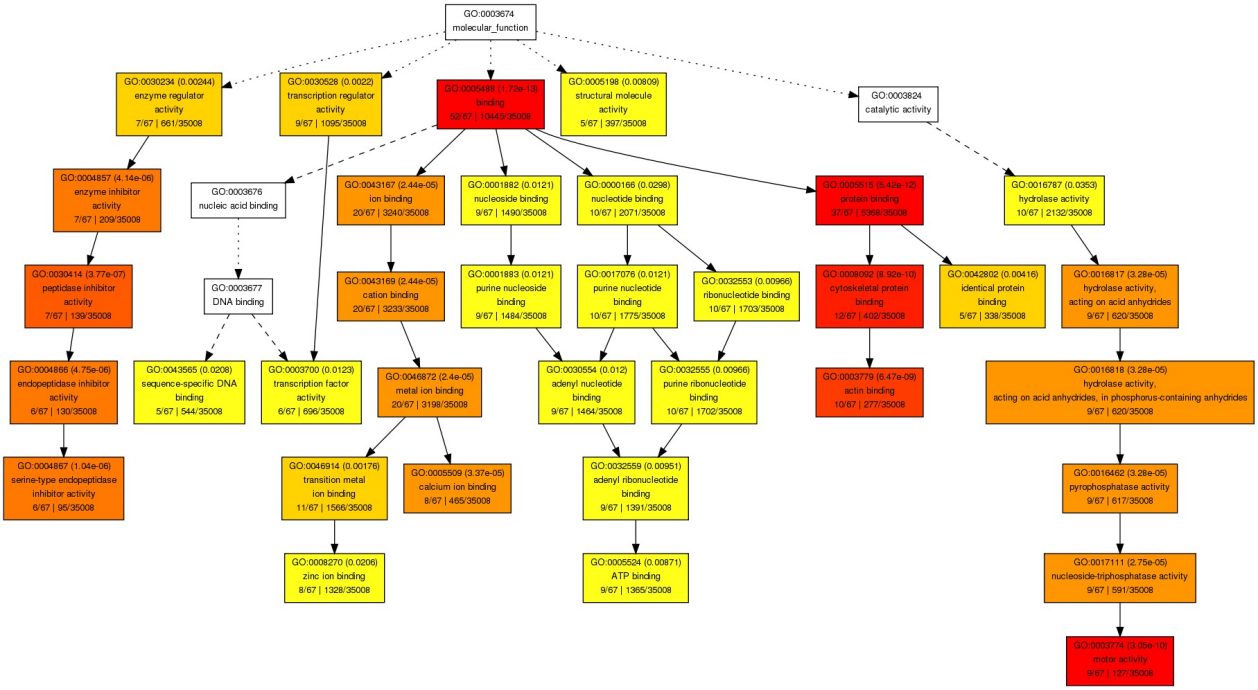



**Detail information**

You can [ Browse in tree traversing mode ] [ Browse all GO terms ] [ Download ] [ REVIGO ]
Or select from following significant terms to [ Draw graphical results ] [ Create bar chart ]

| GO term | Ontology | Description | Number in input list | Number in BG/Ref | p-value | FDR |
|---|---|---|---|---|---|---|
| GO:0006936 | P | muscle contraction | 13 | 119 | 3.5e-19 | 3.4e-16 |
| GO:0003012 | P | muscle system process | 13 | 130 | 1e-18 | 5e-16 |
| GO:0003015 | P | heart process | 11 | 86 | 4.6e-17 | 1.1e-14 |
| GO:0060047 | P | heart contraction | 11 | 86 | 4.6e-17 | 1.1e-14 |

Shown below is the graphical view of enriched GO terms in *Molecular Function* category. Different colors denote the level of significance. Red means highly significant and orange means medium significance. The significance values are shown inside the boxes.




# How do I prepare a custom annotation for AgriGO?

To use custom annotation with Agrigo you need to submit a tab delimited data where one column is an ID and the other is a GO term. For example to use human annotations you would need to first get the association file you get

```
curl -L http://geneontology.org/gene-associations/gene_association.goa_human.gz | gunzip -c >
assoc.txt
```

Note that you can obtain a protein or gene-name GO pair via:

```
cat assoc.txt | grep -v '!' | cut -f 3,5 | head
```

and that looks like:

```
PDCD6   GO:0005509
CDK1    GO:0004674
CDK1    GO:0005524
CDK1    GO:0005634
```

Now sort this so that it later can be joined with another file.

```
cat assoc.txt | grep -v '!' | cut -f 3,5 | sort -k 1b,1 > pairs.txt
```

If you had another set of differentially expressed gene names (from the same naming scheme) then you can sort those too then join them on this file. Take for example this enrichment file deseq-output.txt

```
curl -O http://data.biostarhandbook.com/ontology/deseq-output.txt
```

Take the first 100 differentially expressed genes and sort them:

```
cat deseq-output.txt | head -100 | sort -k 1,1 > top.txt
```

You can join the two files and cut only the first two columns, this subselects from the pairs file only those lines that appear in the top.txt file. This produces an AgriGO input file:

```
join pairs.txt top.txt -t $'\t' | cut -f 1, 2 | head
```

and yes the same term can be there multiple times since each is an association to a function.

```
AARS    GO:0000049
AARS    GO:0001942
AARS    GO:0002161
AARS    GO:0003676
```

You can load the resulting file into AgriGO

```
join pairs.txt top.txt -t $'\t' | cut -f 1,2 > agrigo.txt
```

# Biological data sources

## What is "data"?

While many people may feel that they have a good practical understanding on what data formats are, it can be unexpectedly difficult to articulate even what the word *data* actually means.

> Reflect for a moment, what is data?

You may realize that "data" and "data format" are closely related concepts that imply the following:

- A symbolic representation of some information.
- An organization (a design, an optimization) of that information.

The definitions above also imply the following:

- The same information may be represented in different formats (optimized differently).
- The same format (optimization) may be used to represent different types of information.
- There may be information in data that is not readily accessible (optimized).

Often, it can be surprisingly difficult to predict the types of new information that we can derive from a dataset. Pushing the limits of what we can extract from data is an essential part of science.

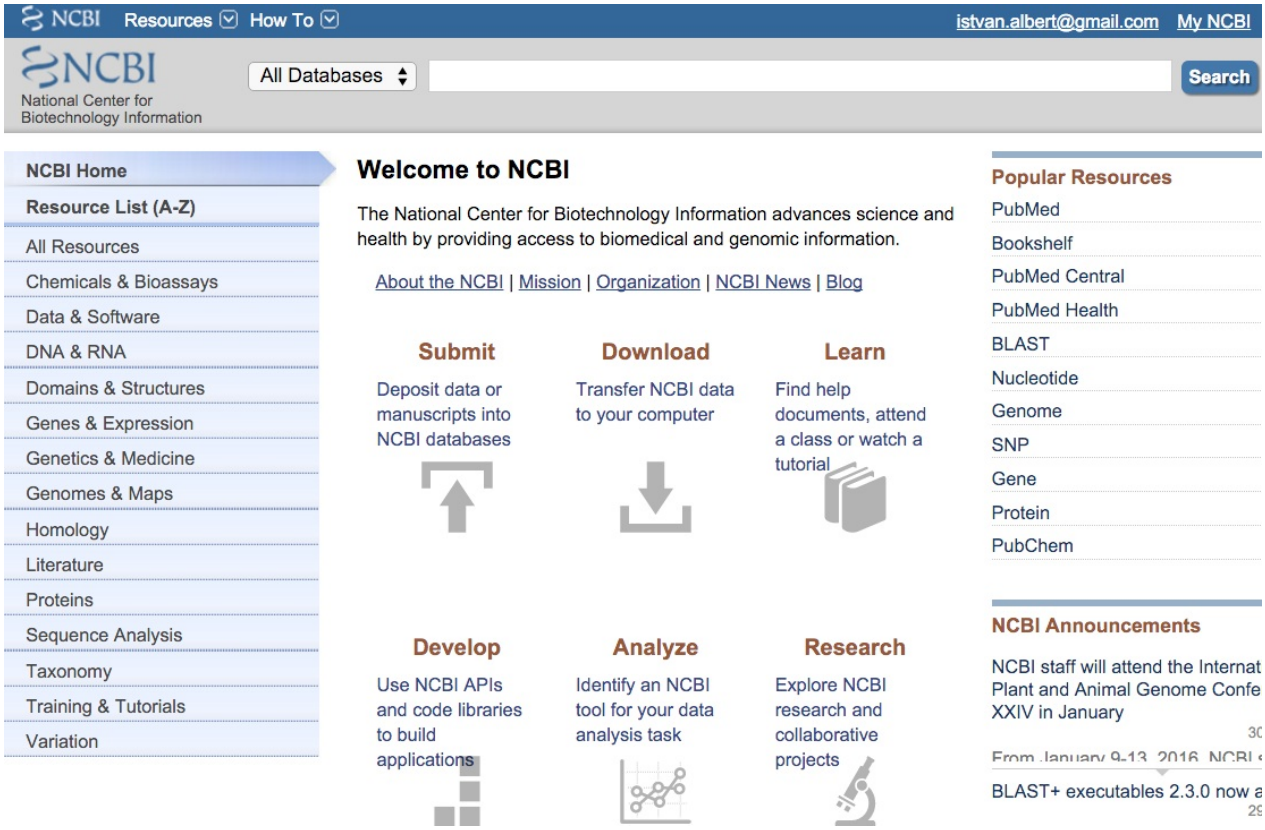
> Understanding data formats, what information is encoded in each, and when it is appropriate to use one format over another is an essential skill of a bioinformatician.

## Where is biomedical data stored?

> Walter Goad of the Theoretical Biology and Biophysics Group at Los Alamos National Laboratory and others established the Los Alamos Sequence Database in 1979, which culminated in 1982 with the creation of the public GenBank. (source: Wikipedia)

GenBank can be considered as the first large-scale, public repository of biomedical data. Today, it is integrated in the the services provided by the *National Center for Biotechnology Information (NCBI)*.

Today, just about all biomedical data is distributed via web-based services. You may already be familiar with one or more such services; for example, NCBI Home Page is a major entry point to a wealth of information.




When visiting these sites, you are likely to be subjected to an information overload coupled with serious usability problems. Typically, it is surprisingly difficult to find relevant, up-to-date, and correct information on any entity of interest. With the growth of data, we anticipate the situation to only worsen.

It is not your fault: it is theirs. Persevere and keep at it. You will eventually find what you need.

# What are the major DNA data repositories?

DNA sequences collected by scientists are deposited in databases and made available to the public via the Internet. The INSDC: International Nucleotide Sequence Database Collaboration has assumed stewardship for maintaining copies of the so called "primary DNA data". The member organizations of this collaboration are:

- NCBI: National Center for Biotechnology Information
- EMBL: European Molecular Biology Laboratory
- DDBJ: DNA Data Bank of Japan

The INSDC has set up rules on the types of data that will be mirrored. The most important of these from a bioinformatician's perspective are:

- GenBank contains all annotated and identified DNA sequence information
- SRA: Short Read Archive contains measurements from high throughput sequencing experiments

UniProt: Universal Protein Resource is the most authoritative repository of protein sequence data.

Protein Data Bank (PDB) is the major repository of 3D structural information about biological macromolecules (proteins and nucleic acids). PDB contains structures for a spectrum of biomolecules - from small bits of proteins/nucleic acids all the way to complex molecular structures like ribosomes.

# What kind of other data sources are there?

Beyond the primary DNA data, each institute (NCBI, EMBL, DDJB) hosts a variety of services and vast numbers of annotated datasets that may be specific to each organization.

In addition, there are different tiers of other organizations that have taken it upon themselves to reorganize, rename, and process existing data and provide additional functionalities and visualizations. Scientific interest groups formed around model organisms may also maintain their own resources:

- UCSC Genome Browser invented the graphical browser visualization of genomes. Today it offers comprehensive comparative genomics data across vertebrate genomes.
- FlyBase is the database of Drosophila (fruit fly) genes and genomes.
- WormBase is the primary resource for nematode biology.
- SGD: Saccharomyces Genome Database provides comprehensive integrated biological information for the budding yeast *Saccharomyces cerevisiae* along with search and analysis tools to explore these data.
- RNA-Central is a meta-database that integrates information from several other resources.
- TAIR The Arabidopsis Information Resource is the primary resource for genetic and molecular data about *Arabidopsis thaliana*, a model higher plant.
- EcoCyc (Encyclopedia of *E. coli* Genes and Metabolic Pathways) is a scientific database for the bacterium *Escherichia coli* K-12 MG1655.

Beyond the second tier of comprehensive databases, there are a large number of specialized data resources, often started and/or managed by single individuals with little-to-no support.

Describing each resource in detail is beyond the scope of this book. Most provide "self discoverable" interface elements, though their usability is typically far from optimal.

It is also hard to shake the feeling that each data provider's ultimate goal is to be "too big to fail"; hence, there is an eagerness to integrate more data for the sole purpose of growing larger. It does not help that interface design is often supervised by scientists and this typically means jamming way too much information into way to little space. As a rule, using bioinformatics data resources is frustrating, requires patience, and often requires the suspension of disbelief.

> Among the greatest challenges for a newcomer, however, is where to look for authoritative and up-to-date information. In addition to using different naming schemes, data types and content may vary from resource to resource, adding no small confusion when trying to combine and reconcile information gleaned from different sources.

# What's in a name?

Two submitters using the same systematic names to describe two different genes would obviously lead to very confusing situtations.

On the other hand, making gene names memnonics that reflect their function, such as calling a *pigment dispersing factor* a *PDF*, or *Protein Kinase A* as *PKA* helps immensely during the biological interpretation process. The problem, of course, is that there could be many different variants of *PKA* in different genomes; moreover, potentially different terms could be shortened to similar or even identical mnemonics.

In general, all biological terms are defined by multiple names: there is a common name, a systematic name, a database-specific name, and even a so-called accession number. These latter are specific to the data source and may not actually be numbers. The common names typically carry a biological meaning, whereas systematic names are built with a logic to make them unique.

Most community projects formed around model organisms have adopted custom gene naming conventions that may only apply to data produced by that community.

# SGD Project systematic names

For example, the Saccharomcyes Genome database project has detailed documentation on naming conventions for every element. For example:

- Gene naming: The gene name should consist of three letters (the gene symbol) followed by an integer (e.g. ADE12)
- Open reading frame naming: First letter 'Y' ('Yeast'); the second letter denotes the chromosome number ('A' is chr I, etc.); the third letter is either 'L' or 'R' for left or right chromosome arm; next is a three digit number indicating the order of the ORFs on that arm of a chromosome starting from the centromere (like a highway exit), irrespective of strand; finally, there is an additional letter indicating the strand, either 'W' for Watson (forward) or 'C' for Crick (reverse). Thus, **YGR116W** is the 116th ORF right of the centromere on chromosome VII on the forward strand.

# Human gene naming

The HUGO Gene Nomenclature Committee is the only worldwide authority that assigns standardised nomenclature to human genes. The names assigned by HGNC are typically formed from a so-called stem (or root) symbol that is used as the basis for a series of approved symbols that are defined as members of either a functional or structural gene family. For example *CYP*: cytochrome P450; *HOX*: homeo box; *DUSP*: dual specificity phosphatase; *SCN2A*: sodium channel voltage-gated type II alpha 2 polypeptide. etc.

Since the the mouse and rat genomes are the closest human models and are widely studied , making human and mouse gene names follow similar rules has always been a priority of the Mouse Gene Nomenclature Committee (MGNC)

The Vertebrate Gene Nomenclature Committee (VGNC) is an extension of the established HGNC (HUGO Gene Nomenclature Committee) project that names human genes. VGNC is responsible for assigning standardized names to genes in vertebrate species that currently lack a nomenclature committee. The current prototype VGNC naming species is the chimpanzee.

# Is it possible to automate data access?

Well-designed databases allow automated data access usually either via a third party program, a programming interface (API), or as database queries.

# How do we automate data access?

Some datasets are stored as flat files on websites. Often, these can be easily processed via UNIX tools. For example, take the sequence data for chromosome 22 of the human genome, which is distributed from the UCSC data site at http://hgdownload.cse.ucsc.edu/goldenPath/hg38/chromosomes/

```
# Download and unzip the file on the fly.
curl http://hgdownload.cse.ucsc.edu/goldenPath/hg38/chromosomes/chr22.fa.gz | gunzip -c > chr2
2.fa

# Look at the file
cat chr22.fa | head -4
```

it produces:

```
>chr22
NNNN
```

# GenBank format

## What formats are sequencing data stored in?

The most common sequence data formats are the GenBank, FASTA, and FASTQ formats.

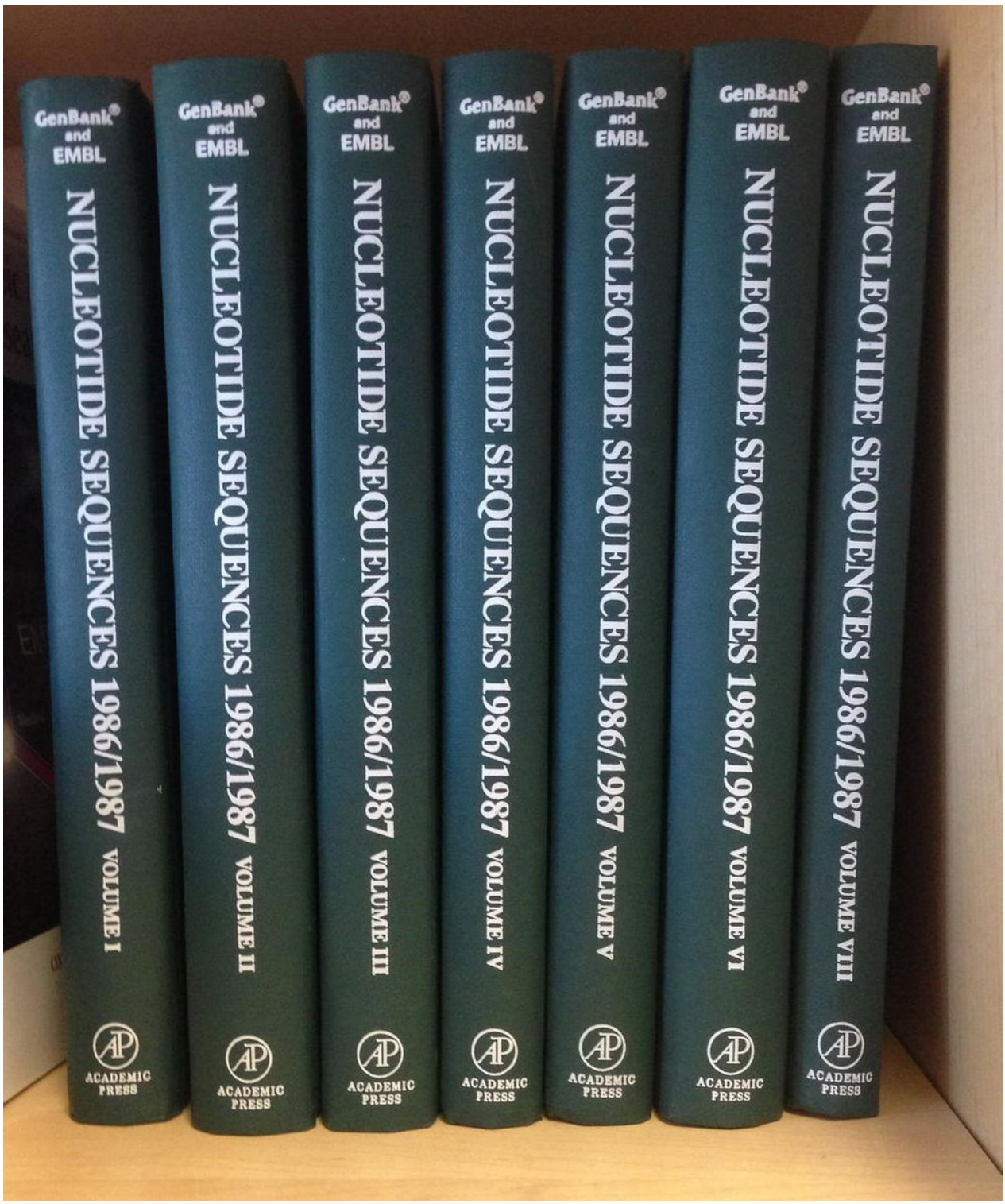
The first two more often represent curated sequence information. The FASTQ format, on the other hand, represents experimentally obtained data typically measured via sequencing instrumentation.

## What is the GenBank format?

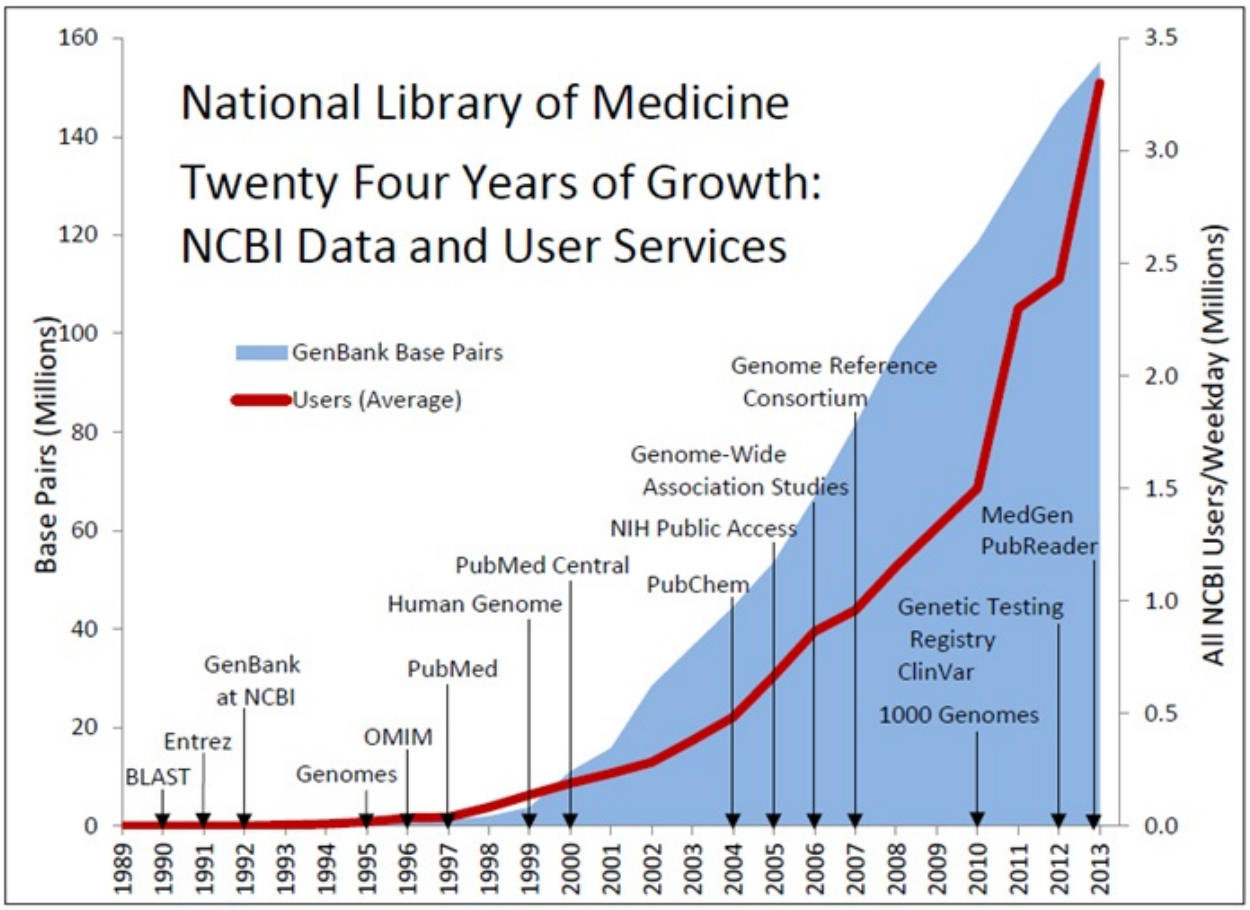
GenBank format is one of the oldest bioinformatics data formats and was originally invented to bridge a gap between a human-readable representation and one that can be efficiently processed by a computer. The format (defined here) has a so-called fixed-width format, where the first 10 characters form a column that serves as an identifier and the rest of the lines are information corresponding to that identifier.

```
LOCUS       AF086833               18959 bp    cRNA    linear   VRL 13-FEB-2012
DEFINITION  Ebola virus - Mayinga, Zaire, 1976, complete genome.
ACCESSION   AF086833
VERSION     AF086833.2  GI:10141003
KEYWORDS    .
SOURCE      Ebola virus - Mayinga, Zaire, 1976 (EBOV-May)
  ORGANISM  Ebola virus - Mayinga, Zaire, 1976
            Viruses; ssRNA viruses; ssRNA negative-strand viruses;
            Mononegavirales; Filoviridae; Ebolavirus.
REFERENCE   1  (bases 1 to 18959)
  AUTHORS   Bukreyev,A.A., Volchkov,V.E., Blinov,V.M. and Netesov,S.V.
  TITLE     The VP35 and VP40 proteins of filoviruses. Homology between Marburg
            and Ebola viruses
  JOURNAL   FEBS Lett. 322 (1), 41-46 (1993)
   PUBMED   8482365
REFERENCE   2  (bases 1 to 18959)
  AUTHORS   Volchkov,V.E., Becker,S., Volchkova,V.A., Ternovoj,V.A.,
            Kotov,A.N., Netesov,S.V. and Klenk,H.D.
  TITLE     GP mRNA of Ebola virus is edited by the Ebola virus polymerase and
            by T7 and vaccinia virus polymerases
```

There were times (up to 1987) when the GenBank database was printed and published as a book.




Today, if that stack of books were to be printed, it may be many miles high. According to the Wikipedia entry on GenBank, as of 15 June 2016, GenBank release 214.0 has 194,463,572 loci, 213,200,907,819 bases, from 194,463,572 reported sequences.




# When do we use the GenBank format?

The GenBank format has the advantage of being a generic format that can represent a wide variety of information while still keeping this information human-readable. And that was its intended purpose.

For this same reason, however, it is not optimized for any particular data analysis task. Hence, it is used mainly to exchange information and is almost never suited as input to an analysis protocol.

We typically convert a GenBank file to some other, simpler format to work with it. ReadSeq is a useful conversion tool to use for this purpose.

# What is RefSeq?

The NCBI Reference Sequence (RefSeq) project provides sequence records and related information for numerous organisms, and provides a baseline for medical, functional, and comparative studies. The RefSeq database is a non-redundant set of reference standards derived from all the data deposited in GenBank that includes chromosomes, complete genomic molecules (organelle genomes, viruses, plasmids), intermediate assembled genomic contigs, curated genomic regions, mRNAs, RNAs, and proteins.

# How are RefSeq sequences named?

The most distinguishing feature of a RefSeq record is the distinct accession number format that begins with two characters followed by an underscore (e.g., NP_).

| Accession prefix | Molecule type | Comment |
|---|---|---|
| AC_ | Genomic | Complete genomic molecule, usually alternate assembly |
| NC_ | Genomic | Complete genomic molecule, usually reference assembly |
| NG_ | Genomic | Incomplete genomic region |
| NT_ | Genomic | Contig or scaffold, clone-based or WGS[a] |
| NW_ | Genomic | Contig or scaffold, primarily WGS[a] |
| NS_ | Genomic | Environmental sequence |
| NZ_[b] | Genomic | Unfinished WGS |
| NM_ | mRNA | |
| NR_ | RNA | |
| XM_[c] | mRNA | Predicted model |
| XR_[c] | RNA | Predicted model |
| AP_ | Protein | Annotated on AC_ alternate assembly |
| NP_ | Protein | Associated with an NM_ or NC_ accession |
| YP_[c] | Protein | |
| XP_[c] | Protein | Predicted model, associated with an XM_ accession |
| ZP_[c] | Protein | Predicted model, annotated on NZ_ genomic records |

# FASTA formats

## What is the FASTA format?

The FASTA format is the "workhorse" of bioinformatics. It is used to represent sequence information. The format is deceivingly simple:

- A `>` symbol on the FASTA header line indicates a *fasta record* start.
- A string of letters called the *sequence id* may follow the `>` symbol.
- The header line may contain an arbitrary amount of text (including spaces) on the same line.
- Subsequent lines contain the sequence.

The sequence is assumed to come from an *alphabet* that corresponds to a biological entity. For example, the standard alphabet for nucleotides would contain: `ATGC`. An extended alphabet may also contain the `N` symbol to indicate a base that could be any of `ATGCN`. A different extension of a nucleotide alphabet may allow extended symbols such as `W` that corresponds to nucleotide that is either an `A` or `T` etc. Gaps may be represented via `.` or `-` symbols. Search the web for "IUPAC nucleotides" to get a list of all such symbols that may be included in an alphabet. The same rationale needs to be followed by protein alphabets. The following is a FASTA file with two FASTA records:

```
>foo
ATGCC
>bar other optional text could go here
CCGTA
```

As it happens, the FASTA format is not "officially" defined - even though it carries the majority of data information on living systems. Its origins go back to a software tool called Fasta written by David Lipman (a scientist that later became, and still is, the director of NCBI) and William R. Pearson of the University of Virginia. The tool itself has (to some extent) been superceded by the BLAST suite but the format (named after the tool itself) not only survived, but has become the de facto standard.

## Are there problems with this format?

The lack of a definition of the FASTA format and its apparent simplicity can be a source of some of the most confounding errors in bioinformatics. Since the format appears so exceedingly straightforward, software developers have been tacitly assuming that the properties they are accustomed to are required by some standard - whereas no such thing exists. In general, when creating FASTA files, the following rules should be observed:

- Sequence lines should not be too long. While, technically, a Fasta file that contains the sequence of the entire human chromosome 1 on a single line (330 million bases) is a valid FASTA file, most tools that run on such a file would fail in various, occasionally spectacular ways (like bringing your entire computer to a screeching halt). Making a FASTA file like that would be useful solely as a practical joke on April 1st.

- Some tools may tacitly(!) accept data containing alphabets beyond those that they know how to deal with. When your data contains alphabets beyond the 4 nucleotides or 20 aminoacid letters, you need to ensure that whatever tool you use is able to recognize the extended values.
- The sequence lines should always wrap at the same width (with the exception of the last line). Some tools will fail to operate correctly and may not even warn the users if this condition is not satisfied. The following is technically a valid FASTA but it may cause various subtle problems.

  ```
  >foo
  ATGCATGCATGCATGCATGC
  ATGCATGCATG
  ATGCATGCATGCATGCA
  ```

  it should be reformatted to:

  ```
  >foo
  ATGCATGCATGCATGCATGC
  ATGCATGCATGATGCATGCA
  TGCATGCA
  ```

  the latter being the desired format.

- Use upper-case letters. Whereas both lower-case and upper-case letters are allowed by the specification, the different capitalization may carry additional meaning and some tools and methods will (tacitly again) operate differently when encountering upper- or lower-case letters. Curiously enough -- and curious is a word we use to avoid mentioning how we feel about it -- some communities chose to designate the lower-case letters as the non-repetitive and the upper-case letters to indicate repetitive regions. Needless to say, confusion can rule the game.

# Is there more information in FASTA headers?

Some data repositories will format FASTA headers to include structured information. Tools may operate differently when this information is present in the FASTA header. For example, NCBI may include both gi and gb accession numbers `>gi|10141003|gb|AF086833.2|` .

Specifically the Blast tool from NCBI is able to query data in more granular fashion when this extra information embedded via header formatting is available. Below is a list of the recognized FASTA header formats.

| Type | Format(s) [1] | Example(s) |
|---|---|---|
| local | `lcl\|integer`<br>`lcl\|string` | `lcl\|123`<br>`lcl\|hmm271` |
| GenInfo backbone seqid | `bbs\|integer` | `bbs\|123` |
| GenInfo backbone moltype | `bbm\|integer` | `bbm\|123` |
| GenInfo import ID | `gim\|integer` | `gim\|123` |
| GenBank | `gb\|accession\|locus` | `gb\|M73307\|AGMA13GT` |
| EMBL | `emb\|accession\|locus` | `emb\|CAM43271.1\|` |
| PIR | `pir\|accession\|name` | `pir\|\|G36364` |
| SWISS-PROT | `sp\|accession\|name` | `sp\|P01013\|OVAX_CHICK` |
| patent | `pat\|country\|patent\|sequence` | `pat\|US\|RE33188\|1` |
| pre-grant patent | `pgp\|country\|application-number\|seq-number` | `pgp\|EP\|0238993\|7` |
| RefSeq [2] | `ref\|accession\|name` | `ref\|NM_010450.1\|` |
| general database reference | `gnl\|database\|integer`<br>`gnl\|database\|string` | `gnl\|taxon\|9606`<br>`gnl\|PID\|e1632` |
| GenInfo integrated database | `gi\|integer` | `gi\|21434723` |
| DDBJ | `dbj\|accession\|locus` | `dbj\|BAC85684.1\|` |
| PRF | `prf\|accession\|name` | `prf\|\|0806162C` |
| PDB | `pdb\|entry\|chain` | `pdb\|1I4L\|D` |
| third-party GenBank | `tpg\|accession\|name` | `tpg\|BK003456\|` |
| third-party EMBL | `tpe\|accession\|name` | `tpe\|BN000123\|` |
| third-party DDBJ | `tpd\|accession\|name` | `tpd\|FAA00017\|` |

# Is there more information in the FASTA sequences?

Lower-case letters may be used to indicate repetitive regions for the genome for example `ATGCATGCagctagctATGTATGC`. In this case `agctagct` is indicated as being present in multiple location of the larger genome. That being said, it is typically not easy to find out what exactly the "repetitiveness" consists of and how it was determined. Usually, it is the case that the "repetitiveness" in a sequence has been determined by a tool run with certain parameters and that has marked the sequence as such.

**Importantly**, some tools like `lastz` will, by default, skip over regions that are lower-cased - this can cause lots of confusion if not recognized right away.

# FASTQ format

## What is the FASTQ format?

The FASTQ format is the de facto standard by which all sequencing instruments represent data. It may be thought of as a variant of the FASTA format that allows it to associate a quality measure to each sequence base, FASTA with QUALITIES.

In simpler terms, it is a format where for every base, we associate a reliability measure: base is `A` and the probability that we are wrong is `1/1000`. Conceptually, the format is very similar to FASTA but suffers from even more flaws than the FASTA format.

We have to recognize that most bioinformatics data formats are adopted via a mixture of past consensus and historical precedent. Most formats may appear to be good enough when first proposed, yet the fast advancements may render them quickly inadequate. The lack of central authority means that even inefficient formats endure past their prime.

Important: The FASTQ format is a multi-line format just as the FASTA format is. In the early days of high-throughput sequencing, instruments always produced the entire FASTQ sequence on a single line (on account of them being so short 35-50bp) In addition the FASTQ format suffers from the unexpected flaw that the `@` sign is both a FASTQ record separator and a valid value of the quality string. For that reason it is a little more difficult to design a correct FASTQ parsing program.

A surprising number of bioinformaticians still operate under the assumption that the FASTQ format is line-oriented and that the entire sequence must always be on a single line, and that the FASTQ record consists of 4 lines. In addition, a large number of "guides" and "tutorials" will tacitly assume this and will show you code that operates on a line by line basis.

While having sequences on a single line greatly simplifies operations, do remember that FASTQ format is not required to be line-oriented! Many short reads instruments will produce them as such, but be advised that this is not a requirement.

The FASTQ format consists of 4 sections (and these are usually produced a single line each):

1. A FASTA-like header, but instead of the `>` symbol it uses the `@` symbol. This is followed by an ID and more optional text, similar to the FASTA headers.
2. The second section contains the measured sequence (typically on a single line), but it may be wrapped until the `+` sign starts the next section.
3. The third section is marked by the `+` sign and may be optionally followed by the same sequence id and header as the first section
4. The last line encodes the quality values for the sequence in section 2, and must be of the same length as section 2. It should (must?) also be wrapped the same way as the section 2.

An example of a single FASTQ record as seen in the Wikipedia entry:

```
@SEQ_ID
GATTTGGGGTTCAAAGCAGTATCGATCAAATAGTAAATCCATTTGTTCAACTCACAGTTT
+
!''*((((***+))%%%++)(%%%%).1***-+*''))**55CCF>>>>>>CCCCCCC65
```

The weird characters in the 4th section (line) are the so called "encoded" numerical values. In a nutshell, each character `!''*((((` represents a numerical value: a so-called Phred score, encoded via a single letter encoding. It is, frankly, a convoluted way of doing something that should be simple. This is a cautionary tale of what happens when scientists start to design data formats.

The idea behind the Phred score is to map two digit numbers to single characters so that the length of the quality string stays the same as the length of the sequence.

Each character has a numerical value, say `!` will be mapped to `0`, `+` will be remapped to mean `10` and say `5` will be mapped to `30` and `I` will be mapped to `40`. There is a convoluted logic behind it, but it is best if that is ignored, and we just take the remappings "as is". The scale of characters to value mappings looks like this:

```
!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHI
|    |    |    |    |    |    |    |    |
0....5...10...15...20...25...30...35...40
|    |    |    |    |    |    |    |    |
worst.............................best
```

The numbers on the scale from `0` to `40` are called Phred scores, and they represent error probabilities.

# Are there different versions of the FASTQ encoding?

Unfortunately, yes. There was a time when instrumentation makers could not decide at what character to start the scale. The current standard shown above is the so-called Sanger ( `+33` ) format where the ASCII codes are shifted by 33. There is the so-called `+64` format that starts close to where the other scale ends. In this scaling `@` is mapped to `0` and `i` is mapped to `40` like so:

```
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghi
|    |    |    |    |    |    |    |    |
0....5...10...15...20...25...30...35...40
|    |    |    |    |    |    |    |    |
worst.............................best
```

Most confusingly, in this encoding the characters that indicate the highest base quality in Sanger will indicate one of the lowest qualities. It is easy to recognize this, though. When our FASTQ quality contains lower case letters `abcdefghi` it means that your data is in this older format. Some tools will allow you to set parameters to account for this different encoding. For others, we will need to convert the data to the correct encoding. You can use `seqtk` for example:

```
seqtk seq -Q64 input.fq > output.fa
```

# Is there more information in FASTQ headers?

Just as with FASTA headers, information is often encoded in the "free" text section of a FASTQ file. For example (see the same Wikipedia entry on FASTQ formats):

```
@EAS139:136:FC706VJ:2:2104:15343:197393 1:Y:18:ATCACG
```

Contains the following information:

- `EAS139` the unique instrument name
- `136` the run id
- `FC706VJ` the flowcell id
- `2` flowcell lane
- `2104` tile number within the flowcell lane
- `15343` 'x'-coordinate of the cluster within the tile
- `197393` 'y'-coordinate of the cluster within the tile
- `1` the member of a pair, 1 or 2 (paired-end or mate-pair reads only)
- `Y` Y if the read is filtered, N otherwise
- `18` 0 when none of the control bits are on, otherwise it is an even number
- `ATCACG` index sequence

This information is specific to a particular instrument/vendor and may change with different versions or releases of that instrument.

This information can be useful, however, to identify quality control problems or systematic technical issues that might affect the instruments.

# What is a Phred score?

A Phred score `Q` is used to compute the probability of a base call being incorrect by the formula `P=10^(-Q/10)`.

You don't need to remember the formula - there is a much simpler way to remember it; like so (where the multiplier to 10 tells you the order of magnitude):

```
 Q     Error       Accuracy
 0    1 in 1           0%
10    1 in 10         90%
20    1 in 100        99%
30    1 in 1000     99.9%
40    1 in 10,000   99.99%
```

# How do I convert FASTQ quality codes at the command line?

When it comes to visually evaluating and inspecting data we rarely need to find out precisely what a value actually decodes to. The simplest is to remember that:

- `!"#$%&'()*+,-.` means low quality `1/10`
- `-./0123456789` means medium quality `1/100`
- `ABCDEFGHI` means high quality `1/1000`

If you need to convert exactly, you may use the following command line shortcuts:

Find the integer value that corresponds to character `A` (prints `65`):

```
python -c 'print (ord("A"))'
```

Find character that corresponds to value `65` (prints `A`):

```
python -c 'print (chr(65))'
```

Find Phred quality score by the Sanger encoding of letter `A` (prints `32`)"

```
python -c 'print (ord("A")-33)'
```

Find Sanger encoded letter that corresponds to Phred quality score `32` (prints `A`):

```
python -c 'print (chr(32+33))'
```

Compute the error probability represented by the Pred quality score `32` (prints `0.00063095734448`):

```
python -c 'from math import*; print (10**-(32/10.0))'
```

Compute the error probability represented by letter `A` (prints `0.00063095734448`):

```
python -c 'from math import*; print (10**-((ord("A")-33)/10.0))'
```

Compute the Phred quality score that corresponds to probability `p=0.00063095734448` (prints `32`)

```
python -c 'from math import*; print (int(round(-10*log(0.00063095734448)/log(10))))'
```

Compute the Sanger encoding that corresponds to probability `p=0.00063095734448` (prints `A`)

```
python -c 'from math import*; print (chr(int(round(-10*log(0.00063095734448)/log(10))+33)))'
```

Due to the imprecision of the floating point representation of numbers, a rounding of the results is required before converting to integer. The extra call to `int` is required only when using Python 2. When using Python 3 `round` alone will do.

# Data Submission

Authors: Aswathy Sebastian, Istvan Albert

To publish analysis results, researchers are required to submit their data sets to appropriate public data repositories.

## Where do we submit what?

There are many repositories for data submission.

- Sequence Read Archive (SRA)
- Gene Expression Omnibus (GEO)
- Database of Short Genetic Variations (dbSNP)
- Database of Genomic Structural Variations (dbVar)
- Database of Expressed Sequence Tags (dbEST)
- Transcriptome Shotgun Assembly Sequence Database (TSA)
- Whole Genome Shotgun Submissions (WGS)
- Metagenomes
- GenBank
- Genomes

With many data repositories out there, it can be difficult to decide the appropriate repository for your data submission.

In short,

- Raw sequence reads from next generation sequencing should be submitted to SRA.
- Functional genomic data like RNASeq, ChIP-seq etc. should be submitted to GEO.
- Variation specific data should be submitted to dbSNP.
- Transcriptome assemblies should be submitted to TSA.
- Most other types of genomic data can be submitted to GenBank or one of its divisions.

A useful resource that summarizes the submission tools and documentation in NCBI can be found here

## Where should I submit high throughput sequence data?

**Sequence Read Archive (SRA)** accepts and archives raw sequence data and alignment information from high throughput sequencing platforms including 454, illumina, SOLiD and PacBio. Details

## Where should I submit RNA-Seq or ChIP-Seq data?

**Gene Expression Omnibus (GEO)** is a functional genomics data repository. It accepts next-generation sequence data that examines gene expression, gene regulation, epigenomics, as well as data from methods like RNA-Seq, ChIP-seq, RIP-seq, HiC-seq, methyl-seq, etc. GEO also accepts expression data such as microarray, SAGE or mass spectrometry data sets. Details

# Where should I submit genetic variation data?

**Database of Short genetic Variations (dbSNP)** accepts variation data, such as single nucleotide variations, microsatellites, and small-scale insertions and deletions. It contains population-specific frequency and genotype data, as well as experimental conditions and mapping information for both neutral variations and clinical mutations. Details

**Database of Genomic Structural Variation (dbVar)** accepts genomic structural variation data, such as large insertions, deletions, translocations etc. Details

# Where should I submit genome assemblies?

**Genomes and WGS** are divisions of GenBank that accept genome assemblies. Incomplete genome assemblies can be submitted to WGS while assemblies that are essentially complete (eg: complete prokaryotic chromosome) should be submitted to Genomes. Genomes WGS

# Where should I submit transcriptome assemblies?

**Transcriptome Shotgun Assembly Sequence Database (TSA)** is a division of GenBank that accepts and archives transcriptome assemblies. In this case, the primary data that produces these assemblies such as ESTs, next-gen data, etc. should be submitted separately to the appropriate databases. More on TSA submission here

# Where should I submit metagenomes?

Submission of a metagenome project to **NCBI** involves several databases. Raw sequence data should be submitted to SRA, assembled contigs or scaffolds of taxonomically defined organisms should be submitted to WGS/Genome, and other supporting sequences such as 16S ribosomal RNAs or fosmids should be submitted to regular GenBank. Details

**EBI Metagenomics** is another place where you can submit raw metagenome sequence and associated metadata. Accession numbers will be provided for the submitted sequences. Once the data is submitted, it can be analyzed using EBI metagenome analysis pipeline. More

# Where should I submit ESTs?

**Database of Expressed Sequence Tags (dbEST)** is a division of GenBank that accepts Expressed Sequence Tags (ESTs). ESTs are short (usually < 1000 bp) sequence reads from mRNA (cDNA). Details about submitting ESTs can be found here

# What are some challenges in data submission ?

Submitting data to public repositories can be a very tedious process. Some challenges that you may come across while submitting your data are:

- Usually it requires many steps and recording minute details of the project. Projects involve many people, and the person submitting the data may not be aware of all the finer details of the project. This necessitates the submitter to collect information from multiple sources. The submitter cannot proceed from one step to the next unless each step is successfully completed. This can make for a time-consuming process.
- Another challenge is that some required details may not be applicable to the project, and the submitter will have to make a decision of what to put in.
- Having to make some edits once the data is submitted can be painful too, as it requires emailing the database facilitator. This may take some back and forth communication before it can finally be done.

However, you do not have to approach the data submission process with a weary mind. Data submission is not an everyday task. You will have to do it only occasionally. These are some precautions for when you have to do it - their purpose is to help you be aware of the challenges and be patient through the process.

# How do I submit to SRA ?

An outline of SRA submission process is below. The submitter should have an NCBI account to start the process.

1. Register BioProject.
2. Register BioSample.
3. SRA submission. (Fill in the meta-data details at this step).
4. Upload files.

# How do I submit to GEO ?

From our experience, the GEO data submission process is more straightforward than SRA submission process.

The submitter should have an NCBI account to start the process. To submit illumina data to GEO, one needs to have the following three pieces of information:

- The metadata spreadsheet.
- The processed data files.
- The raw data files.

An outline of GEO submission is below:

1. Download and fill in the metadata spreadsheet.
2. Upload files.
3. Once the submission is complete, email GEO with the following information.
    - GEO account username.
    - Names of the directory and files deposited.
    - Public release date.

# Where to get to what data

(TODO: this page needs more work) (Yes, it needs quite a bit of editing, which I'm not doing in this proofreading run - RamRS)

First, there is often substantial overlap between the type and amount of data that different data sources offer. That being said, not all resources are equally well set up to help you find the data that you need. We tend to learn via trial-and-error which source works better for what.

At the same time, each of us bioinformaticians develops habits and customs for how we like to get our data from resources.

When working at the command line, access to a file transfer protocol service (FTP) is especially useful.

# What can I get from NCBI?

- NCBI web: https://www.ncbi.nlm.nih.gov/
- NCBI FTP: ftp://ftp.ncbi.nlm.nih.gov/

NCBI and Entrez are the best resource to obtain any sequence that humanity has ever found and stored. But in handling that immense variety, you can forget about expecting any naming scheme that "makes sense". Chromosome 1 of the human genome will not be called as such - and you will have a hard time (at least we do) fishing out which one of the sequences corresponds to that.

Entrez is excellent at getting you any version of any gene that you know of.

# What can I get from ENSEMBL?

- Ensembl web: http://useast.ensembl.org/index.html
- Ensembl FTP: ftp://ftp.ensembl.org/pub/release-86/

Ensembl has the most detailed annotations of the genomes. Ensembl operates via releases.

If you wanted to download the human transcriptome according to Ensembl you could do that with:

```
curl -O ftp://ftp.ensembl.org/pub/release-86/gtf/homo_sapiens/Homo_sapiens.GRCh38.86.gtf.gz
```

# What can I get from BioMart?

Biomart: http://www.ensembl.org/biomart/martview/

(TODO)

# What can I get from UCSC table browser?

UCSC is the only repository that stores large scale result files produced by scientific consortia such as ENCODE. It is also the only site that maintains multiple sequence alignment datasets across entire genomes.

UCSC offers unique services such as "liftOver", transforming coordinates across genomic builds.

UCSC Downloads: http://hgdownload.cse.ucsc.edu/downloads.html UCSC FTP: ftp://hgdownload.cse.ucsc.edu/goldenPath/

# Automating access to NCBI

## What is Entrez?

The NCBI data store is a repository of gargantuan proportions that stores data far beyond the primary DNA data that the NCBI's original mission called for.

Currently, NCBI provides storage, categorization, and context on just about all life-sciences-oriented information, from raw datasets to curated results. It even stores the content of most of the scientific publications themselves.

> Entrez is NCBI's primary text search and retrieval system that integrates the PubMed database of biomedical literature with 39 other literature and molecular databases including DNA and protein sequence, structure, gene, genome, genetic variation, and gene expression.

With the large size come the challenges - the interfaces can be cumbersome and complicated; it is typically not easy to find the information that one is looking for and is easy to get lost across the many pages. This makes command line access even more important as it makes actions explicit and reproducible.

## How is Entrez pronounced?

Technically, it is a French word, and the French pronunciation would sound similar to "on tray". We recommend pronouncing it as it is written: "en-trez"

## How do we automate access to Entrez?

NCBI offers a web API interface called Entrez E-utils and a toolset called Entrez Direct. See the installation page for installing `entrez direct` .

Biostar News of the Day: Ncbi Releases Entrez Direct, The Entrez Utilities On The Unix Command Line

> NCBI has just released Entrez Direct, a new software suite that enables users to use the UNIX command line to directly access NCBI databases, as well as to parse and format the data to create customized downloads.

## How is data organized in NCBI?

The diversity of data sources and the need to keep up with an evolving body of knowledge poses challenges when trying to to identify current and past information in an unambiguous way.

As an example, search the NCBI nucleotide database for the words Ebola virus 1976. Among the first hits will be data that starts with:

```
LOCUS       AF086833               18959 bp    cRNA    linear   VRL 13-FEB-2012
DEFINITION  Ebola virus - Mayinga, Zaire, 1976, complete genome.
ACCESSION   AF086833
VERSION     AF086833.2  GI:10141003
```

Note the accession number `AF086833` and the version number `AF086833.2`

> An accession number, for example `AF086833`, applies to the complete database record and remains stable even if updates/revisions are made to the record.
>
> The version number is formed by adding a dotted number such as `.2` to the accession number to form: `AF086833.2`. This number is a unique identifier for the sequence data within its record.
>
> If any change occurs to the sequence data, no matter how large or small, the version number for that sequence is incremented by one decimal.

Starting in 2015, NCBI has decided to phase out an older standard called `GI` numbers that were integer numbers associated with a record. Older document may still refer to `GI` or `gi` numbers.

# How do I use Entrez E-utils web API?

Entrez web API allows us to query NCBI data sources via a specially constructed URL. A query URL will be of the form `https://service.nih.gov?param1=value1&param2=value2&param3=value3`

Since the `&` character has a special meaning for the shell we need to either put the URL within single quotes or we need to "escape/protect" the `&` character by placing a `\` in front of it like so `\&`

Returns the the data for accession number `AF086833.2` in the FASTA format:

```
curl -s https://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?id=AF086833.2\&db=nuccore\&r
ettype=fasta | head
```

or equivalently:

```
curl -s 'https://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?id=AF086833.2&db=nuccore&re
ttype=fasta' | head
```

Read the Entrez E-utils documentation for information on all parameters.

# How do I use Entrez Direct?

The Entrez E-utils can be unwieldy and are not well suited for exploratory analysis. The tool suite called entrez-direct (EDirect) simplifies this web access via a series of tools, including `efetch` and others:

```
efetch -db=nuccore -format=gb -id=AF086833 | head
```

We can retrieve the same information in different formats:

```
# Accession number AF086833 in Genbank format.
efetch -db=nuccore -format=gb -id=AF086833 > AF086833.gb

# Accession number AF086833 in Fasta format.
efetch -db=nuccore -format=fasta -id=AF086833 > AF086833.fa

# efetch can take additional parameters and select a section of the sequence.
efetch -db=nuccore -format=fasta -id=AF086833 -seq_start=1 -seq_stop=3
```

It can even produce the sequence from reverse strands:

```
efetch -db=nuccore -format=fasta -id=AF086833 -seq_start=1 -seq_stop=5 -strand=1
efetch -db=nuccore -format=fasta -id=AF086833 -seq_start=1 -seq_stop=5 -strand=2
```

But just what exactly did `strand=2` do? Is it the complement or reverse complement? Always ensure you understand what parameters do. Don't assume you understand their function. Below are the actual outputs of the last two commands. We can see that the second command produces the reverse complement as indicated by the `c5-1` tag in the FASTA file header.

```
>gb|AF086833.2|:1-5 Ebola virus - Mayinga, Zaire, 1976, complete genome
CGGAC

>gb|AF086833.2|:c5-1 Ebola virus - Mayinga, Zaire, 1976, complete genome
GTCCG
```

# How do we search with Entrez Direct?

Once we have a project accession number, say `PRJNA257197`, we can use it to search for the data that comes with it. Project accession numbers are included in published papers, or we find them in the supporting information.

```
esearch -help
esearch -db nucleotide -query PRJNA257197
esearch -db protein -query PRJNA257197
```

These commands produce a so called "environment" that can be passed into other Entrez direct programs. You can get a sense of what the results are from looking at the results of the search:

```
<ENTREZ_DIRECT>
  <Db>nucleotide</Db>
  <WebEnv>NCID_1_17858181_130.14.18.34_9001_1472654178_1691087027_0MetA0_S_MegaStore_F_1</WebE
nv>
  <QueryKey>1</QueryKey>
  <Count>249</Count>
  <Step>1</Step>
</ENTREZ_DIRECT>
```

To fetch the data for a search, pass it into `efetch`:

```
esearch -db nucleotide -query PRJNA257197 | efetch -format=fasta > genomes.fa
```

To get all proteins:

```
esearch -db protein -query PRJNA257197 | efetch -format=fasta > proteins.fa
```

# Can we extract more complex information with Entrez Direct?

(TODO: expand)

Entrez Direct is a very sophisticated tool with surprising powers - though to fully use it, one needs to have a firm understanding of the XML file format that `efetch` returns.

Specifically, the `xtract` tool in Entrez Direct allows navigating and selecting parts of an XML file.

```
efetch -db taxonomy -id 9606,7227,10090 -format xml | xtract -Pattern Taxon -first TaxId Scien
tificName GenbankCommonName Division
```

Produces:

```
9606    Homo sapiens    human    Primates
7227    Drosophila melanogaster    fruit fly    Invertebrates
10090    Mus musculus    house mouse    Rodents
```

# Automating data access with Entrez Direct

(TODO, migrating content from the ncbi automation document)

# How do I use Entrez Direct?

The Entrez E-utils can be unwieldy and are not well suited for exploratory analysis. The tool suite, called entrez-direct (EDirect), simplifies this web access via a series of tools titled `efetch`, `esearch`, etc. Here's an example of `efetch`:

```
efetch -db=nuccore -format=gb -id=AF086833 | head
```

We can retrieve the same information in different formats:

```
# Accession number AF086833 in Genbank format.
efetch -db=nuccore -format=gb -id=AF086833 > AF086833.gb

# Accession number AF086833 in Fasta format.
efetch -db=nuccore -format=fasta -id=AF086833 > AF086833.fa

# efetch can take additional parameters and select a section of the sequence.
efetch -db=nuccore -format=fasta -id=AF086833 -seq_start=1 -seq_stop=3
```

It can even produce the sequence from reverse strands:

```
efetch -db=nuccore -format=fasta -id=AF086833 -seq_start=1 -seq_stop=5 -strand=1
efetch -db=nuccore -format=fasta -id=AF086833 -seq_start=1 -seq_stop=5 -strand=2
```

But just what exactly did `strand=2` do? Is it the complement or reverse complement? Always make sure to understand what parameters do. Don't just assume you understand their function. Below are the actual outputs of the last two commands. We can see that the second command produces the reverse complement as indicated by the `c5-1` tag in the FASTA file header.

```
>gb|AF086833.2|:1-5 Ebola virus - Mayinga, Zaire, 1976, complete genome
CGGAC

>gb|AF086833.2|:c5-1 Ebola virus - Mayinga, Zaire, 1976, complete genome
GTCCG
```

# How do we search with Entrez Direct?

Once we have a project accession number, say `PRJNA257197`, we can use it to search for the data that comes with it. Project accession numbers are included in published papers, or we find them in the supporting information.

```
esearch -help
esearch -db nucleotide -query PRJNA257197
esearch -db protein -query PRJNA257197
```

These commands produce a so called "environment" that can be passed into other Entrez direct programs. You can get a sense of what the results are from looking at the results of the search:

```
<ENTREZ_DIRECT>
  <Db>nucleotide</Db>
  <WebEnv>NCID_1_17858181_130.14.18.34_9001_1472654178_1691087027_0MetA0_S_MegaStore_F_1</WebE
nv>
  <QueryKey>1</QueryKey>
  <Count>249</Count>
  <Step>1</Step>
</ENTREZ_DIRECT>
```

To fetch the data for a search, pass it into `efetch`:

```
esearch -db nucleotide -query PRJNA257197 | efetch -format=fasta > genomes.fa
```

To get all proteins:

```
esearch -db protein -query PRJNA257197 | efetch -format=fasta > proteins.fa
```

# Can we extract more complex information with Entrez Direct?

(TODO: expand)

Entrez Direct is a very sophisticated tool with surprising powers - though to fully use it, one needs to have a firm understanding of the XML file format that `efetch` returns.

Specifically, the `xtract` tool in Entrez Direct allows navigating and selecting parts of an XML file.

```
efetch -db taxonomy -id 9606,7227,10090 -format xml | xtract -Pattern Taxon -first TaxId Scien
tificName GenbankCommonName Division
```

Produces:

```
9606    Homo sapiens    human    Primates
7227    Drosophila melanogaster    fruit fly    Invertebrates
10090    Mus musculus    house mouse    Rodents
```

# Accessing the Short Read Archive (SRA)

## What is the Short Read Archive (SRA)?

The Short Read Archive (SRA) is a data storage service provided by NCBI. The history of the SRA's development is fairly convoluted and includes a widely announced, promoted, then retracted cancellation of the service. In parallel, the data design and usage interface were standardized perhaps prematurely, before the community fully understood how it might need to access the data. The incredible rate of growth of the deposited data locked and froze those standards in place. Today it seems that it would be hard to change them without inconveniencing a large number of people. No wonder there is an entire "cottage industry" of tools trying to bypass and/or automate access to SRA.

## What are the SRA naming schemes?

Data in the SRA is organized under a hierarchical structure where each top category may contain one or more subcategories:

- NCBI BioProject: `PRJN****` (example: `PRJNA257197`) contains the overall description of a single research initiative; a project will typically relate to multiple samples and datasets.
- NCBI BioSample: `SAMN****` and/or `SRS****` (example: `SAMN03254300`) describe biological source material; each physically unique specimen should be registered as a single BioSample with a unique set of attributes.
- SRA Experiment: `SRX****` a unique sequencing library for a specific sample
- SRA Run: `SRR****` or `ERR****` (example: `SRR1553610`) is a manifest of data file(s) linked to a given sequencing library (experiment).

## How do we download data from SRA?

We can download data via a web browser (after installing a web-plugin) or from the command line via the sratoolkit package.

## Are there alternatives to the SRA?

The European Nucleotide Archive (ENA) provides an alternative repository to store the world's nucleotide sequencing information, covering raw sequencing data, sequence assembly information, and functional annotation. The naming schemes and the organization of the data mirror those of the SRA, though the files are stored directly as FASTQ and BAM formats and do not need conversion via the `sratoolkit`.

A more user-friendly interface is provided by the DNAnexus SRA site and is hosted by DNA Nexus company. This interface allows users to search and filter all the publicly accessible Sequence Read Archive data. The data itself is still hosted at NCBI. Their search interface returns a list of URLs that

contain the data at NCBI.

# Where is the SRA documentation?

- The SRA Handbook contains a detailed description of the Short Read Archive.
- The SRA Toolkit Documentation describes the command line tools that may be used to access data.
- SRA Toolkit GitHub may have more up-to-date information on troubleshooting.

# How does the sratoolkit work?

There are several tools (see the tool directory). The most commonly used ones are:

- `fastq-dump`, downloads data in FASTQ format
- `sam-dump`, downloads data in SAM alignment format

A typical usage case is

```
fastq-dump SRR1553607
```

This creates the file:

```
SRR1553607.fastq
```

It is worth looking at this file with `fastqc` to understand the approach and quality of the sequencing data.

By default, the SRA data will concatenate the paired reads because that is actually how the instrument measured them as well. For paired end reads the data needs to be separated into different files:

```
fastq-dump --split-files  SRR1553607
```

This creates the paired end files:

```
SRR1553607_1.fastq
SRR1553607_2.fastq
```

The `fastq-dump` command does two things in sequence. It first downloads the data in the so-called SRA format, then it converts it to FASTQ format. We can ask the tool to convert just a subset of data; for example, the first `10,000` reads:

```
fastq-dump --split-files -X 10000 SRR1553607
```

But we note that even in that case, it needs to first download the full data file.

# Is it possible to download the SRA files from a server?

Yes, there is a website called ftp://ftp-trace.ncbi.nih.gov/sra/sra-instant that uses an encoding system to map a SRR number to a URL. For example, `SRR1972739` can be downloaded as:

```
curl -O ftp://ftp-trace.ncbi.nih.gov/sra/sra-instant/reads/ByRun/sra/SRR/SRR197/SRR1972739/SRR1972739.sra
```

This will download a file named `SRR1972739.sra`. This file can be operated on the same way with fastq-dump:

```
fastq-dump -X 10000 --split-files SRR1972739.sra
```

# Where do downloads go?

When any download via `fastq-dump` is initiated, the **entire** data for that run id is first downloaded to a location on your computer. By default this location is at `~/ncbi/public/sra/`. There may be other sequences related to an SRA file, in which case those too will be downloaded. To investigate the downloaded SRA data do:

```
ls -l ~/ncbi/public/sra/
```

After the download is completed, a conversion takes place according to the parameters.

> Note: Even if we only asked for the first 10 reads of a FASTQ file, the entire FASTQ content would need to be first downloaded to the hidden location then out of that the first 10 reads will be shown. For large datasets, this may take a considerable amount of time and may use up a substantial amount of disk space.

Now, if we were to access the same SRA run the second time, the data would already be there and conversion would be much faster. This is because the download needs to take place only once.

# What is a "spot" in SRA?

SRA tools report most results in terms of "spots" and not as "reads" (measurements), though the two almost always seem to mean the same thing. Answers to the Biostar question: What Is A "Spot" In Sra Format? shed some light:

"The spot model is Illumina GA centric. [...] All of the bases for a single location constitute the spot."

As it turns out, the terminology is both confusing and may not be applicable to other instruments.

In the Illumina sequencers, a single location on the flow cell generates the first read, the barcode (if that exists), and the second (paired) read (if the run was a paired end sequencing run).

We almost always split the spots (via the `--split-files` option) into originating files.

# How do we get information on the run?

The `sra-stat` program can generate an XML report on the data.

```
sra-stat --xml --quick SRR1553610
```

Get read length statistics on a PacBio dataset:

```
sra-stat --xml --statistics SRR4237168
```

# Automating access to the Short Read Archive (SRA)

In this chapter, we demonstrate how to automate access to the SRA and answer questions such as - how many sequencing runs have been deposited so far? How much data is stored in the SRA?

## How can we automate downloading multiple SRA runs?

Obtain a list of `SRR` SRA sequencing run numbers, then run `fastq-dump` on each.

Start with the bioproject number `PRJNA257197` for the Zaire ebolavirus Genome sequencing

```
esearch -db sra -query PRJNA257197
```

This tells us that there are `891` sequencing runs for this project id:

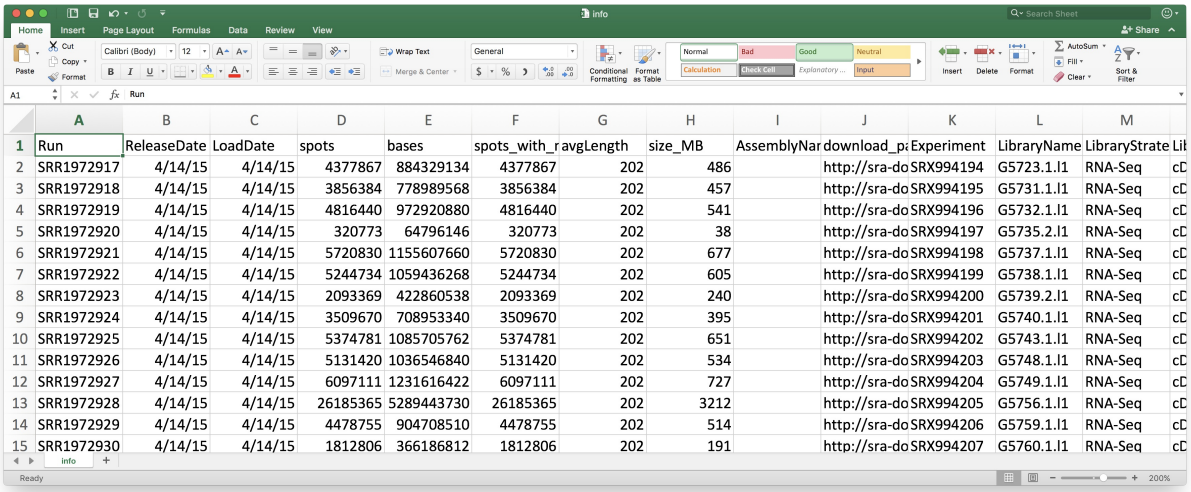
```
<ENTREZ_DIRECT>
  <Db>sra</Db>
  <WebEnv>NCID_1_74789875_130.14.18.34_9001_1474294072_501009688_0MetA0_S_MegaStore_F_1</WebEnv
>
  <QueryKey>1</QueryKey>
  <Count>891</Count>
  <Step>1</Step>
</ENTREZ_DIRECT>
```

Format the output as a so-called RunInfo type (comma separated values). :

```
esearch -db sra -query PRJNA257197 | efetch -format runinfo > info.csv
```

The command may take a surprisingly long time (30 seconds) to complete, even though it is just a simple table. The resulting CSV file is easiest to interpret when opened with Excel (or other spreadsheet tools) though we should warn you that Excel as a data analysis tool has quite the bad reputation among bioinformaticians. Most of that is just us, bioinformaticians, being snobs. Now it is true that Excel has a number of very annoying features - for example, it attempts to convert certain words into other types. A famous example is gene named SEP9, Excel converts it automatically to September 9 - or that it trims off leading zeros from numbers. For a recent treatise, see the paper titled Gene name errors are widespread in the scientific literature (Genome Biology, 2016) That being said, it's just as easy to make mistakes at the command line (if not more so) as Excel at least visually shows the results. We get riled up about Excel primarily since it silently introduces errors - which feels like a betrayal of sorts.

The resulting `info.csv` file is quite rich in metadata and contains a series of attributes for the run:



| | A | B | C | D | E | F | G | H | I | J | K | L | M |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Run | ReleaseDate | LoadDate | spots | bases | spots_with_r | avgLength | size_MB | AssemblyNar | download_pa | Experiment | LibraryName | LibraryStrate |
| 2 | SRR1972917 | 4/14/15 | 4/14/15 | 4377867 | 884329134 | 4377867 | 202 | 486 | | http://sra-do | SRX994194 | G5723.1.l1 | RNA-Seq |
| 3 | SRR1972918 | 4/14/15 | 4/14/15 | 3856384 | 778989568 | 3856384 | 202 | 457 | | http://sra-do | SRX994195 | G5731.1.l1 | RNA-Seq |
| 4 | SRR1972919 | 4/14/15 | 4/14/15 | 4816440 | 972920880 | 4816440 | 202 | 541 | | http://sra-do | SRX994196 | G5732.1.l1 | RNA-Seq |
| 5 | SRR1972920 | 4/14/15 | 4/14/15 | 320773 | 64796146 | 320773 | 202 | 38 | | http://sra-do | SRX994197 | G5735.2.l1 | RNA-Seq |
| 6 | SRR1972921 | 4/14/15 | 4/14/15 | 5720830 | 1155607660 | 5720830 | 202 | 677 | | http://sra-do | SRX994198 | G5737.1.l1 | RNA-Seq |
| 7 | SRR1972922 | 4/14/15 | 4/14/15 | 5244734 | 1059436268 | 5244734 | 202 | 605 | | http://sra-do | SRX994199 | G5738.1.l1 | RNA-Seq |
| 8 | SRR1972923 | 4/14/15 | 4/14/15 | 2093369 | 422860538 | 2093369 | 202 | 240 | | http://sra-do | SRX994200 | G5739.2.l1 | RNA-Seq |
| 9 | SRR1972924 | 4/14/15 | 4/14/15 | 3509670 | 708953340 | 3509670 | 202 | 395 | | http://sra-do | SRX994201 | G5740.1.l1 | RNA-Seq |
| 10 | SRR1972925 | 4/14/15 | 4/14/15 | 5374781 | 1085705762 | 5374781 | 202 | 651 | | http://sra-do | SRX994202 | G5743.1.l1 | RNA-Seq |
| 11 | SRR1972926 | 4/14/15 | 4/14/15 | 5131420 | 1036546840 | 5131420 | 202 | 534 | | http://sra-do | SRX994203 | G5748.1.l1 | RNA-Seq |
| 12 | SRR1972927 | 4/14/15 | 4/14/15 | 6097111 | 1231616422 | 6097111 | 202 | 727 | | http://sra-do | SRX994204 | G5749.1.l1 | RNA-Seq |
| 13 | SRR1972928 | 4/14/15 | 4/14/15 | 26185365 | 5289443730 | 26185365 | 202 | 3212 | | http://sra-do | SRX994205 | G5756.1.l1 | RNA-Seq |
| 14 | SRR1972929 | 4/14/15 | 4/14/15 | 4478755 | 904708510 | 4478755 | 202 | 514 | | http://sra-do | SRX994206 | G5759.1.l1 | RNA-Seq |
| 15 | SRR1972930 | 4/14/15 | 4/14/15 | 1812806 | 366186812 | 1812806 | 202 | 191 | | http://sra-do | SRX994207 | G5760.1.l1 | RNA-Seq |

Cutting out the first column of this file

```
cat info.csv | cut -f 1 -d ',' | head
```

Produces:

```
Run
SRR1972917
SRR1972918
SRR1972919
...
```

To filter for lines that match `SRR` and to store the first ten run ids in a file we could write:

```
cat info.csv | cut -f 1 -d ',' | grep SRR | head > ids.txt
```

Our `ids.txt` file contains:

```
SRR1972917
SRR1972918
SRR1972919
...
```

We now have access to the SRR run ids for the experiment. We can just invoke `fastq-dump` on each:

```
fastq-dump -X 10000 --split-files SRR1972917
fastq-dump -X 10000 --split-files SRR1972918
fastq-dump -X 10000 --split-files SRR1972919
...
```

We are in UNIX world, so naturally different techniques can be used to automate the process above.

The `xargs` command is a tool that can be used to construct other new commands. Here we run the command `echo This is number $1` on each element of the file:

```
cat ids.txt | xargs -n 1 echo This is number $1
```

Will produce:

```
This is number SRR1972917
This is number SRR1972918
This is number SRR1972919
...
```

We can plug the `fastq-dump` command in the place of the `echo` :

```
cat ids.txt | xargs -n 1 fastq-dump -X 10000 --split-files $1
```

To obtain 20 files (two files for each of the 10 runs).

We also have options for filtering it for various metadata in this file. We can, for example, filter for runs with the release data of August 19, 2014:

```
cat info.csv | grep '2014-08-19'  | cut -f 1 -d ',' | grep SRR | head -10 > ids.txt
```

# How do we get information for all runs in the SRA?

Here is where Unix command line tools shine.

We first get the run info via a `esearch` query. When downloading large amounts of data, the best approach, whenever possible is to divide the data into smaller units. For example, we could download it by months with:

```
esearch -db sra -query '"2015/05"[Publication Date]' | efetch -format runinfo > 2015-04.csv
```

Then we just need to automate this line of code to iterate over each year and month pair. We did that and you can obtain the result files via (196MB):

```
curl -O http://data.biostarhandbook.com/sra/sra-runinfo-2016-09.tar.gz
```

We can now answer a series of interesting questions (we may update this data in the future and that may put it out of sync with the rest of the book).

# How many sequencing runs are in the SRA?

Combine all files into a single file; keep only lines that contain SRR numbers:

```
cat sra/*.csv | grep SRR > allruns.csv
```

How many runs are there:

```
cat allruns.csv | wc -l
```

Prints:

```
1,496,565
```

# How do we extract columns from a comma separated file?

Cutting columns on tabs is usually fine, as tabs are not part of normal text. Commas, on the other hand, are commonly used within the text itself. When splitting directly on commas, we may end up lining up the wrong columns. Hence, we need a more sophisticated tool that cuts the file properly: at the "column delimiter" commas and not at the commas that are within a text field (quotes).

In other words, we need a tool that operates on `csv` files:

- `csvkit`
- `miller`

and other similar tools.

In the following commands we will replace `cut` with `csvcut` that comes with the `csvkit` package.

# How do we clean up the data?

In addition (and this is something that unfortunately is an endemic problem of bioinformatics), even though we've selected data from an official source, sometimes (very rarely) the numeric column is invalid and instead of a number, it contains a word.

Thankfully, well designed tools will raise an error that pinpoints the line that caused the error. In our case, one of our later commands raised the following error:

```
datamash: invalid numeric value in line 1453106 field 2: 'avgLength'
```

Basically stating that line `1,453,106` is invalid. If we print out that line with the `sed` tool:

```
cat allruns.csv | sed -n -e 1453106p
```

it shows:

```
SRR3587673,Run,ReleaseDate,LoadDate,spots,bases,spots_with_mates,avgLength ...
```

That line does not look correct. Instead of values, it contains words. It is a mystery why that happened, but we'll remove it and see if the rest works. Of course, that may not be a trivial thing to do - how would you remove line `1,453,106` from a file keeping the rest intact? Unix comes to the rescue. The following command modifies the file in place and removes that line:

```
sed -i -e "1453106d" allruns.csv
```

Thankfully the rest of the files all pass muster.

# Are we data janitors?

In a way, yes.

I used to get worked up about the data problems, but then I read the New York Times opinion piece: For Big-Data Scientists, 'Janitor Work' Is Key Hurdle to Insights, where they state:

## *For Big-Data Scientists, 'Janitor Work' Is Key Hurdle to Insights*

> [.] Yet far too much handcrafted work — what data scientists call "data wrangling," "data munging" and "data janitor work" — is still required. Data scientists, according to interviews and expert estimates, spend from 50 percent to 80 percent of their time mired in this more mundane labor of collecting and preparing unruly digital data, before it can be explored for useful nuggets []

What we had to do above is one of these "janitorial" tasks. From a 600MB file with 1.5 million entries, we first had to find and get rid of that single line that caused the error.

But we are all good now.

# How many sequenced bases are in the SRA?

Now install `datamash` to run this command to summarize the 5th column, the number of sequenced bases in the run:

```
cat allruns.csv |  csvcut -c 5 | datamash sum 1
```

Prints:

```
4.9780612824451e+15
```

That is around 5 PB (peta bases), or 5,000 trillion bases.

Note how often use the pattern:

```
sort | uniq -c | sort -rn
```

We can create a shortcut to this:

```
alias tabulate='sort | uniq -c | sort -rn'
```

Now we can use this shortcut to make the commands simpler.

# How many sequencing runs per organism?

Column 29 of the run info file contains the organism:

```
cat allruns.csv | csvcut -c 29 | tabulate | head -25
```

Produces:

```
480,622 Homo sapiens
141,135 Mus musculus
 47,929 human metagenome
 37,644 soil metagenome
 36,740 human gut metagenome
 26,440 Salmonella enterica
 24,641 Drosophila melanogaster
 21,396 Arabidopsis thaliana
 19,577 Salmonella enterica subsp. enterica
 19,370 Plasmodium falciparum
 18,543 Escherichia coli
 17,331 gut metagenome
 16,854 Saccharomyces cerevisiae
 15,408 Caenorhabditis elegans
 14,197 marine metagenome
 13,821 mouse gut metagenome
 10,658 Listeria monocytogenes
  8,482 Bos taurus
  6,985 human skin metagenome
  6,391 aquatic metagenome
  6,375 freshwater metagenome
  6,308 Danio rerio
  5,794 Streptococcus pyogenes
  5,788 metagenome
  5,691 Zea mays
```

# How many sequencing runs for each sequencing platform?

Column 19 of the run info file contains the sequencing platform name:

```
cat allruns.csv | csvcut -c 19  | tabulate  | head
```

Produces:

```
1,231,420 ILLUMINA
  182,407 LS454
   31,865 ABI_SOLID
   31,223 PACBIO_SMRT
   14,550 ION_TORRENT
    3,736 COMPLETE_GENOMICS
      869 HELICOS
      398 CAPILLARY
       96 OXFORD_NANOPORE
```

# How many runs per sequencing instrument model?

Column 20 of the run info file contains the instrument model name:

```
cat allruns.csv | csvcut -c 20  | tabulate  | head -20
```

Produces:

```
633,701 Illumina HiSeq 2000
203,828 Illumina MiSeq
174,695 Illumina HiSeq 2500
120,715 454 GS FLX Titanium
 99,268 Illumina Genome Analyzer II
 47,066 Illumina Genome Analyzer IIx
 33,187 454 GS FLX
 26,527 NextSeq 500
 25,243 Illumina Genome Analyzer
 17,558 PacBio RS II
 13,665 PacBio RS
 12,302 Ion Torrent PGM
 10,776 454 GS Junior
  9,730 AB SOLiD 4 System
  9,699 454 GS FLX+
  8,430 AB 5500xl Genetic Analyzer
  7,886 Illumina HiSeq 1000
  6,353 454 GS
  5,540 AB SOLiD System 2.0
  4,769 AB 5500 Genetic Analyzer
```

# How big is the largest SRA sequencing run?

The run info contains data on the number of reads (spots) as well as on the size of the data. There is also an invalid entry (perhaps the same as before)

```
cat allruns.csv | csvcut -c 1,8  | datamash -f -t ',' max 2 mean 2 sum 2
```

Prints:

```
SRR923764,533571,533571,1931.7792757276,2891031320
```

This means that the largest file in SRA is `SRR923764`. It has the size of `533571MB` that is `533571/1000=533GB` (technically `533571/1024 = 521.1 GiB`)

The total amount of data in SRA is `2,891,031,320 MB` that is `2.8 PB` (peta bytes).

# FASTA/Q manipulations

Author: **Wei Shen**

This page illustrates common FASTA/Q manipulations using SeqKit. Some other utilities, including csvtk (CSV/TSV toolkit) and shell commands were also used.

Note: SeqKit seamlessly supports FASTA and FASTQ formats both in their original form as well as gzipped compressed format. We list FASTA or FASTQ depending on the more common usage but you can always use it on the other type as well.

# Example data

One FASTQ file (sample reads, 1M) and two FASTA files (Virus DNA and protein sequences from NCBI RefSeq database, 60+40M) are used.

```
wget http://data.biostarhandbook.com/reads/duplicated-reads.fq.gz
wget ftp://ftp.ncbi.nih.gov/refseq/release/viral/viral.1.1.genomic.fna.gz
wget ftp://ftp.ncbi.nih.gov/refseq/release/viral/viral.1.protein.faa.gz
```

# How to produce an overview of FASTQ files?

Sequence format and type are automatically detected.

```
seqkit stat *.gz
```

prints:

```
file                    format  type     num_seqs      sum_len  min_len   avg_len    max_len
duplicated-reads.fq.gz  FASTQ   DNA        15,000    1,515,000      101       101        101
viral.1.1.genomic.fna.gz FASTA  DNA         7,048  203,325,120      200  28,848.6  2,473,870
viral.1.protein.faa.gz  FASTA   Protein   252,611   62,024,702        5     245.5      8,960
```

# How do I get the GC content of every sequence in a FASTA/Q file?

`seqkit fx2tab` coverts FASTA/Q to 3-column tabular format (1st: name/ID, 2nd: sequence, 3rd: quality), and can also provide various information in new columns, including sequence length, GC content/GC skew, alphabet.

GC content:

```
seqkit fx2tab --name --only-id --gc viral*.fna.gz
```

generates:

```
gi|526245010|ref|NC_021865.1|                  40.94
gi|460042095|ref|NC_020479.1|                  41.82
gi|526244636|ref|NC_021857.1|                  49.17
```

Custom bases (A, C and A+C):

```
seqkit fx2tab -H -n -i -B a -B c -B ac viral.1.1.genomic.fna.gz
```

produces:

```
#name   seq     qual    a       c       ac
gi|526245010|ref|NC_021865.1|                  33.20   18.24   51.44
gi|460042095|ref|NC_020479.1|                  32.57   19.63   52.20
gi|526244636|ref|NC_021857.1|                  25.52   23.06   48.59
```

# How to extract a subset of sequences from a FASTA/Q file with name/ID list file?

This is a frequently used manipulation. Let's create a sample ID list file, which may also come from some other method like a mapping result.

```
seqkit sample --proportion 0.001  duplicated-reads.fq.gz | seqkit seq --name --only-id > id.txt
```

ID list file:

```
head id.txt
```

shows:

```
SRR1972739.2996
SRR1972739.3044
SRR1972739.3562
```

Searching by ID list file:

```
seqkit grep --pattern-file id.txt duplicated-reads.fq.gz > duplicated-reads.subset.fq.gz
```

# How do I find FASTA/Q sequences containing degenerate bases and locate them?

`seqkit fx2tab` coverts FASTA/Q to tabular format and can output the sequence alphabet in a new column. Text searching tools can then be used to filter the table.

```
seqkit fx2tab -n -i -a viral.1.1.genomic.fna.gz | csvtk -H -t grep -f 4 -r -i -p "[^ACGT]"
```

prints:

```
gi|446730228|ref|NC_019782.1|                  ACGNT
gi|557940284|ref|NC_022800.1|                  ACGKT
gi|564292828|ref|NC_023009.1|                  ACGNT
```

Long-option version of the command:

```
seqkit fx2tab --name --only-id --alphabet  viral.1.1.genomic.fna.gz | csvtk --no-header-row --
tabs grep --fields 4 --use-regexp --ignore-case --pattern "[^ACGT]"
```

You can then exclude these sequences with `seqkit grep`:

```
# save the sequence IDs.
seqkit fx2tab -n -i -a viral.1.1.genomic.fna.gz | csvtk -H -t grep -f 4 -r -i -p "[^ACGT]" | c
svtk -H -t cut -f 1 > id2.txt

# search and exclude.
seqkit grep --pattern-file id2.txt --invert-match viral.1.1.genomic.fna.gz > clean.fa
```

Or locate the degenerate bases, e.g, `N` and `K`

```
seqkit grep --pattern-file id2.txt viral.1.1.genomic.fna.gz | seqkit locate --ignore-case --on
ly-positive-strand --pattern K+ --pattern N+
```

shows:

```
seqID   patternName     pattern strand  start   end     matched

gi|564292828|ref|NC_023009.1|   N+      N+      +       87972   87972   N
gi|564292828|ref|NC_023009.1|   N+      N+      +       100983  100983  N
gi|557307918|ref|NC_022755.1|   K+      K+      +       1788    1788    K
gi|557307918|ref|NC_022755.1|   K+      K+      +       4044    4044    K
gi|589287065|ref|NC_023585.1|   K+      K+      +       28296   28296   K
gi|590911929|ref|NC_023639.1|   N+      N+      +       741654  741753  NNNNNNNNNNNNNNNNNNNNN
NNNNN
```

# How do I remove FASTA/Q records with duplicated sequences?

```
seqkit rmdup --by-seq --ignore-case duplicated-reads.fq.gz > duplicated-reads.uniq.fq.gz
```

If the FASTA/Q file is very large, you can use the flag `-m/--md5`, which uses MD5 instead of the original sequences to reduce memory usage during sequence comparison.

You can also deduplicate according to sequence ID (default) or full name (`--by-name`).

# How do I locate motif/subsequence/enzyme digest sites in FASTA/Q sequence?

Related posts: Question: Count and location of strings in fastq file reads, Question: Finding TATAWAA in sequence.

Assuming you have a list of motifs (enzyme digest sites) in FASTA format that you would like to locate in your sequences:

```
cat enzymes.fa

>EcoRI
GAATTC
>MmeI
TCCRAC
>SacI
GAGCTC
>XcmI
CCANNNNNNNNNTGG
```

Flag `--degenerate` is on because our patterns of interest contain degenerate bases. Command:

```
seqkit locate --degenerate --ignore-case --pattern-file enzymes.fa viral.1.1.genomic.fna.gz
```

Sample output (simplified and reformated by `csvtk -t uniq -f 3 | csvtk -t pretty`)

```
seqID                          patternName   pattern           strand   start   end     match
ed
gi|526245010|ref|NC_021865.1|  MmeI          TCCRAC            +        1816    1821    TCCGA
C
gi|526245010|ref|NC_021865.1|  SacI          GAGCTC            +        19506   19511   GAGCT
C
gi|526245010|ref|NC_021865.1|  XcmI          CCANNNNNNNNNTGG   +        2221    2235    CCATA
TTTAGTGTGG
```

# How do I sort a huge number of FASTA sequences by length?

Sorting a FASTA file in order of sequence size (small to large).

```
seqkit sort --by-length viral.1.1.genomic.fna.gz > viral.1.1.genomic.sorted.fa
```

If the files are too big, use flag `--two-pass` which will consume less memory.

```
seqkit sort --by-length --two-pass viral.1.1.genomic.fna.gz > viral.1.1.genomic.sorted.fa
```

You can also sort by sequence ID (default), full header ( `--by-name` ) or sequence content ( `--by-seq` ).

# How do I split FASTA sequences according to information in the header?

Related posts: Question: extract same all similar sequences in FASTA based on the header.

For example, the FASTA header line of `viral.1.protein.faa.gz` contains the species name in square brackets.

Overview of FASTA Headers:

```
seqkit head -n 3 viral.1.protein.faa.gz | seqkit seq --name
```

generates:

```
gi|526245011|ref|YP_008320337.1| terminase small subunit [Paenibacillus phage phiIBB_Pl23]
gi|526245012|ref|YP_008320338.1| terminase large subunit [Paenibacillus phage phiIBB_Pl23]
gi|526245013|ref|YP_008320339.1| portal protein [Paenibacillus phage phiIBB_Pl23]
```

`seqkit split` can split FASTA/Q files according to ID, number of parts, size of each part, or sequence region. In this case, we'll split according to sequence ID (species name) which can be specified by flag `--id-regexp` .

Default ID:

```
seqkit head -n 3 viral.1.protein.faa.gz | seqkit seq --name --only-id
```

outputs:

```
gi|526245011|ref|YP_008320337.1|
gi|526245012|ref|YP_008320338.1|
gi|526245013|ref|YP_008320339.1|
```

New ID:

```
seqkit head -n 3 viral.1.protein.faa.gz | seqkit seq --name --only-id --id-regexp "\[(.+)\]"
```

prints:

```
Paenibacillus phage phiIBB_Pl23
Paenibacillus phage phiIBB_Pl23
Paenibacillus phage phiIBB_Pl23
```

Split:

```
seqkit split --by-id --id-regexp "\[(.+)\]" viral.1.protein.faa.gz
```

# How do I search and replace within a FASTA header using character strings from a text file?

Related posts: Question: Replace names in FASTA file with a known character string from a text file, Question: Fasta header, search and replace...?.

`seqKit replace` can find substrings in FASTA/Q headers using regular expressions and replace them with strings or corresponding values of found substrings provided by a tab-delimited key-value file.

For example, to unify names of protein with unknown functions, we want to rename "hypothetical" to "putative" in lower case. The replacing rules are listed below in tab-delimited file:

```
cat changes.tsv
Hypothetical    putative
hypothetical    putative
Putative        putative
```

Overview the FASTA headers containing "hypothetical":

```
seqkit grep --by-name --use-regexp --ignore-case --pattern hypothetical viral.1.protein.faa.gz
| seqkit head -n 3 | seqkit seq --name
gi|526245016|ref|YP_008320342.1| hypothetical protein IBBPl23_06 [Paenibacillus phage phiIBB_P
l23]
gi|526245019|ref|YP_008320345.1| hypothetical protein IBBPl23_09 [Paenibacillus phage phiIBB_P
l23]
gi|526245020|ref|YP_008320346.1| hypothetical protein IBBPl23_10 [Paenibacillus phage phiIBB_P
l23]
```

A regular expression, `^([^ ]+ )(\w+)`, was used to specify the key to be replaced, which is the first word after the sequence ID in this case. Note that we also capture the ID ( `^([^ ]+ )` ) so we can restore it in "replacement" with the capture variable `${1}` (more robust than `$1` ). And flag `-I/--key-capt-idx` (default: 1) is set to 2 because the key `(\w+)` is the second captured match. Command:

```
seqkit replace --kv-file changes.tsv --pattern "^([^ ]+ )(\w+) " --replacement "\${1}{kv} " --
key-capt-idx 2 --keep-key viral.1.protein.faa.gz > renamed.fa
```

# How do I extract paired reads from two paired-end reads files?

Let's create two unbalanced PE reads files:

```
seqkit rmdup duplicated-reads.fq.gz | seqkit replace --pattern " .+" --replacement " 1" | seqk
it sample --proportion 0.9 --rand-seed 1 --out-file read_1.fq.gz
seqkit rmdup duplicated-reads.fq.gz | seqkit replace --pattern " .+" --replacement " 2" | seqk
it sample --proportion 0.9 --rand-seed 2 --out-file read_2.fq.gz
```

Overview:

```
# number of records
seqkit stat read_1.fq.gz read_2.fq.gz

file          format  type  num_seqs  sum_len  min_len  avg_len  max_len
read_1.fq.gz  FASTQ   DNA      9,033  912,333      101      101      101
read_2.fq.gz  FASTQ   DNA      8,965  905,465      101      101      101

# sequence headers
seqkit head -n 3 read_1.fq.gz | seqkit seq --name

SRR1972739.1 1
SRR1972739.3 1
SRR1972739.4 1

seqkit head -n 3 read_2.fq.gz | seqkit seq --name

SRR1972739.1 2
SRR1972739.2 2
SRR1972739.3 2
```

Firstly, extract sequence IDs of the two file and compute the intersection:

```
seqkit seq --name --only-id read_1.fq.gz read_2.fq.gz | sort | uniq -d > id.txt

# number of IDs
wc -l id.txt

8090 id.txt
```

Then extract reads using `id.txt`:

```
seqkit grep --pattern-file id.txt read_1.fq.gz -o read_1.f.fq.gz
seqkit grep --pattern-file id.txt read_2.fq.gz -o read_2.f.fq.gz
```

Check if the IDs in two files are the same by `md5sum`:

```
seqkit seq --name --only-id read_1.f.fq.gz > read_1.f.fq.gz.id.txt
seqkit seq --name --only-id read_2.f.fq.gz > read_2.f.fq.gz.id.txt

md5sum read_*.f.fq.gz.id.txt

537c57cfdc3923bb94a3dc31a0c3b02a  read_1.f.fq.gz.id.txt
537c57cfdc3923bb94a3dc31a0c3b02a  read_2.f.fq.gz.id.txt
```

Note that this example assumes that the IDs in the two reads file have same order. If not, you can sort them after previous steps. GNU `sort` can sort large file using disk, so temporary directory is set as current directory by option `-T .`.

```
gzip -d -c read_1.f.fq.gz | seqkit fx2tab | sort -k1,1 -T . | seqkit tab2fx | gzip -c > read_1
.f.sorted.fq.gz
gzip -d -c read_2.f.fq.gz | seqkit fx2tab | sort -k1,1 -T . | seqkit tab2fx | gzip -c > read_2
.f.sorted.fq.gz
```

# How to concatenate two FASTA sequences in to one?

Related posts: Combining two fasta sequences into one

Data (not in same order):

```
cat 1.fa
>seq1
aaaaa
>seq2
ccccc
>seq3
ggggg

cat 2.fa
>seq3
TTTTT
>seq2
GGGGG
>seq1
CCCCC
```

**Step 1**. Convert FASTA to tab-delimited (3 columns, the 3rd column is blank (no quality for FASTA)) file:

```
seqkit fx2tab 1.fa > 1.fa.tsv
seqkit fx2tab 2.fa > 2.fa.tsv

cat -A 1.fa.tsv
seq1^Iaaaaa^I$
seq2^Icccccc^I$
seq3^Igggggg^I$
```

**Step 2**. Merge two table files:

```
csvtk join -H -t 1.fa.tsv 2.fa.tsv | cat -A
seq1^Iaaaaa^I^ICCCCC^I$
seq2^Icccccc^I^IGGGGG^I$
seq3^Igggggg^I^ITTTTT^I$
```

**Step 3**. Note that there are two TAB between the two sequences, so we can remove them to join the sequences

```
csvtk join -H -t 1.fa.tsv 2.fa.tsv | sed 's/\t\t//'
seq1    aaaaaCCCCC
seq2    cccccGGGGG
seq3    gggggTTTTT
```

**Step 4**. Convert tab-delimited file back to FASTA file:

```
csvtk join -H -t 1.fa.tsv 2.fa.tsv | sed 's/\t\t//' | seqkit tab2fx
>seq1
aaaaaCCCCC
>seq2
cccccGGGGG
>seq3
gggggTTTTT
```

All in one command:

```
csvtk join -H -t <(seqkit fx2tab 1.fa) <(seqkit fx2tab 2.fa) | sed 's/\t\t//' | seqkit tab2fx
```

# Using TaxonKit

## How do I list the taxonomy tree of a list of taxids?

List taxonomy tree of `9605` and `239934` in JSON format:

```
taxonkit list --show-rank --show-name --ids 9605,239934 --json

{
  "9605 [genus] Homo": {
    "9606 [species] Homo sapiens": {
      "63221 [subspecies] Homo sapiens neanderthalensis": {},
      "741158 [subspecies] Homo sapiens ssp. Denisova": {}
    }
    "1425170 [species] Homo heidelbergensis": {}
  },
  "239934 [genus] Akkermansia": {
    "239935 [species] Akkermansia muciniphila": {
      "349741 [no rank] Akkermansia muciniphila ATCC BAA-835": {}
    }
    "512293 [no rank] environmental samples": {
      "512294 [species] uncultured Akkermansia sp.": {},
      "1131822 [species] uncultured Akkermansia sp. SMG25": {},
      "1262691 [species] Akkermansia sp. CAG:344": {},
      "1263034 [species] Akkermansia muciniphila CAG:154": {}
    }
    "1131336 [species] Akkermansia sp. KLE1605": {},
    "1574264 [species] Akkermansia sp. KLE1797": {},
    "1574265 [species] Akkermansia sp. KLE1798": {},
    "1638783 [species] Akkermansia sp. UNK.MGS-1": {},
    "1679444 [species] Akkermansia glycaniphila": {},
    "1755639 [species] Akkermansia sp. MC_55": {},
    "1896967 [species] Akkermansia sp. 54_46": {}
  }
}
```

You can easily collapse and uncollapse nodes in a modern text editor:

taxon.json


```
{
  "1 [no rank] root": {
    "1 [no rank] root": {
      "10239 [superkingdom] Viruses": {
      "12884 [superkingdom] Viroids": {
      "12908 [no rank] unclassified sequences": {
      "28384 [no rank] other sequences": {
      "131567 [no rank] cellular organisms": {
        "2 [superkingdom] Bacteria": {
          "1224 [phylum] Proteobacteria": {
          "2323 [no rank] unclassified Bacteria": {
          "32066 [phylum] Fusobacteria": {
          "40117 [phylum] Nitrospirae": {
          "48479 [no rank] environmental samples": {
          "57723 [phylum] Acidobacteria": {
          "67814 [phylum] Caldiserica": {
          "68297 [phylum] Dictyoglomi": {
          "74152 [phylum] Elusimicrobia": {
          "200783 [phylum] Aquificae": {
          "200918 [phylum] Thermotogae": {
          "200930 [phylum] Deferribacteres": {
          "200938 [phylum] Chrysiogenetes": {
          "200940 [phylum] Thermodesulfobacteria": {
          "203691 [phylum] Spirochaetes": {
          "508458 [phylum] Synergistetes": {
          "1783257 [no rank] PVC group": {
          "1783270 [no rank] FCB group": {
          "1783272 [no rank] Terrabacteria group": {
          "1802340 [no rank] Nitrospinae/Tectomicrobia group": {
          "1853220 [phylum] Rhodothermaeota": {
        },
        "2157 [superkingdom] Archaea": {
        "2759 [superkingdom] Eukaryota": {
      }
    }
  }
}
```


# How do I retrieve species names and lineages using taxIDs from NCBI Taxonomy?

Related post: Question: Retrieve species name using taxaIDs of NCBI.

Example data:

```
cat taxids.txt

9606
349741
239935
11932
314101
1327037
```

Retrieve lineage:

```
cat taxids.txt | taxonkit lineage

taxonkit lineage taxids.txt
9606    cellular organisms;cellular organisms;Eukaryota;Opisthokonta;Metazoa;Eumetazoa;Bilateria;Deuterostomia;Chordata;Craniata;Vertebrata;Gnathostomata;Teleostomi;Euteleostomi;Sarcopterygii;Dipnotetrapodomorpha;Tetrapoda;Amniota;Mammalia;Theria;Eutheria;Boreoeutheria;Euarchontoglires;Primates;Haplorrhini;Simiiformes;Catarrhini;Hominoidea;Hominidae;Homininae;Homo;Homo sapiens
349741  cellular organisms;cellular organisms;Bacteria;PVC group;Verrucomicrobia;Verrucomicrobiae;Verrucomicrobiales;Akkermansiaceae;Akkermansia;Akkermansia muciniphila;Akkermansia muciniphila ATCC BAA-835
239935  cellular organisms;cellular organisms;Bacteria;PVC group;Verrucomicrobia;Verrucomicrobiae;Verrucomicrobiales;Akkermansiaceae;Akkermansia;Akkermansia muciniphila
11932   Viruses;Viruses;Retro-transcribing viruses;Retroviridae;unclassified Retroviridae;Intracisternal A-particles;Mouse Intracisternal A-particle
314101  cellular organisms;cellular organisms;Bacteria;environmental samples;uncultured murine large bowel bacterium BAC 54B
1327037 Viruses;Viruses;dsDNA viruses, no RNA stage;Caudovirales;Siphoviridae;unclassified Siphoviridae;Croceibacter phage P2559Y
```

You can also extract custom levels of rank with `taxonkit reformat`. The default format is `{k};{p};{c};{o};{f};{g};{s}`:

```
cat taxids.txt | taxonkit lineage | taxonkit reformat --fill-miss-rank | cut -f 3
```

produces:

```
Eukaryota;Chordata;Mammalia;Primates;Hominidae;Homo;Homo sapiens
Bacteria;Verrucomicrobia;Verrucomicrobiae;Verrucomicrobiales;Akkermansiaceae;Akkermansia;Akkermansia muciniphila
Bacteria;Verrucomicrobia;Verrucomicrobiae;Verrucomicrobiales;Akkermansiaceae;Akkermansia;Akkermansia muciniphila
Viruses;Retro-transcribing viruses;unclassified Viruses class;unclassified Viruses order;Retroviridae;Intracisternal A-particles;Mouse Intracisternal A-particle
Bacteria;environmental samples;unclassified Bacteria class;unclassified Bacteria order;unclassified Bacteria family;unclassified Bacteria genus;uncultured murine large bowel bacterium BAC 54B
Viruses;dsDNA viruses, no RNA stage;unclassified Viruses class;Caudovirales;Siphoviridae;unclassified Siphoviridae;Croceibacter phage P2559Y
```

# How do I retrieve all protein sequences of specific taxons from the NCBI nr database?

Dataset: prot.accession2taxid.gz

Taking viruses for example.

Step 1. Getting all taxids of viruses (taxid 10239):

```
taxonkit list --ids 10239 --indent "" > virus.taxid.txt
```

Step 2. Extacting accessions or GIs with csvtk:

- accession

```
zcat prot.accession2taxid.gz | \
    csvtk -t grep -f taxid -P virus.taxid.txt | \
    csvtk -t cut -f accession.version > virus.taxid.acc.txt
```

  - gi

```
zcat prot.accession2taxid.gz | \
    csvtk -t grep -f taxid -P virus.taxid.txt | \
    csvtk -t cut -f gi > virus.taxid.gi.txt
```

Step 3. Extracting nr sequences using accessions or GIs.

- accesion

```
blastdbcmd -db nr -entry all -outfmt "%a\t%T" | \
    csvtk -t grep -f 2 -P virus.taxid.acc.txt | \
    csvtk -t cut -f 1 | \
    blastdbcmd -db nr -entry_batch - -out nr.virus.fa
```

  - gi

```
blastdbcmd -db nr -entry all -outfmt "%g\t%T" | \
    csvtk -t grep -f 2 -P virus.taxid.gi.txt | \
    csvtk -t cut -f 1 | \
    blastdbcmd -db nr -entry_batch - -out nr.virus.fa
```

# Sequencing Instruments

Sequencing instrumentation has evolved at a dramatic pace.

In 2005 the 454 sequencing instrument jump-started the so called "next-gen" sequencing revolution. Within just 10 years, the 454 methodology and the entire platform has been discontinued and supplanted by more efficient instruments.

## Is there a document that compares sequencing instruments?

Travis Glenn's Field Guide to Next Generation DNA Sequencer (written in 2011) attempted to summarize the state of instrumentation at that time. There are yearly updates to this guide that (unlike the paper itself) are available for free.

- 2016: Updates to the NGS Field Guide

## What type of sequencing instruments are in use?

### Illumina MiniSeq, MiSeq, NextSeq, HiSeq

Illumina is the current undisputed leader in the high-throughput sequencing market. Illumina currently offers sequencers that cover the full range of data output. More details are available on the Illumina sequencers page.

- Up to 300 million reads (HiSeq 2500)
- Up to 1500 GB per run (GB = 1 billion bases)

### IonTorrent PGM, Proton

The IonTorrent targets more specialized clinical applications.

- Up to 400bp long reads
- Up to 12 GB per run

### PacBio Sequel

This is the leader in long read sequencing. More details on the PacBio sequencers page.

- Up to 12,000 bp long paired end reads
- Up to 4 GB per run

### MinION

A portable, miniaturized device that is not yet quite as robust and reliable as the other options. More details on the MinION sequencers page.

- Up to 10,000 long reads
- Up to 240 MB per run (MB = 1 million bases)

# What are some of the major obsolete sequencing instruments I should know about?

Two major platforms -- SOLiD and 454 -- have been discontinued. With the passage of time it is becoming increasingly difficult to analyze data produced by these platforms, as newer software packages do not support them. Older software sometimes fails to run at all.

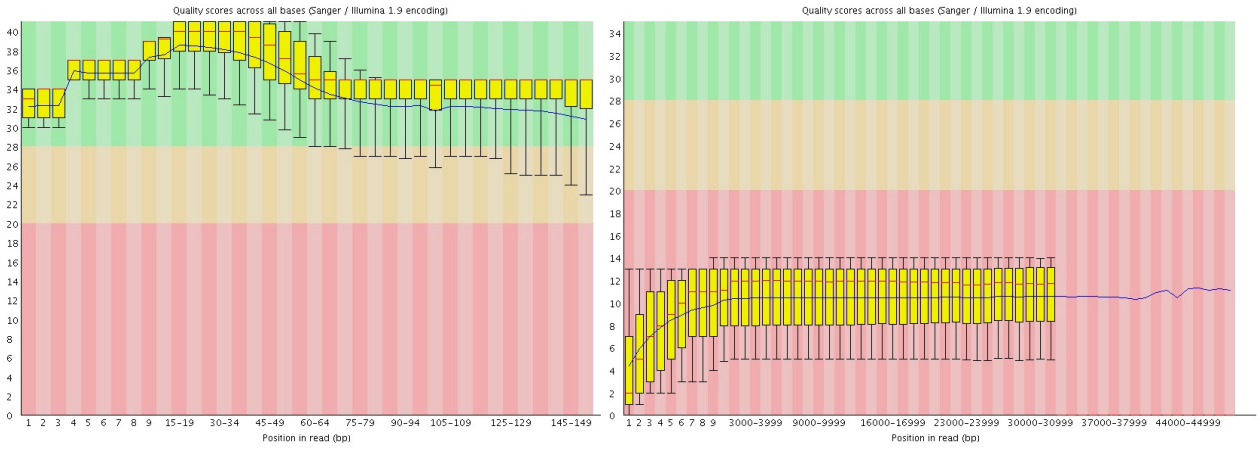
Specifically the SOLiD platform uses so called "colorspace" data, a convoluted and indirect method to measure base identities that makes data analysis even more complicated.

# How accurate are sequencing instruments?

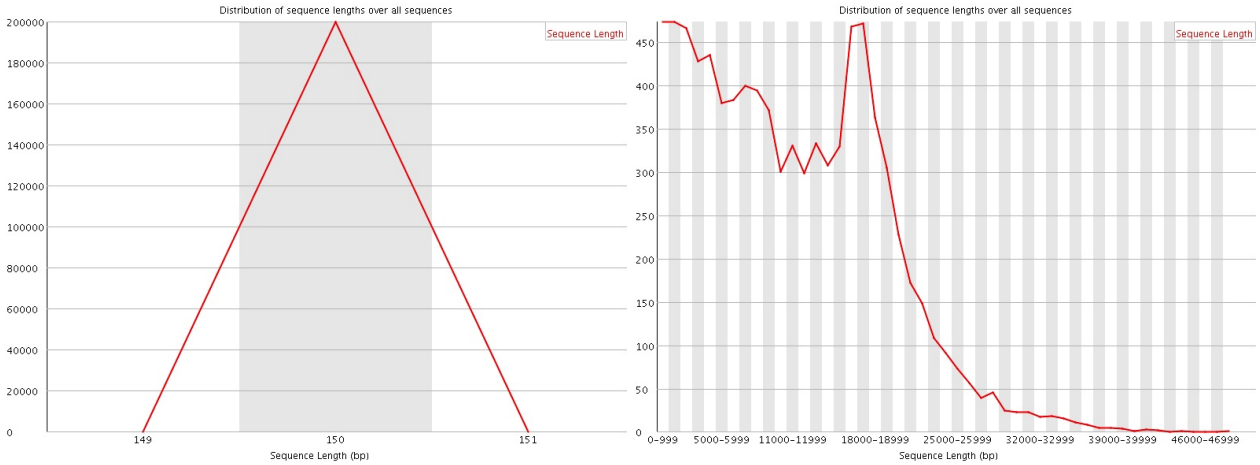
Typically the longer read instruments operate at substantially higher error rates than those producing short reads.

- Illumina: 0.1% error rates (1/1000)
- PacBio: 10% error rates(1/10)
- MinION: 20% error rates (1/5)

Errors in long reads are easier to correct and account for than errors in short reads. Compare the error rates between the Illumina (left) and PacBio (right) instruments.




Read lengths are radically different across platforms:




# How do sequencing instruments work?

A typical sequencer measures a single strand DNA fragment and from it produces a "sequencing read" of a certain length. The term "read" is very widely used to define this single measurement. The read may be shorter or even longer than the original fragment. This because the original DNA fragment (some may also call this "template") has been altered by adding sequencing adapters to it.

Below we will describe the preparation of a sequencing library for the Illumina sequencer.

Suppose that the original, double stranded DNA fragment obtained by isolating and fragmenting DNA from a cell is the following:

```
AAAATTTTGGGGCCCC
TTTTAAAACCCCGGGG
```

During library preparation, uniquely designed DNA adapters of lengths that are typically over 30 bases are added to the `5'` (right) and `3'` (left) ends of each sequence.

Below we label these as `XXXX` and `YYYY`. Hence after adding the adapters the single stranded sequences that make it onto the instrument flowcell will be of the form:

```
XXXXAAAATTTTGGGGCCCCYYYY
```

the reverse strand's directionality is reversed hence it will form:

```
XXXXGGGGCCCCAAAATTTTYYYY
```

For any given fragment, both or neither of these strands may be sequenced.

The sequencer will typically recognize the `XXXX` at the beginning and will not report it. We can illustrate the process with arrows that show the direction of sequencing:

```
---->
AAAAT
AAAATTTTGGGGCCCC
TTTTAAAACCCCGGGG
           CGGGG
           <----
```

It is important to note that the size distribution of DNA fragments varies although the range typically is within 20-50bp. What this means however is that the fragment may occasionally be shorter than the measurement. When the read length is exactly as long a given fragment, we get to the following situation:

```
--------------->
AAAATTTTGGGGCCCC
TTTTAAAACCCCGGGG
<---------------
```

As the read length grows longer than the measured DNA size, it will start running into the `3'` adapter. Neither the sequencer nor we ourselves can immediately differentiate artificial DNA from the one we wish to measure:

```
------------------>
AAAATTTTGGGGCCCCYYY
YYYTTTTAAAACCCCGGGG
<------------------
```

Hence our measurements above have artifacts at their ends. Typically this type of situation is best represented as shown below to highlight the useful region that gets sequenced.

```
   ------------------>
   AAAATTTTGGGGCCCCYYY
YYYTTTTAAAACCCCGGGG
<------------------
```

It is a situation of a so called "read-through", where the sequencing is longer than the fragment. If the read-through is sufficiently long, say at least 5 bases, we may attempt to remove these computationally after sequencing with tools that recognize the start of the adapter sequence.

# Can reads be in different orientations?

Depending on sequencing protocols and instrumentations, data may end up with read orientations such as:

```
---->
           ---->
```

or even:

```
<----
          ---->
```

The vast majority of software tools cannot handle data with these orientations and may treat them in various incorrect ways.

It is relatively simple to convert data orientation. A simple post processing operation can be used to reverse complement the reads into the most commonly used `----> <----` orientation.

# What is paired-end sequencing?

Paired-end (PE) sequencing is a method to sequence both ends of a fragment and to make the pairing information available in the data.

DNA fragments are typically longer than the measured read lengths. For many applications, it is greatly advantageous to be able to measure (if not the entire fragment) at least both ends of it. Many biological processes initiate at certain locations and knowing exactly where the fragment starts and ends can provide critically important information.

For that reason, some instruments offer the option of running the instrument differently to achieve two measurements from a single strand DNA.

In the Illumina sequencing protocol for example, the first round of reads are called the "single end" (SE) or "first" reads.

```
---->
AAAATTTTGGGGCCCC
```

If the paired end protocol is employed, after producing the "first" reads the *same* sequence is flipped over within the instrument, it is reverse complemented and a second measurement is taken to produce another set of reads. These are called the "second" reads.

```
---->
GGGGCCCCAAAATTTT
```

The net effect is that we obtain two measurements from one single stranded fragment:

```
---->
AAAATTTTGGGGCCCC
TTTTAAAACCCCGGGG
          <----
```

The two reads are typically stored in separate FASTQ files and are synchronized by name and order. Each read in file 1 has a corresponding entry in file 2.

Important: It is our responsibility to maintain this order and perform all operations on these files in such a manner that they are kept in sync and each contains the reads in the exact same order.

# What are advantages and disadvantages of paired-end sequencing?

Paired-end sequencing is more expensive than single end sequencing but not radically so (perhaps 20% more).

The ability to identify the ends of a DNA fragment typically carries a lot of importance, as biological processes always have a directionality. In addition, two measurements of the same fragment offers the user a better chance to properly identify the location and composition of the original DNA fragment. We recommend that genomic variation and genome assembly analyses use paired end sequencing as much as possible.

On the downside, paired end sequencing measures the same fragment twice, hence at the same genomic coverage, it will use half as many unique fragments. For analysis methods that use sequencing for quantification such as RNA-Seq and ChIP-Seq, this may be a disadvantage. Typically single end sequencing is used, as it produces more uniform and higher resolution coverage.

# What is mate-pair sequencing?

Mate-pair sequencing typically has the same goals as the paired-end approach -- it attempts to measure two ends of a fragment. The implementation, however, is quite different and, unlike the paired-end sequencing, requires a better understanding of each particular instrument.

Notably, mate-pair DNA fragments are much much longer than PE methods and the read orientations are usually different, for example:

```
---->
AAAATTTTGGGGCCCC
           ---->
```

Paired-end data is mainstream and most tools support it directly. Mate-paired data is only supported by specialized software.

# Illumina sequencers

## What types of sequencers does Illumina manufacture?

MiniSeq - the smallest bench-top sequencer Illumina sells (as of 2016). It only makes sense for those labs who have limited budgets and want to get started with high-throughput sequencing.

- Run time 4-24h (depending on the type of run)
- Up to 150bp long paired-end reads
- Up to 25 million reads per run
- Up to 8 GB per run (GB = 1 billion bases)

MiSeq - perhaps the most popular instrument that started the trend for truly "bench-top" sequencers. MiSeq is small enough to fit on a regular lab bench and only requires a network connection (if you wish to use BaseSpace or off-load data from the instrument). Otherwise it is completely self-contained.

- Run time 4-55h
- Up to 300bp long paired-end reads
- Up to 25 million reads per run
- Up to 15 GB per run (GB = 1 billion bases)

MiSeqDX is a special variant of MiSeq that is the first FDA-cleared high-throughput sequencer for *in vitro* diagnostic (IVD) testing.

- Currently there are two Cystic Fibrosis assays approved for use with MiSeqDX.
  - A panel of 139 clinically relevant variants for CFTR
  - A test for comprehensive view of the CFTR gene

NextSeq 500/550 bench-top sequencer

- Run time 12-30h
- Up to 150bp paired-end reads
- Up to 400 million reads per run
- Up to 120 GB per run (GB = 1 billion bases)

HiSeq 2500/3000/4000 sequencers are the workhorses of sequencing. HiSeq 2500 is meant for large-scale genomics and HiSeq 3000/4000 are meant for production-scale (think core facilities).

- Run time 1-3.5d (HiSeq 3000/4000)
- Run time 7h-6d (HiSeq 2500)
- Up to 150bp paired end reads (HiSeq 3000/4000)
- Up to 250bp paired end reads (HiSeq 2500 rapid mode)
- Up to 5 billion reads (HiSeq 3000/4000)
- Up to 300 million reads (HiSeq 2500)
- Up to 1500 GB per run (GB = 1 billion bases)

HiSeq X Five/X Ten sequencers: As the name suggests, there are 5/10 sequencers (sold as a package) to enable population-level genome sequencing. These were originally only certified for human DNA sequencing.

- Run time 3d
- Up to 150bp paired end reads
- Up to 6 billion reads per run
- Up to 1800 GB per run (GB = 1 billion bases)

NovaSeq 5000 : Meant for counting applications

- New flow cells (S1, S2)
- Two-color chemistry like NextSeq 500
- 50, 100 and 150bp paired end
- Up to 2 terabases data (1.6 billion reads) in 2.5 d
- Two flowcells per sequencer, can be run independently
- Same flowcells can be used for single-end or paired-end runs
- One library pool per flowcell
- Four lanes per flowcell, visually distinct but optically identical
- On-board cluster generation

NovaSeq 6000 : High sequence coverage applications

- New flow cells (S1, S2, S3 and S4)
- S3 and S4 flocells only usable in NovaSeq 6000
- Two-color chemistry like NextSeq 500
- 50, 100 and 150bp paired end
- Up to 6 terabases data (10 billion reads) in 2 d
- Two flowcells per sequencer, can be run independently
- Same flowcells can be used for single-end or paired-end runs
- One library pool per flowcell
- Four lanes per flowcell, visually distinct but optically identical
- On-board cluster generation
- 132 exomes/transctipromes per run

# PacBio sequencers

Pacific Biosciences is the leader in Single Molecule Real Time (SMRT) sequencing technology. Their *Sequel* platform is a high-throughput long read sequencer.

## What resources are available for PacBio data analysis?

Since PacBio offers longer reads than Illumina, a new set of tools and pipelines are required for data analysis.

Resources:

- https://github.com/PacificBiosciences/
- https://github.com/PacificBiosciences/Bioinformatics-Training/wiki
- http://www.pacb.com/products-and-services/analytical-software/devnet/

## What is an SMRT cell?

A SMRT cell consists of an array of measurement devices called Zero-Mode Waveguides (ZMWs). At the time of writing, *Sequel* system has one million ZMWs per SMRT cell.

## What is a Zero Mode Waveguide (ZMW)?

A ZMW is a small hole with a unique structure in a SMRT cell that enables real time sequencing observation. It is a nanophotonic device with a diameter too small to permit propagation of light in the wavelength range used for detection.

## What is a subread?

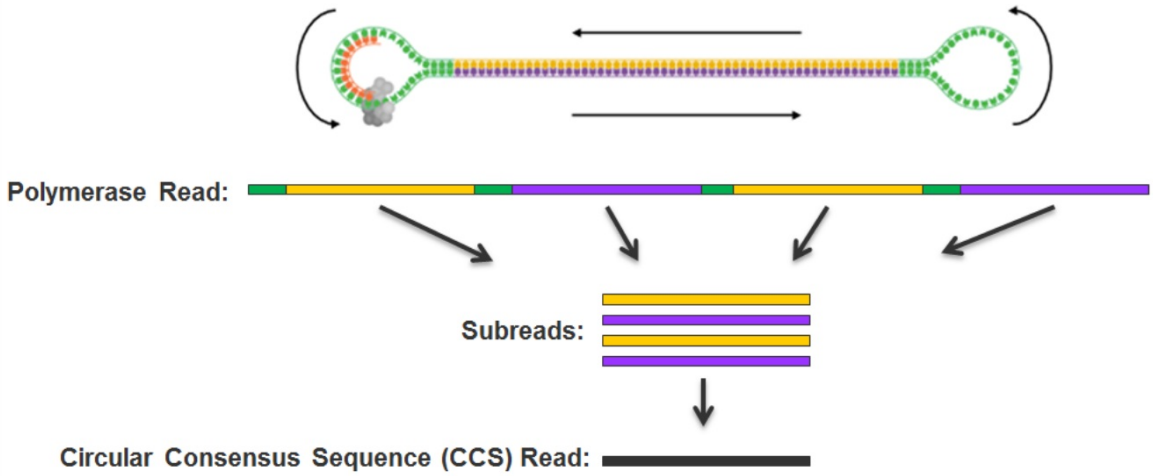
A subread is an individual measurement (read) that corresponds to the DNA template between the adapter sequences. The PacBio methodology allows for the same template to be measured multiple times, hence multiple measurements called subreads can be produced by a single ZMW waveguide.

## How many subreads are produced from a template?

Divide the length of the DNA fragment (template) by the length of a read. For example, if the templates are 1KB and the sequencer produces 10KB reads, 10 subreads can ideally be produced from each template. This is a highly simplified case. Realistically we tend to observe: (TODO)

# What is a Circular Consensus Sequence (CCS) read?

A CCS read is a consensus sequence obtained by the alignment of all subreads. When the template is shorter than the read length the polymerase will loop back, sequencing the template again. A sequence from a single pass is called a subread and multiple subreads can be used to obtain a single consensus sequence with much higher accuracy than that of the individual subreads. More than 2 full pass subreads are required to generate a CCS read.




# What is the output of a PacBio run?

A PacBio run folder contains unaligned bam files, adapter fasta file and metadata in xml format.

The primary output of a PacBio run is an unaligned BAM file called `subreads.bam`. The data in this file is to be used for downstream analysis. All rejected subreads, excised barcodes, and adapters are combined in a file called `scraps.bam`.

# What other information is encoded in the BAM file?

PacBio-produced BAM files also encode instrument-specific information in the header and alignment sections.

# What SAM headers does the instrument set?

- `RG` (read group) - 8 character string read group identifier for PacBio data.
- `PL` (platform) - contains "PacBio".
- `PM` (platform model) - contains PacBio instrument series eg: SEQUEL.
- `PU` (platform unit) - contains PacBio movie name.
- `DS` (description) - contains some semantic information about the reads in the group, encoded as a semicolon-delimited list of “Key=Value” strings.

# How is the BAM file formatted?

The query template name convention for a subread is:

```
{movieName}/{zmw-holeNumber}/{qStart}_{qEnd}
```

The query template name convention for a ccs read is:

```
{movieName}/{holeNumber}/ccs
```

# What SAM tags are added to the reads?

- **qs** - 0-based start of the query in the ZMW read (absent in CCS).
- **qe** - 0-based end of the query in the ZMW read (absent in CCS).
- **zm** - ZMW hole number.
- **np** - NumPasses (1 for subreads, for CCS, it encodes the number of complete passes of the insert).
- **rq** - expected accuracy of the read.
- **sn** - average signal-to-noise ratio of A, C, G, and T (in that order) over the HQRegion

# How do I get subreads from a single ZMW?

We can use the `zm` tag in the bam file to extract all subreads from a single zmw. This can be done with the *bamtools filter* command.

```
bamtools filter -in subreads.bam -out zmw.bam -tag 'zm:10027159'
```

The command will read the entire BAM file to extract the matching entries. This approach is reasonable only if we need to do this for one or two ZMWs. To split the BAM file into parts just by ZMWs, a custom program needs to be written.

# Why would I want to split a BAM file by ZMWs?

The consensus caller (see below) operates on data for a single ZMW at a time. You can massively speed up the process by running the consensus caller in parallel on data that corresponds to each ZMW separately.

# How do I get the consensus sequence from the subreads?

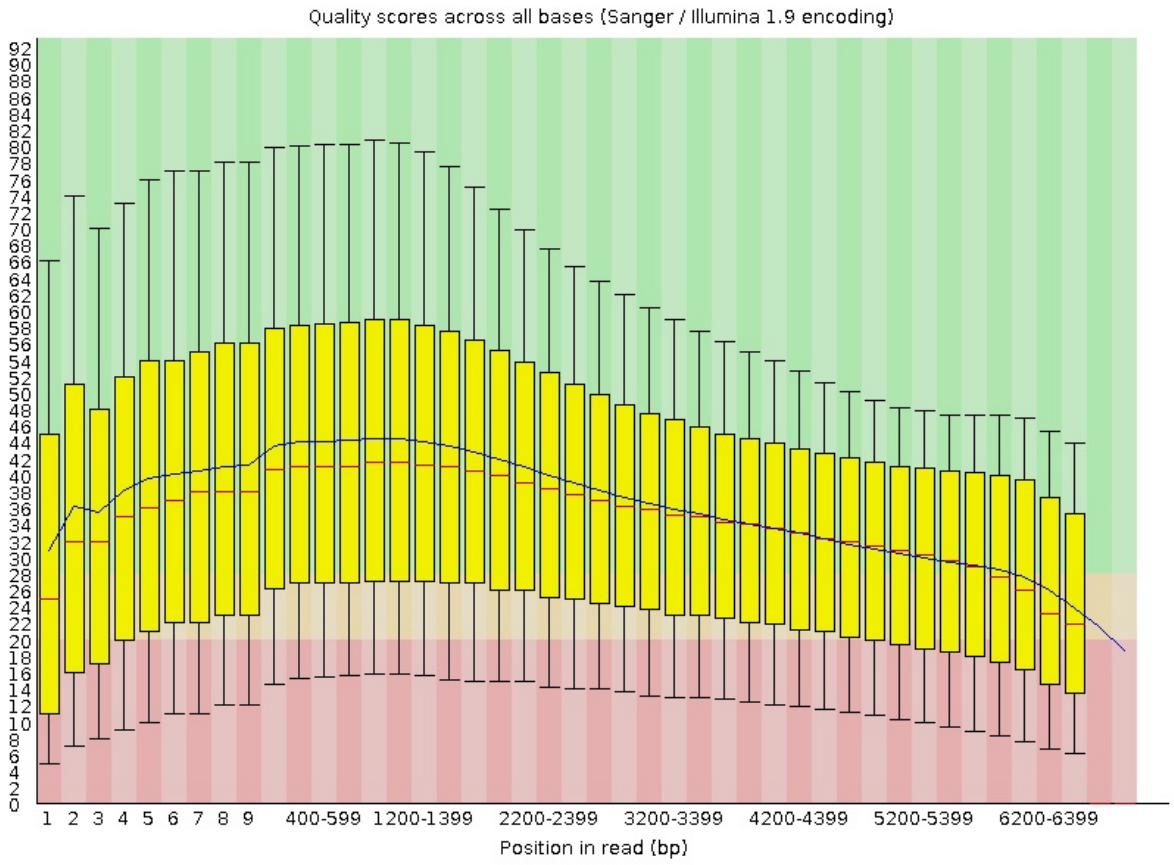
The `ccs` command from the Unanimity C++ library can be used to generate CCS reads. This command requires a minimum of 3 full pass subreads to a generate a CCS read.

```
# ccs [options] INPUT OUTPUT
ccs subreads.bam ccs.bam
```

# What is the per-base quality of PacBio reads?

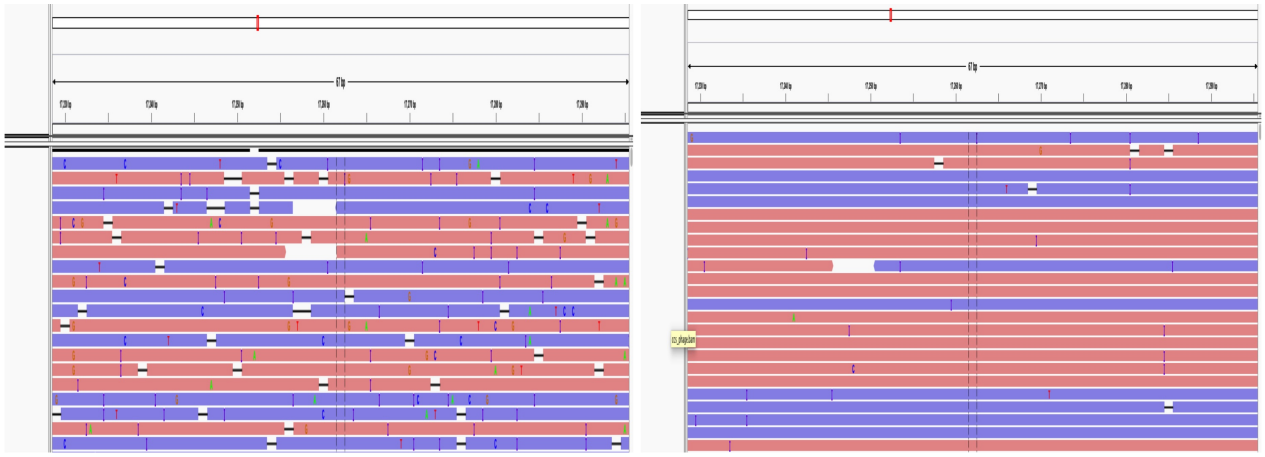
Whereas the quality of the original reads is low, the quality of consensus corrected reads improves dramatically.

The FastQC report plot below shows the per-base quality of some CCS reads.




# How good are the consensus corrected reads?

On the lower left is the alignment of subreads to the genome and on the right is the alignment of CCS reads.



# Minion sequencers

## What is MinION?

The MinION is a pocket-sized long read sequencing instrument from Oxford Nanopore technologies powered by a USB cable. It uses nanopore sequencing technology. Sequencing occurs by measuring changes in the ionic current as DNA translocates through protein nanopores in an insulated membrane on the flow cell. Below each nanopore channel is a very sensitive electrical sensor. A characteristic electric signal is generated by the flow of ions when no molecule is in the pore. When a strand of DNA or RNA passes through the pore, partially blocking the flow of ions, the current pattern will be disturbed in a manner characteristic to the molecule and dependent on the nucleotide sequence. The technology has essentially no size limitations -- if you manage to keep a molecule intact during the library prep, reads up to hundreds of kilobases can be obtained. A standard library prep will routinely result in the majority of reads being greater than 10kb. Over the past few years, Nanopore sequencing technology has seen a rapid development with drastic improvements in throughput and accuracy.

## What is MinKNOW?

MinKNOW is the software that controls the MinION sequencer. The scripts in MinKNOW software carry out several tasks including device configuration, platform QC and calibration, data acquisition etc. During the platform QC, the quality of the pores for sequencing is assessed and the most optimal pores will be selected for starting the run. A sequencing script starts the sequencing process. As the run progresses, it produces one FAST5 file per DNA molecule and stores it in the specified run directory on the SSD disk of your sequencing computer. It is possible to modify the (python) sequencing scripts to tweak the run and obtain higher yields, but this is something that is best left to those with considerable experience with the technology.

## What is the Metrichor Agent?

The Metrichor Agent manages the connection to the base-calling service in the cloud (hosted by Metrichor). When Metrichor Agent is started on the sequencing laptop, it moves the pre-base-called FAST5 files to the specified uploads folder and then transfers a copy to the base-calling service. The FAST5 file with original information and base calls is then returned and stored in the specified download folder on the sequencing computer. In addition to cloud base-calling, local alternatives are available. It's expected that cloud base-calling will be discontinued in the near future. Other applications are available in the cloud -- such as the What's In My Pot (WIMP) workflow, in which a metagenomic sample is classified based on the obtained sequence to identify species contained therein.

## What is the output of a MinION run?

Each read sequenced on the MinION will generate a FAST5 file, which is a variant of the HDF5 file format. This format is essentially a container for storing a variety of data types (eg. integers, floats, strings, arrays) in a single file. The format has a hierarchical structure with 'groups' for organizing data objects and 'datasets' which contain a multidimensional array of data elements. These files contain voltage and time information, metadata identifying the pore and flow cell used for sequencing, among other things. This data allows for base-calling of the reads, either in the cloud or on your local system. In addition, when new base-calling software is available, the analysis can be repeated. Third-party base-callers for nucleotide modifications such as methyl-cytosine are also available.

## What do 'pass' or 'fail' subfolders contain?

If you activate the quality filter at the start of a run, the base-called data will get stored into 'pass' or 'fail' subfolders in the specified download folder. The pass/fail classification system is intended to help the user distinguish higher and lower quality data. Lower quality data isn't necessarily useless -- depending on your application this data may still be valuable. The classification criteria may change between different versions of Metrichor.

## What is the different between 1D and 2D reads?

In general two library preparations are possible, termed 1D and 2D sequencing. The highest accuracy is obtained by ligating a hairpin adapter on one side of the molecule, hence sequencing the double-stranded DNA in both directions (2D). The first strand is called the template read, the second strand the complement. Since a consensus read can be created based on both strands, the accuracy of a 2D read is higher. For a 1D library prep, no hairpins are ligated and sequencing is performed for each strand separately, effectively increasing the throughput but sacrificing some accuracy. Since May 2016, the accuracy of 1D reads has reached acceptable levels.

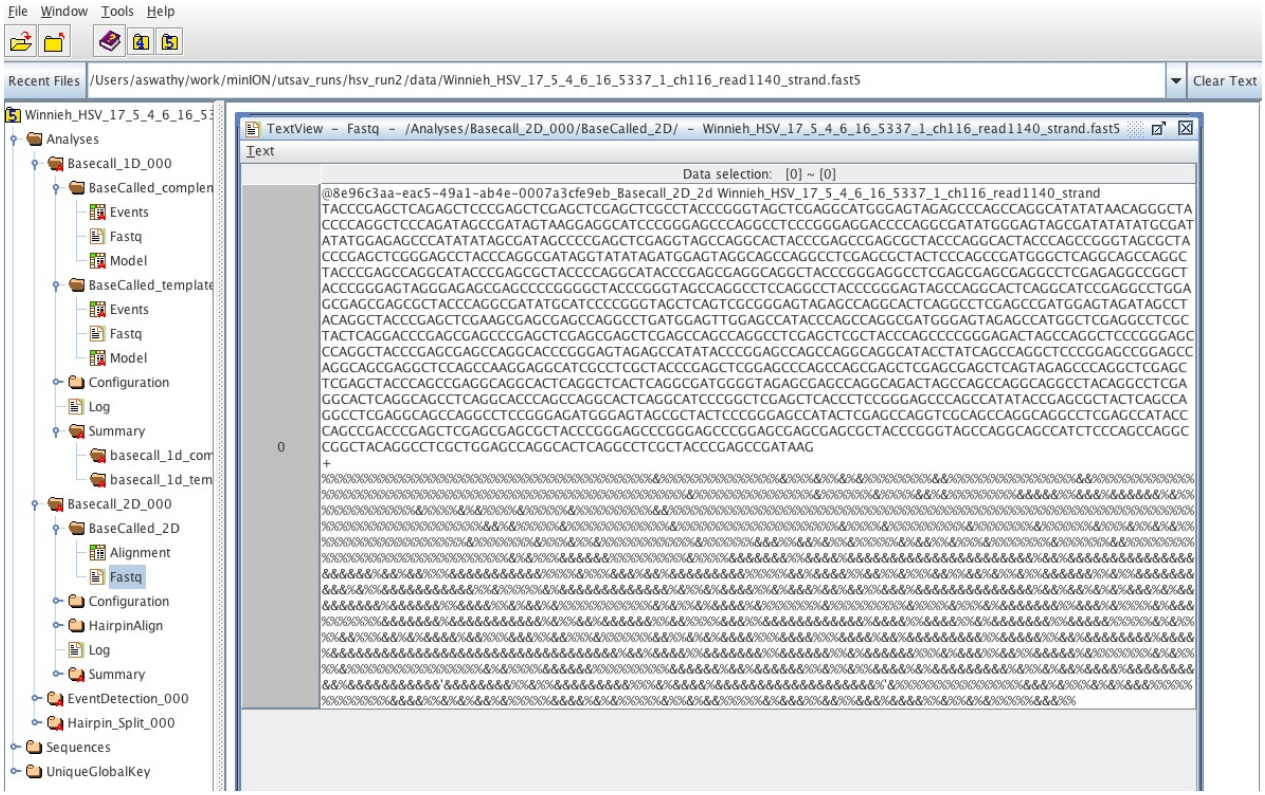
## How do I extract data from a FAST5 file?

Different commands and tools are available to extract data from FAST5 file.

- HDFView
- poretools
- poRe

## What is HDFView?

HDFView provides a Graphical User Interface (GUI) to browse the contents of a FAST5 file. Shown below is an HDFView display of a 2D read sequence.




# How can I extract data from a FAST5 file using poretools?

Poretools is a toolkit for extracting data from fast5 data, written by Nick Loman and Aaron Quinlan. publication

Type the command with no arguments to get the basic usage.

```
poretools

usage: poretools [-h] [-v]
{combine,fastq,fasta,stats,hist,events,readstats,tabular,nucdist,quald
ist,winner,squiggle,times,yield_plot,occupancy}
                 ...
poretools: error: too few arguments
```

To extract FASTQ sequences from a set of FAST5 files in a directory 'fast5-data'

```
poretools fastq fast5-data/
```

Extract all 2D reads from FAST5 files:

```
poretools fastq --type 2D fast5-data/
```

To get the length, name, sequence, and quality of the sequence in one or a set of FAST5 files, we can use the 'tabular' command.

```
poretools tabular data.fast5
```

# Sequencing Data Preparation

## From sample submission to receiving sequence data

Author: Hemant Kelkar

Note: Discussion below draws upon personal observations over the past few years. Your experience may vary due to practices prevalent at the sequencing facility you use.

Getting samples ready for high-throughput sequencing is a result of a days/weeks/months long effort on your part but suffice it to say that a significant amount of money and labor has already been expended on that process.

## A recap of high-throughput sequencing

High-throughput sequencing data broadly falls into two categories:

- Short reads (300 bp or smaller).
- Long reads (>350 bp to several kilobases).

What goes into a high-throughput sequencing experiment may be familiar to many wet-lab scientists.

Here is a quick summary:

- Sequencers sequence sheared and purified DNA fragments.
- Addition of technology-specific adapter sequences (oligos) create "sequencing ready libraries".
- Controlled amplification (PCR) is required with most prep methods (PCR-free methods are available for some applications).
- Sequencing is done by DNA synthesis (with the exception of Oxford Nanopore and some optical mapping technologies).
- Detection of synthesis is done using light (Illumina, PacBio) or non-light (Oxford Nanopore, Ion) methods.
- Post-processing of "raw" data is required to generate actual sequence.

Note: Direct RNA sequencing has been demonstrated with some technologies, but no commercial applications are currently available.

Illumina sequencing is by far the most common high-throughput sequencing technology currently in use.

- It has been commercially available for almost a decade.
- Sequencers and sequencing kits have evolved to cover the entire spectrum of data yields.
- Error profiles are reasonably well understood.
- Sequencing is relatively cost-effective.

Ion technology is used for specific applications and when fast data turnaround is a high priority.

Pacific Biosciences implements a mature long read technology that has been on the market for some time. Oxford Nanopore (another long read method) is gaining interest in academic labs at a rapid pace.

# Where can I find illustrated summaries of sequencing technologies?

The following videos illustrate currently popular sequencing technologies:

- Illumina, Intro to Sequencing by Synthesis (HiSeq 2500)
- Illumina, Patterned Flow Cell Technology (HiSeq 3000,4000,X)
- Ion Torrent next-gen sequencing technology
- Pacific Biosciences (SMRT Technology) and Introduction to SMRT Sequencing
- Oxford Nanopore Sequencing Technologies

# How do I select proper sample identifiers?

Do NOT use spaces or the following characters ( `? ( ) [ ] / \ = + < > : ; " ' , * ^ | & .` ) in sample identifiers. These characters can cause problems with Illumina (and/or other technology) data pre-processing software and should be avoided.

DO use sample names or identifiers that make immediate sense to you. It is best to avoid personal identifiers of any kind to prevent regulatory issues. If you use "Sample1/Sample2" for multiple submissions the data files that you receive will have those exact names. While facilities will keep the data segregated you may have a hard time keeping track of 'what is what' down the road.

Sequencing cores may track samples using internal IDs (as part of a LIMS or otherwise), but the sample name you submit will be carried forward through the process. In the case of Illumina technology, this name may get incorporated into the sequence file itself.

# How to choose right sequencing method?

The number of cycles of sequencing and read type (single-end/paired-end) determines the kind of data you will receive. The type of library you made is also important. All of these choices have a direct impact on the cost of sequencing. In general, longer and/or paired-end reads (Illumina) and very long reads (PacBio) are comparatively more expensive.

It is not possible to cover the entire breadth of sequencing choices in this section.

Here is a representative example. If you are only interested in read counts and differential gene expression (e.g. RNAseq) then single-end reads of a reasonable size may be enough. You will not get spatial information about fragments unless you choose to do paired-end sequencing. If you were interested in identifying full-length alternately spliced transcripts then Pacific Biosciences IsoSeq would be the method of choice (assuming there are no budget limitations).

# Why is a submission of sample metadata critical?

Sample metadata (type of sample, specific sample characteristics, experiment type) that you include with your sample submission may play a crucial role in the success of sequencing.

Illumina sequencing technology assumes a reasonably uniform distribution of nucleotides (A/C/T/G, ~25% each) in your samples. If your sample is known to deviate from this assumption (e.g. amplicons, metagenomics, high GC/AT samples) then make a note of this during sample submission. A "spike-in" may need to be added to such samples (generally phiX is used but any "normal" genomic DNA may be used) to address "low nucleotide diversity" (for sample examples noted above).

Ion/Oxford Nanopore technologies are not very robust when homopolymers (e.g. AAAAA) of significant length are present in samples. They would not be appropriate for such samples.

# Why is sample/library QC essential?

Qubit and Bioanalyzer analysis are preferred options (qPCR may also be used) for analyzing the quality of starting material and finished libraries. Not all labs have access to these technologies and hence they may use alternate means for QC (e.g. nanodrop). The amount of material you think you are submitting invariably turns out to be less (sometimes 50% or more), which can directly affect data yields.

Final sequencing data quality depends on quality of samples (garbage in, garbage out). It may be worth asking the sequencing facility to do QC via their own methods -- some facilities may do this as a part of their standard sequencing protocol. There may be a charge for this QC, but it sure beats having sequencing fail because of samples with bad quality.

Making good sequencing libraries is an art. If you intend to keep making libraries for many samples then it would be cost effective to acquire this expertise in house. Otherwise, it is best to get a professional (at a core or elsewhere) to take care of this crucial step.

# What can large-scale projects do to ensure sample data integrity?

If you are submitting a large number of samples (e.g. samples in multi-well plates), then it is useful to think about having an independent means of identifying samples from sequence data by correlation with an independent parameter (e.g. genotype). While sequencing facilities do take precautions in handling plates/large batches of samples, a human is involved in some steps and errors will sometimes occur.

# What happens during a sequencing run?

Once your sample libraries clear these initial checks they are ready to run on the sequencer. As noted elsewhere the machine run times may vary from a few hours to a week depending on the type of run and model of the sequencer.

Illumina sequencers use an elegant combination of precision engineered components, motors, manifolds, lasers, optics, and on-board proprietary software to capture sequence data. It is possible to monitor the sequencing run both locally and remotely (with the right set up) in real time.

Illumina flowcells are imaged and processed in real time on the workstation (or internal computer) attached to the sequencer. Per-cycle base calls generated are saved in an elaborate folder structure that contains data for each flowcell. The naming convention for the folders is in this format: `YYDDMM_SequencerName_RunNo_FlowcellBarcode` (FlowcellBarcode is tagged in front with `A` or `B` in case of sequencers that can run two flowcells at one time).

# What does an original Illumina sequence data folder contain?

Depending on the run type (length, flowcell, sequencer type) a raw data folder can contain between 25 to ~550 GB of data. There can be hundreds of thousands files, in multiple, nested folders, that follow an identical layout for every flowcell. This folder structure allows for scripting and/or workflow based processing of data. Individual file sizes range from a few bytes to tens of megabytes. Some sequencing instruments (e.g. MiSeq) may store some imaging data that are useful for diagnostic purposes in case of run failure.

# How does one get "finished" data from "raw" sequence data?

Smaller sequencing cores may use the onboard data analysis capabilities of sequencers (MiniSeq/MiSeq/NextSeq) and/or BaseSpace, Illumina's cloud computing solution for management of sequence data. If an institution can't use third-party cloud computing solutions, a locally installed variant of BaseSpace called "BaseSpace Onsite" is also available as an option. BaseSpace uses "apps" to provide added analytics functionality beyond processing of raw sequence data.

Larger cores generally use `bcl2fastq`, which is Illumina's software for converting the per-cycle base calls (stored in BCL files) into fastq format. `bcl2fastq` is currently available in two versions (v. 1.8.4 for older sequencers, v. 2.18.x for newer sequencers), but is backwards compatible with all current Illumina sequencers.

`bcl2fastq` does per-cycle BCL file to per-read FASTQ format conversion and simultaneous demultiplexing of samples. `bcl2fastq` can optionally trim adapters and remove Unique Molecular Identifiers (UMI) from reads.

`bcl2fastq` conversion requires access to a cluster (or a high-performance multicore computer). These conversion runs may take between 30 min to a few hours (depending on the type of run). The end result of a `bcl2fastq` run is a folder of sequence files and several report/results files/folders. These are used by sequencing facilities to do internal quality control of sequence data. Once a sample passes these internal QC measures, it is generally ready for release to the submitter.

# What should I expect to get from a sequencing provider?

The main deliverable for high throughput sequencing is a gzip compressed FASTQ formatted sequence data file(s) for each sample. You will get as many files as the number of samples you had submitted.

File name formats may vary depending on how the local pipeline is run. The file may be named something like:

```
SampleID_ACGGCTGAC_L001_R1_001.fastq.gz
```

If you requested paired-end sequencing, then you should expect to get two files per sample. The order of the reads within the files indicate the read pairings.

```
SampleID_ACGGCTGAC_L001_R1_001.fastq.gz
SampleID_ACGGCTGAC_L001_R2_001.fastq.gz
```

Note: In rare instance, you may receive R1/R2 reads in an interleaved format in a single file (R1_r1, R2_r1, R1_r2, R2_r2 etc).

In case your samples are not multiplexed "Barcode/Tag" (ACGGCTGAC above) in the filename may be replaced by the word "NoIndex". Newer versions of `bcl2fastq` do not put the "Barcode" in the file name by default, so it may not always be present.

`L001` indicates the lane your sample ran in. `001` indicates a single data file per sample. Illumina software allows sequence data files to be split into defined chunks (2 Million reads per file is default). If that is the case, you will get a series of files (with the same SampleID) where `001` in the file name will be incremented for each file chunk for that particular sample ( `001` through `nnn` ).

# How is the sequencing data delivered?

This will largely depend on your sequencing provider. Sequencing data files (even when compressed) can be fairly large (tens of GB per sample). You will need adequate internet bandwidth to download the files directly. Ideally, your provider will create MD5 hashes for your files so that you can verify the file integrity after download.

Commonly used data delivery options are :

- Delivery via a storage partition mounted on a local computer cluster (if you are using an internal sequencing facility).
- Via the Internet (web links)
- Illumina BaseSpace (if it is in use at the sequencing facility). You will need a free BaseSpace account to access data.
- Amazon/Google/Microsoft/Other cloud storage solution. Preferred, if data needs to be shared with many people.
- Shipping a copy of the data on external hard drives may still be an (or only) option for some who are security conscious, don't want their data via the internet, or do not have adequate bandwidth

# What should I do with the raw data?

You should ensure there is a backup copy for internal needs. External drives are fine for small lab groups (keep two copies on separate drives, for added security). Consider submitting the raw data to NCBI SRA as soon as is practical. You can request an embargo period (before the data becomes publicly available) to allow you to complete the analysis and write a paper.

Many facilities keep internal backups of raw sequence and processed data files for varying periods of time (ask about your facilities' data policy). Bear in mind that retrieval of data may incur an additional charge, beyond a certain period of time.

# Where can I download Illumina software?

`bcl2fastq` (and many other software packages) can be obtained from Illumina's portal (iCom) by creating a free account. Most Illumina software is available at no cost. Software obtained from iCom is NOT open source. The only software hosted on iCom is software supported by Illumina technical support.

Illumina maintains a GitHub repository (Illumina Repo) specifically for open source software. Open source software on GitHub repo is NOT supported by Illumina technical support.

# Sequencing data and coverage

## How much data do I need?

This is one of the first questions that everyone asks. The answers to this question are always a bit complicated and depend on the goals of the experiment.

The amount of data is typically expressed as "coverage" and represents how many measurements (on average) will be taken from each base of the genome.

## How is genome coverage computed?

The simple definition of coverage is

```
C = number of sequenced bases / total genome size
```

The only slightly confounding factor is that the number of sequenced bases come in chunks of read lengths rather than one base at a time.

Typically the coverage is expressed with the `x` symbol. For example, `10x` indicates that on average, each base of the genome will be covered by `10` measurements.

For a sequencing instrument that produces variable reads lengths we have:

```
C = sum(Li) / G
```

Where the sum goes over `N` the read count:

```
Li = length of read i
N = number of reads
G = size of genome
```

For an instrument with fixed read length `L`, the formula simplifies to:

```
C = N * L / G
```

### Example

What is the coverage of a 20KB virus if we measured 1000 reads of 50bp each?

```
C = 50 * 1000 / 20000 = 2.5
```

The coverage is `2.5x` that is on average each base is sequenced `2.5` times.

If we needed to raise that coverage to `10x` we could do the following:

1. Increase the read lengths.
2. Produce more reads.
3. A combination of both of the above.

We can invert the formula and compute it exactly but usually it is really easy to see by eye what needs to be done. To get to `10x` we need to increase the coverage by a factor of 4. We could choose to:

1. Increase `L=200` and keep `N=1000`
2. Keep `L=50` and increase `N=4000`

Usually there are various real life constrains on how the instrument operates and a combination ends up being chosen such as `L=100` and `N=2000`

# How many bases are missing?

The coverage is an average metric and has variability in it. At a given coverage some bases will likely not get covered. There are formulas to compute it based on a theoretical model called the Lander/Waterman model (random fragmentation).

At a coverage `C` the probability of a base not being sequenced is:

```
P = exp(-C)
```

## Example

How many bases will not be sequenced when we cover a `20KB` virus at `10x` coverage.

```
P = exp(-10)
```

From command line:

```
python -c "import math; print math.exp(-10)"
# Prints: 4.53999297625e-05
```

To figure out how many bases will be missed we need to multiply this result by the genome length:

```
python -c "import math; print math.exp(-10) * 20000"
# Prints: 0.90799859525
```

So at 10x coverage we will miss about one base. When we cover the human genome at the same coverage:

```
python -c "import math; print math.exp(-10) * 3E9"
136199.789287
```

We will miss `136,199` bases. Note, though, that these gaps will be theoretically dispersed across the entire genome.

# Do theoretical coverages describe reality?

Not really. Think of theoretical coverages as the ideal, absolute best case outcomes. Reality is typically a lot more complicated. DNA does not fragment randomly and may exhibit various non-random patterns. We don't sequence the genome directly; instead, a series of complex material extraction and preparation protocols need to be performed that will show preferences toward some parts of the DNA. Notably:

1. Increase your coverages well over the theoretical limits.
2. Sometime some parts of the genome do not show up in the data. These are called *hard to sequence* regions, but it is not always clear why they are hard to sequence.
3. What part of the genome is uniquely coverable with a given read size? Repeat-rich regions of the genome require longer reads to resolve.
4. Is a given read from one region or is it potentially sourced from multiple regions? We may resolve the source with a longer read.

Scientists often use terms such as: “accessible”, “mappable”, “effective” genome to indicate that only some parts of the genome can be easily studied.

# Visualizing sequencing data quality

The first step after receiving sequencing data is to evaluate its quality.

## What gets visualized?

The FASTQ format contains sequenced bases and a quality value associated with each. For example, consider a record that looks like this:

```
@id
ATGC
+
05:I
```

The `05:I`, when decoded, corresponds to quality measures of `15 20 25 40`. Data quality visualizations generate various plots out of these numbers, for example, distribution of values by each base or by average for each sequence.

Also, the composition of sequences may also be used to identify characteristics that could indicate errors.

## How can we visualize FASTQ quality?

The undisputed champion of quality control visualization is a tool named FastQC developed by Babraham Institute, an independent, charitable life sciences institute involved in biomedical research.

Even though it is a de-facto standard of visualization, its results are not always the simplest to interpret. On the positive side, the tool is easy to run (requires only Java), simple, reasonably efficient (though it is tuned for the Illumina platform and may be unstable on other types of data) and produces aesthetically pleasing plots. On the downside, some of its beautiful charts are uninformative and occasionally confusing. For example, in some cases, charts will switch from non-binned to binned columns within the same panel, easily misleading casual observers.

Then there are plots, such as the K-MER plot and the Overrepresented Sequences plot, that don't show what most people assume they do. There is also the difficulty of having to open HTML files one by one for result interpretation. In the current state of the field, where dozens of samples are being investigated in any given sequencing run, even just opening the results of the QC analysis becomes exceedingly tedious. The seemingly simple task to check the data quality requires clicking and scrolling through each file individually.

To some extent, even the tool's name is confusing. FASTQC does not perform quality control: it only visualizes the quality of the data.

But even with its flaws, FASTQC is still by far the best FASTQ quality visualization tool. It is one of the tools that scientific funding agencies should seek to fund even if the author were not to apply for funding.

# How does FastQC work?

FastQC generates its reports by evaluating a small subset of the data and extrapolating those findings to the entirety of the dataset. Many of the metrics are only computed on the first 200,000 measurements then are being tracked through the rest of the data.

The tool help contains detailed (although not entirely satisfactory) descriptions of what each data analysis plot is supposed to show.

# How do we run FastQC?

Obtain and unpack the data for different sequencing platforms:

```
curl -O http://data.biostarhandbook.com/data/sequencing-platform-data.tar.gz
tar xzvf sequencing-platform-data.tar.gz
```

Run the FastQC tool:

```
fastqc illumina.fq
```

This action generates an HTML file called `illumina_fastqc.html` that contains the results of the run. There are options you can pass to `fastqc` :

```
fastqc illumina.fq --nogroup
```

that will change the plots in some way, the `--nogroup` option for example turns off the binning on columns.

There is data from several platforms in the file you downloaded above. As of 2016, one will fail to run with `fastqc`

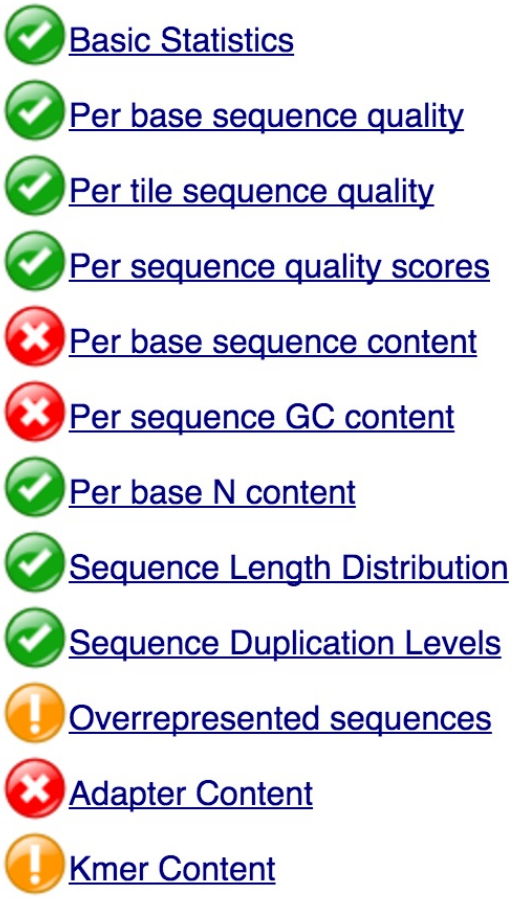
```
fastqc minion.fq
```

Will raise a memory error. This is just an indication how some tools are too closely tuned for one specific sequencing instrument. Even our quality control tool can fail on a 'standard' dataset that just happens to be from a different platform.

# Should I be worried about the "stoplight" symbols?

Usually not.

When FastQC runs, it generates "stoplight" icons for each analysis having "pass," "warning," and "error" symbols. Most of the time, these symbols are not meaningful. They were developed for only a particular class of samples and library preparation methods and only for certain types of instruments.




Though it is fair to say if most or all your stop light icons are red then you probably have a data problem.

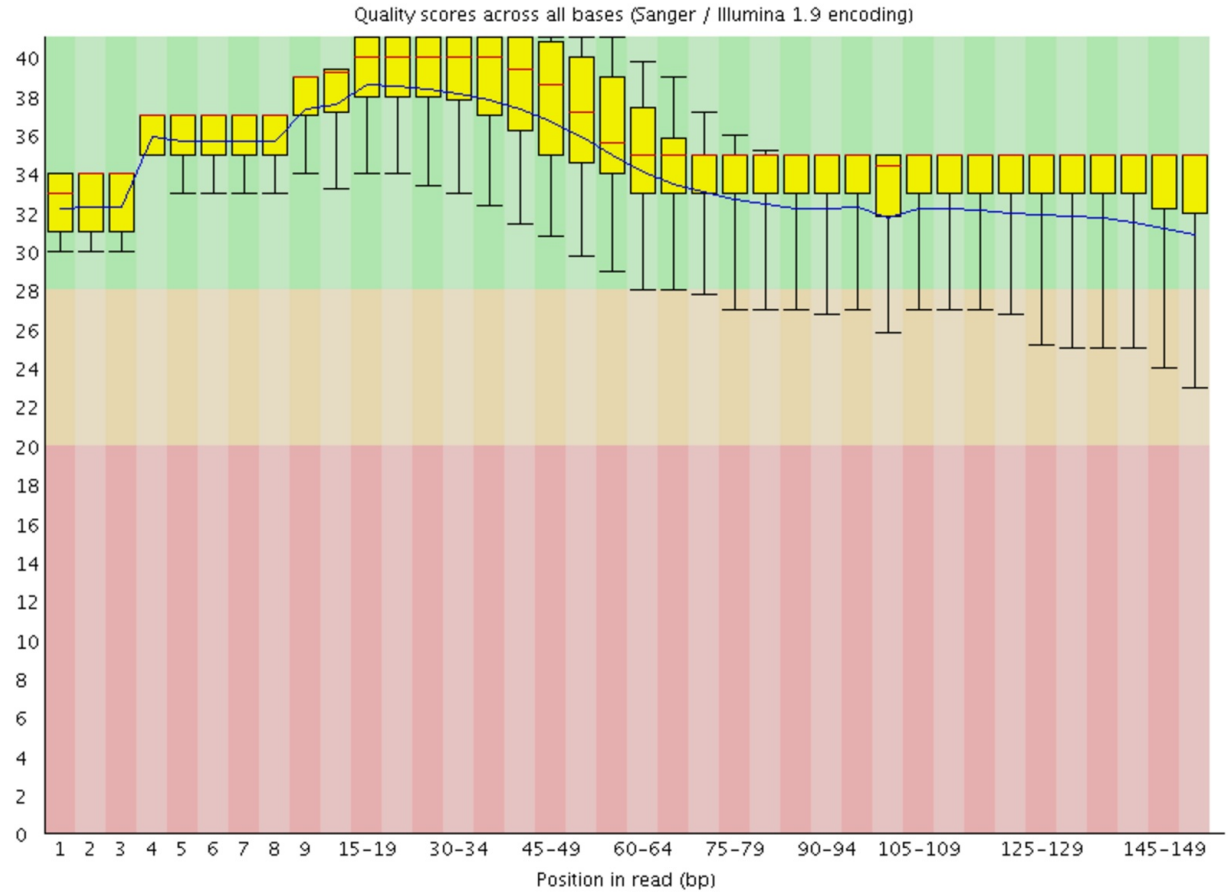
## What does the sequence quality visualization tell us?

The simplest way to snapshot the quality of a sequencing run is via a chart that plots the error likelihood at each position averaged over all measurements.

The vertical axis are the FASTQ scores that represent error probabilities:

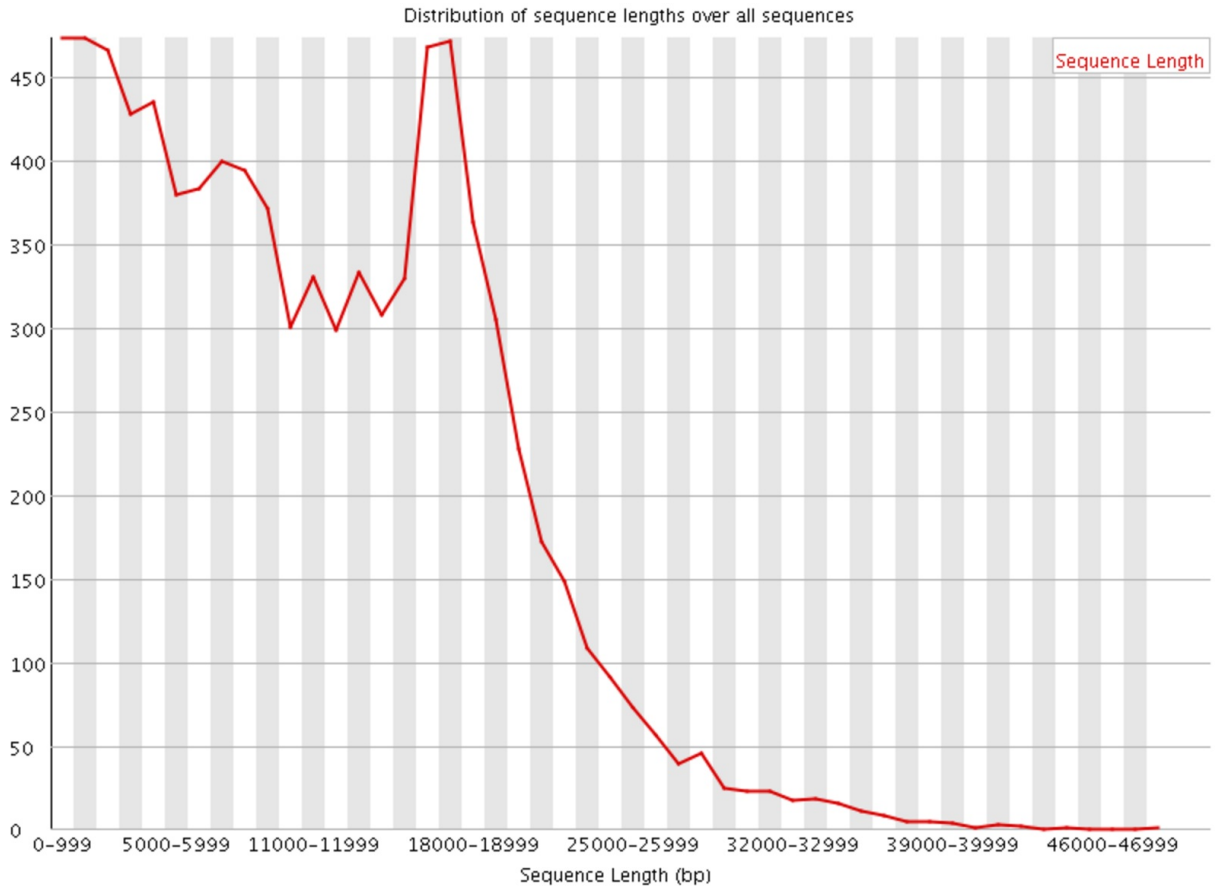
- 10 corresponds to 10% error (1/10),
- 20 corresponds to 1% error (1/100),
- 30 corresponds to 0.1% error (1/1,000) and
- 40 corresponds to one error every 10,000 measurements (1/10,000) that is an error rate of 0.01%.

The three-colored bands illustrate the typical labels that we assign to these measures: reliable (30-40, green), less reliable (20-30, yellow) and error prone (1-20, red). The yellow boxes contain 50% of the data, the whiskers indicate the 75% outliers.



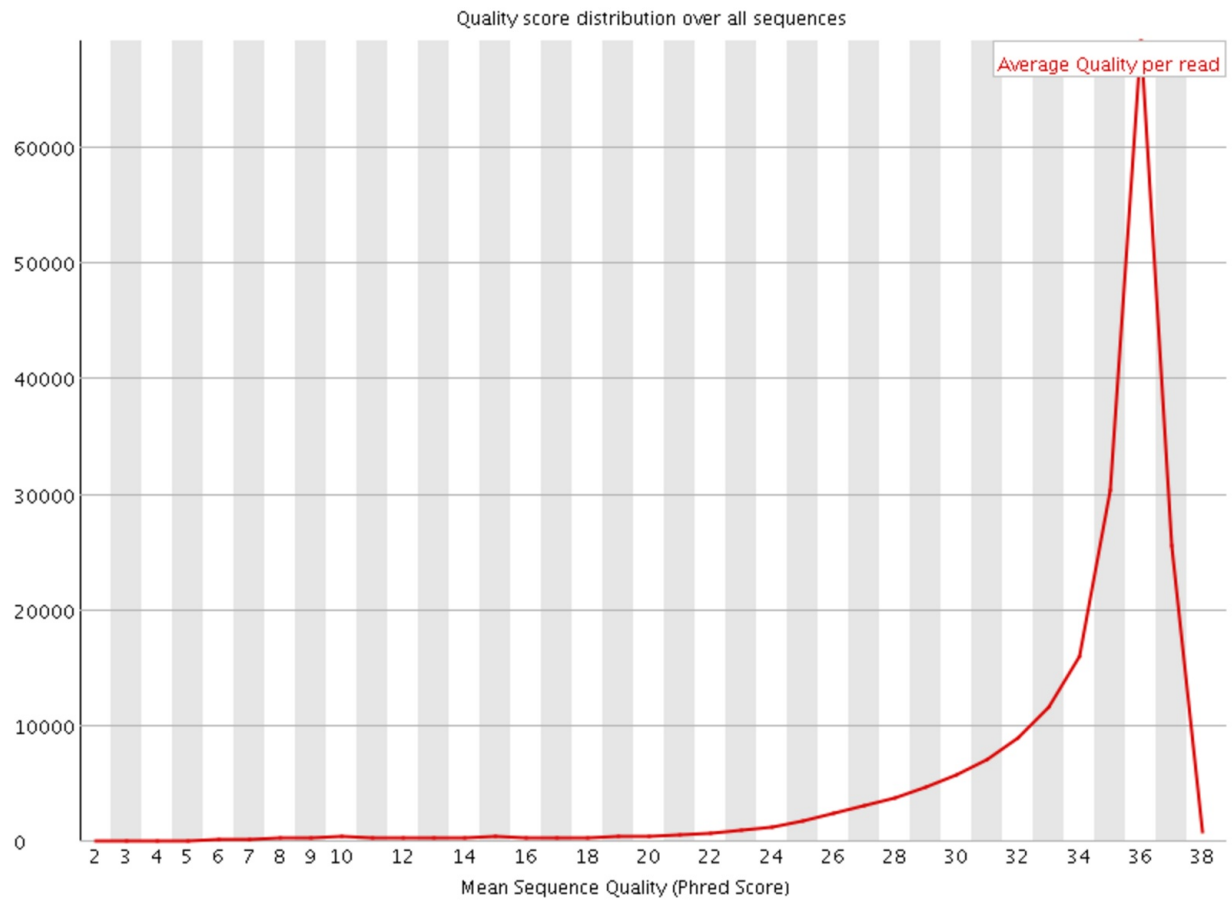

# What does the sequence length histogram show?

The sequence length distribution shows how many sequences of each length the data contains. For fixed read length instruments, like the Illumina sequencer, all read lengths are the same. For long read technologies like the PacBio and MinION, the distribution can be a lot more varied.




# What does the sequence quality histogram tell us?

Another way to visualize data quality is to generate the histograms of the average qualities. The horizontal scales are the quality scores; the vertical axis indicates the number of reads of that quality.




# What are sequencing adapters?

During library preparation, uniquely designed DNA adapters of lengths that are typically over 30 bases are added to the `5'` and `3'` ends of each sequence.

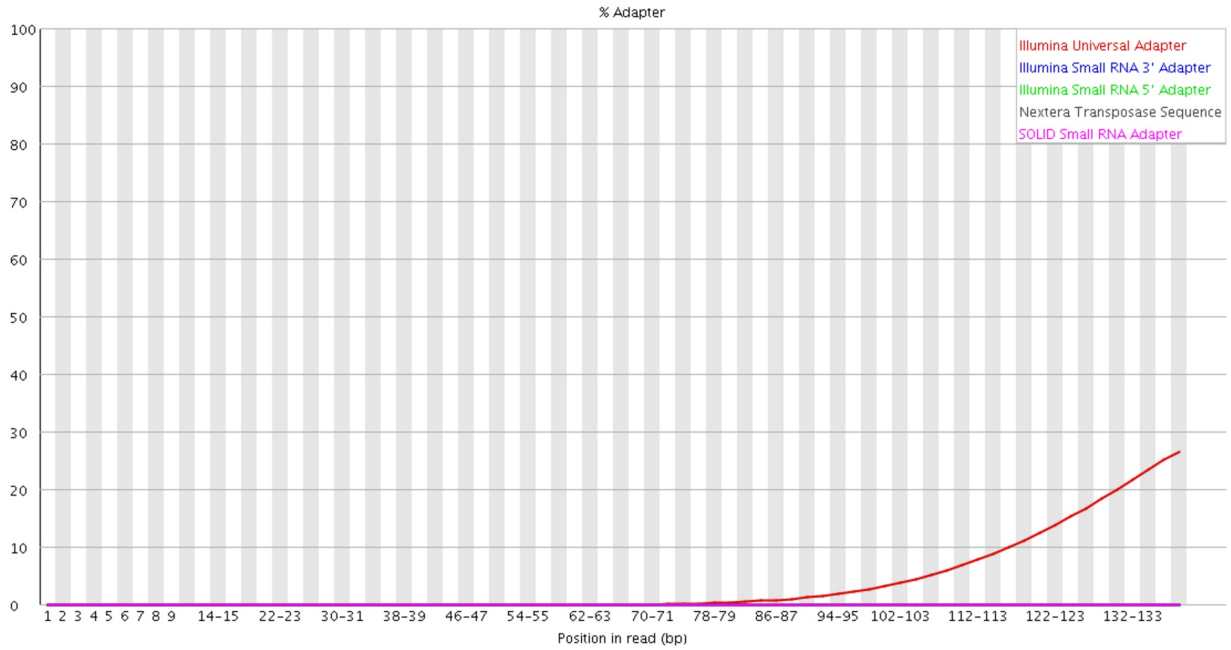
Hence, after adding the adapters, the single-stranded sequences that make it into the instrument will be of the form:

```
XXXXAAAATTTTGGGGCCCCYYYY
```

where `XXXX` and `YYYY` are the sequencing adapters. The instrument can recognize the starting adapters but usually won't detect the end adapters if these make it into the measurement. When the read length exceeds the length of the DNA fragment, the adapter sequence may appear in the data.

# How do I visualize the sequencing adapters?

FastQC will detect adapters based on Illumina sequences and will display them in the following manner:




The rising line indicates the number of adapter sequences contained at that base.

# Can we customize the adapter detection?

Yes. FASTQC has files that specify the adapters to search for. See the installation instructions that we provide for fastqc where we describe the location and content of the adapter file. Adding your adapters will allow you to have them displayed in the FASTQC plot.

# Do we have to remove adapters from data?

Adapter read-through problems may not need to be dealt with explicitly if the genomes that the data originated from are known.

Many high-throughput aligners can automatically clip the alignments and trim off adapters during the alignment phase.

On the other hand, the presence of adapter sequences may cause substantial problems when assembling new genomes or transcriptomes; hence they should be removed prior to starting that process.

# What is sequence duplication?

Duplication usually means having the same measurements in the data. Depending on the expected coverages duplicates may be correct measurements or errors. The word "duplicate" is misleading too; it seems to imply that there is "two" of something, what it means is "more than one".

Duplicates hence fall into two classes:

1. Natural duplicates - these were identical fragments present in the sample
2. Artificial duplicates - these are produced artificially during the sequencing process, PCR amplification,

detection errors

# How do we detect sequence duplication?

There are two main approaches:

1. Sequence identity - where we find and remove sequences that are have the exact sequence.
2. Alignment identity - where we find and remove sequences that align the same way.

As counterintuitive as that might sound, by filtering on sequence identity, we are running the risk of rewarding reads with errors in them. We are removing identical reads, yet keep those reads that were supposed to be identical but were measured erroneously. There is a danger of actually enriching our data with incorrect measurements.

On the other hand, alignment-based identity is only feasible if there is a known genome to align the data against.

There is a new class of methods that detect subsequence composition (called k-mers) that can be used to correct errors.

# What is the primary concern with duplicates?

The adverse effect of read duplication manifests itself mostly during variation detection. This process assigns a reliability score to each variant based on the number of times it has been observed. If a single read with a variant happens to be duplicated artificially, it runs the risk of producing a seemingly more reliable result than in reality.

# Should we remove duplicates?

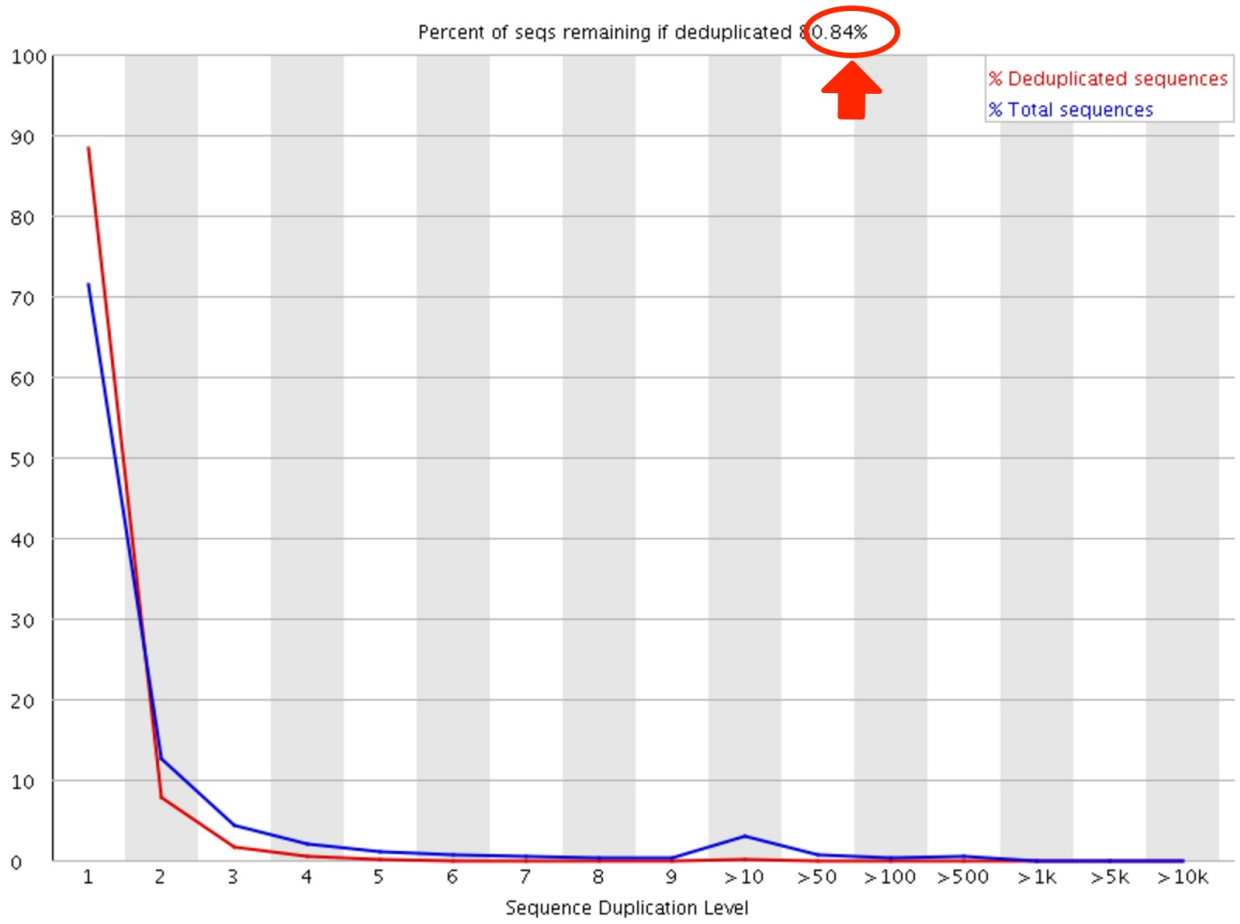
For SNP calling and genome variation detection, the answer is usually yes. For other processes, the answer is usually no.

Duplicate removal is a substantial alteration of the data - one should have additional evidence that none of the duplicates are of natural origin prior to doing it.

# What does the FastQC duplicate plot mean?

It is not all that simple to interpret the FastQC duplication report. We ourselves have struggled with it.

First of all, that small number circled at the top is one of the most important values. It shows what percent of the data is distinct.




- Biostar question of the day: Revisiting the FastQC read duplication report

There are two kinds of duplicate counts: one for all sequences, and another for distinct sequences.

As an example, take a situation where two sets would have to be built as

```
A A A A B B B B
```

and

```
A A A A A A A B
```

Both examples have only two distinct sequences: `A` and `B` and the duplication rate would be the same. We have 2 distinct options and 8 observations: `2/8`, But obviously, the sets' internal structure is quite different. We should not use the same measure to quantify the duplication of the first set as the second.

This set structure is what the blue and red lines try to capture and visualize for us.

- The blue lines show the percent (Y-axis) of **all sequences** that are duplicated at a given rate (X-axis).
- The red line shows the number of **distinct sequences** that are duplicated at a given rate.

To work this out explicitly. In the first example we have two sequences each duplicated at rate of 4, whereas in the second case we have one unique sequence and one sequence duplicated at the rate of 7.

# Quality control of sequencing data

## What is quality control?

Quality control (abbreviated as QC from now on) is the process of improving data by removing identifiable errors from it. It is typically the first step performed after data acquisition.

Since it is a process that alters the data, we must be extremely cautious not to inadvertently introduce new features into it. Ideally, we want the same data (information) only better (more accurate). But please do note:

> It is common to expect too much from quality control. Remember that QC cannot turn bad data into good data and we can never salvage what appears to be a total loss.

When the sequencing data looks really bad from the beginning, it is best to move on and collect new data. No amount of QC will save it. Only wasted work and frustrations lie that way.

## How critical is quality control?

The more unknowns about the genome under study, the more important it is to correct any errors.

When aligning against well-studied and understood genomes, we can recognize and identify errors by their alignments. When assembling a de-novo genome, errors can derail the process; hence, it is more important to apply a higher stringency for filtering.

## What can go wrong with the data?

Probably the greatest challenge of QC is our inability to foresee all the possible ways that data can be affected. Good data is all alike, but once things go awry, the possible problems that may need correcting can run a whole gamut: from puzzling to fascinating to outright maddening.

For some people, performing quality control is also somewhat of a psychological barrier. There are many that feel that they should only move to the next step after they have done everything possible to improve the data. They may run, rerun, alter the data, and spend weeks and months on it. Don't be one of those people:

> Data improvement via quality control is an incremental process with ever-diminishing returns. Once basic corrections have been made, no amount of tweaking will produce radically better outcomes.

In addition, a common pitfall is to keep tweaking QC in a manner that seems to "improve" the final results; the danger with this is overfitting

## When do we perform quality control?

Quality control is performed at different stages

- Pre-alignment: “raw data” - the protocols are the same regardless of what analysis will follow
- Post-alignment: “data filtering” - the protocols are specific to the analysis that is being performed.

# How do we perform quality control?

QC typically follows this process.

1. Evaluate (visualize) data quality.
2. Stop QC if the quality appears to be satisfactory.
3. If not, execute one or more data altering steps then go to step 1.

# How reliable are QC tools?

First, let's start with a mind-blowing observation: Many quality control tools are of very low software quality.

There was an era (one that may not even be over yet) when writing a high-throughput sequencing quality control tool seemed to be "in fashion" because such tools were considered "easy" to publish. A literature search may turn up dozens of such published tools. Unfortunately, in our experience, a majority of these tools are naïve solutions that perform surprisingly poorly in real-world scenarios.

It also doesn't help that there is a surprising amount of inconsistency in the way QC tools work. Even though the concepts that the tools implement appear to be well defined, we have never been able to produce the exact same outcomes when employing different tools with seemingly identical parameters.

Finally, the truth of the matter is that objective assessment of changes in data quality is difficult. What you will often see and hear are subjective, almost hand-waving assessments of what good data "looks like". Beyond that, there are few objective measures, standards, and recommendations that we can test against.

# Does quality control introduce errors?

This might be the second mind blowing observation - one that just about every bioinformatician seems to forget.

Make no mistake - quality control does introduce errors. It is just that we expect the number or effect of the new errors that the QC creates to be much smaller than the number of errors that the QC corrects. But you can see how at some point it is almost a philosophical issue - would you rather be dealing with errors that an instrument caused or errors that you yourself introduced when correcting for the machine errors?

Do not error correct data that does not need it.

# Is there a list of QC tools?

Quite a number of QC tools have been published. We have only tested a few of these and out of those we recommend `Trimmomatic`, `BBDuk`, `flexbar` and `cutadapt`. While all tools implement basic quality control methods, each often includes unique functionality specific to that tool.

Quality control tools are often full sequence manipulation suites. Here is a list (in alphabetical order):

- BBDuk part of the BBMap package
- BioPieces a suite of programs for sequence preprocessing
- CutAdapt application note in Embnet Journal, 2011
- fastq-mcf published in The Open Bioinformatics Journal, 2013
- Fastx Toolkit: collection of command line tools for Short-Reads FASTA/FASTQ files preprocessing - one of the first tools
- FlexBar, Flexible barcode and adapter removal published in Biology, 2012
- NGS Toolkit published in Plos One, 2012
- PrinSeq application note in Bioinformatics, 2011
- Scythe a bayesian adaptor trimmer
- SeqPrep - a tool for stripping adaptors and/or merging paired reads with overlap into single reads.
- Skewer: a fast and accurate adapter trimmer for next-generation sequencing paired-end reads.
- TagCleaner published in BMC Bioinformatics, 2010
- TagDust published in Bioinformatics, 2009
- Trim Galore - a wrapper tool around Cutadapt and FastQC to consistently apply quality and adapter trimming to FastQ files, with some extra functionality for MspI-digested RRBS-type (Reduced Representation Bisufite-Seq) libraries
- Trimmomatic application note in Nucleic Acid Research, 2012, web server issue

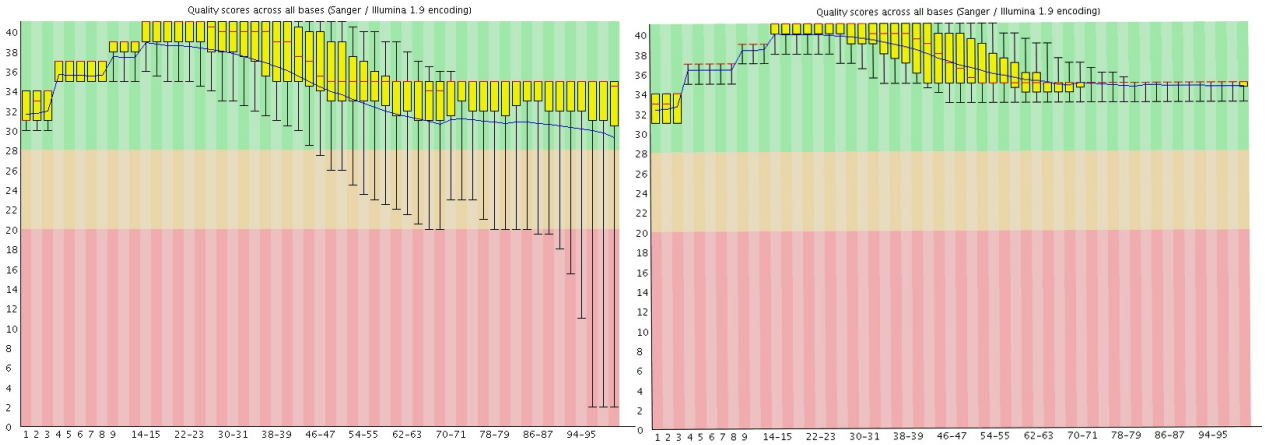
There also exist libraries via R (Bioconductor) for QC: PIQA, ShortRead

# How does read quality trimming work?

Originally, the reliability of sequencing decreased along the read. A common correction is to work backwards from the end of each read and remove low quality measurements from it. This is called trimming.

```
# Get the data
fastq-dump --split-files -X 10000 SRR1553607

# Run trimmomatic in single end mode
trimmomatic SE SRR1553607_2.fastq trimmed_2.fq SLIDINGWINDOW:4:30

# Generate fastqc reports on both datasets.
fastqc SRR1553607_1.fastq trimmed_2.fq
```

The resulting plot shows the original file on the left and the quality trimmed version on the right.




A similar (but not identical) result would be found using `bbduk` to trim adapters:

```
bbduk.sh in=SRR1553607_2.fastq out=bbduk.fq qtrim=r overwrite=true qtrim=30
```

To perform the same process on both files at the same time, we need a more complicated command

```
trimmomatic PE SRR1553607_1.fastq SRR1553607_2.fastq  trimmed_1.fq unpaired_1.fq trimmed_2.fq
unpaired_2.fq SLIDINGWINDOW:4:30
```

Or with BBDuk:

```
bbduk.sh in1=SRR1553607_1.fastq in2=SRR1553607_2.fastq outm1=bbduk_1.fq out1=unpaired_bb_1.fq
outm2=bbduk_2.fq out2=unpaired_bb_2.fq qtrim=r overwrite=true qtrim=30
```

# Why do we need to trim adapters?

As you recall, within the sequencing instrument, each read has artificial sequences attached to its end. If the sequence is `AAAAAAAAAA` and the adapter is `TTTTTT` then depending on the library size and read length the sequencing may run into the adapter.

```
AAAAAAAAAATTTTTT
------>
---------->
---------------->
```

The last read will have the adapter sequence `TTTTT` at the end.

If the overlap of bases into the adapter is sufficiently long (5-6), we can detect the adapter itself and cut the sequence at the start of the adapter.

Clearly, to recognize the adapters, we need to know what their sequence is. This information is surprisingly difficult to find out from those that put these adapters on. Most companies consider these as "trade secrets" and you would be hard pressed to find clear documentation of it. When found, it is usually buried deep in other information and needs to be fished out from it. For example, the Illumina TruSeq index adapters will have the following sequence:

```
# TruSeq Indexed Adapter
GATCGGAAGAGCACACGTCTGAACTCCAGTCACNNNNNNATCTCGTATGCCGTCTTCTGCTTG
```

To complicate matters even more, an additional `A` is ligated to the end of the sequences before the adapter is attached. Hence, the artificial sequence present at the end of some reads will typically be some fraction of the sequence:

```
AGATCGGAAGAGCACACGTCTGAACTCCAGTCAC
```

To see other adapters that for example the FastQC tool detects see:

```
cat ~/src/FastQC/Configuration/adapter_list.txt
```

That will produce:

```
Illumina Universal Adapter      AGATCGGAAGAG
Illumina Small RNA 3' Adapter   TGGAATTCTCGG
Illumina Small RNA 5' Adapter   GATCGTCGGACT
Nextera Transposase Sequence    CTGTCTCTTATA
SOLID Small RNA Adapter         CGCCTTGGCCGT
```

These sequences are used to detect adapters in the sequences. Many studies may add other adapters. Insert a custom adapter into this sequence to see it visualized in the FastQC report.

# How do we trim adapters?

We can create your own adapter or use the ones that come with Trimmomatic Let's create an Illumina adapter file.

```
echo ">illumina" > adapter.fa
echo "AGATCGGAAGAGCACACGTCTGAACTCCAGTCAC" >> adapter.fa
```

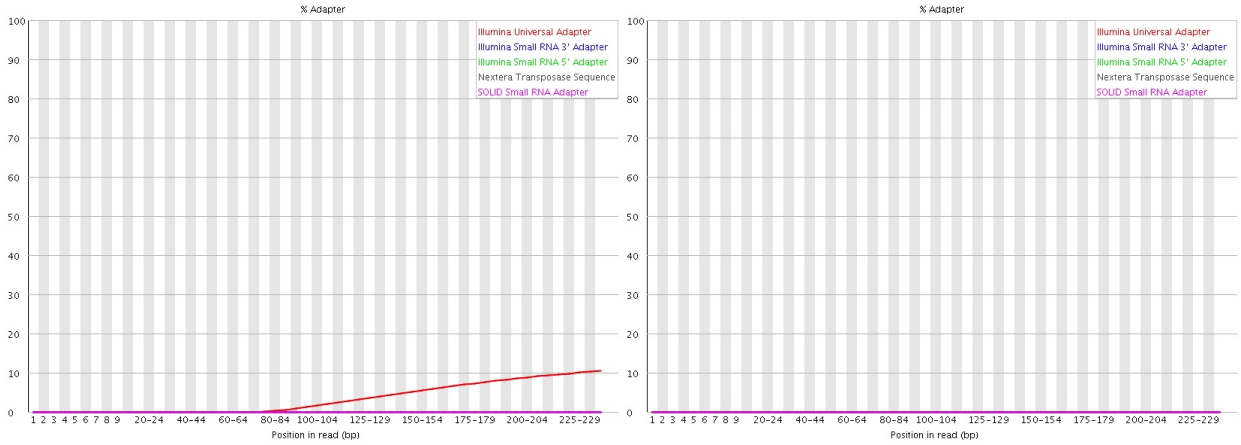
Let's use this to filter another SRA file:

```
fastq-dump -X 10000 --split-files SRR519926
fastqc SRR519926_1.fastq
```

Trim adapters with `trimmomatic`:

```
trimmomatic SE SRR519926_1.fastq output.fq ILLUMINACLIP:adapter.fa:2:30:5
```

The left plot shows the adapter content of the data before removing the adapters. The right-hand side shows the effect of the command above.




With `bbduk`, the trimming would be:

```
bbduk.sh in=SRR519926_1.fastq  ref=adapter.fa out=bbduk.fq
```

# How do we perform multiple steps at once?

Most tools allow us to chain together multiple actions, though usually the resulting command line is quite an eyeful.

```
trimmomatic PE SRR519926_1.fastq SRR519926_2.fastq trimmed_1.fq unpaired_1.fq trimmed_2.fq unp
aired_2.fq SLIDINGWINDOW:4:30 TRAILING:30 ILLUMINACLIP:adapter.fa:2:30:5
```

It is important to recognize that the order of action matters and results may be different depending on which action is performed first:

1. Do we first trim for quality then trim the adapters?
2. Do we trim the adapters and then trim for quality?

Since the adapters have to overlap if we trim by quality first, we may run the risk of removing a base that will not allow us to detect the adapter. On the other hand, if the base is of low quality, it may not align to the adapter; hence, the adapter won't be detected. Often the order of operation does not make a notable difference, but on problematic data it's best to experiment.

# Trimming adapter sequences - is it necessary?

There is no full consensus on the matter, although Brian Bushnell makes a very strong case that adapter trimming greatly improves the quality of data.

Read on the Biostar question of the day Trimming adapter sequences - is it necessary?

# Can there be other adapters?

Depending on the library preparation and experiment design other artificial sequences may be attached at the end of a DNA fragment:

```
fastq-dump -X 10000 SRR1553606 --split-files
```

The FastQC report will show a Nextera adapter contamination:

```
fastqc SRR1553606-1.fastq
```

Verify what FastQC can detect:

```
cat ~/src/FastQC/Configuration/adapter_list.txt
```

Create the adapter that is to be cut (this may need searching for information ) for convenience here is the Nextera adapter:

```
echo ">nextera" > nextera.fa
echo "CTGTCTCTTATACACATCTCCGAGCCCACGAGAC" >> nextera.fa
```

Trim the sequences with this new adapter.

# What is error correction?

Data produced by sequencing instruments may contain errors. The quality value in a FASTQ process is meant to indicate the reliability of a given measurement.

Since the errors are random, it is unlikely that they will occur in nearby locations. As a consequnce, if the data has high coverage, it is possible to recognize errors by identifying locations that have a high consensus and are affected only by very few measurements. This can be achieved even without knowing the genome that provided the DNA by computing the so called k-mer density of the data. In simple terms, a k-mer is a short subsequence of the data. k-mers that come from real data will be present a very large number of times. K-mers that contain errors will be rare.

A FASTQ error correction program will attempt to correct or remove reads that look to have errors in them. This can make some processes (especially genome assembly) more efficient.

When data is of high coverage, error correction might have a more pronounced beneficial effect.

# How do we correct errors?

The `bbmap` package includes the `tadpole.sh` error corrector:

```
fastq-dump -X 10000 --split-files SRR519926

# Tadpole is part of the bbmap package
 tadpole.sh in=SRR519926_1.fastq out=tadpole.fq mode=correct

# Error correction with bfc https://github.com/lh3/bfc
bfc SRR519926_1.fastq > bfc.fq
```

# Sequence patterns

## What is a sequence pattern?

A sequence pattern is a sequence of bases described by certain rules. These rules can be as simple as a direct match (or partial matches) of a sequence, like:

- Find locations where `ATGC` matches with no more than one error

Patterns may have more complicated descriptions such as:

- All locations where `ATA` is followed by one or more `GC` s and no more than five `T` s ending with `GTA`

Patterns can be summarized probabilistically into so called **motifs** such as:

- Locations where a `GC` is followed by an `A` 80% of time, or by a `T` 20% of the time, then is followed by another `GC`

## Can adapters be identified by a pattern?

Yes. Adapters in the data are among the simplest types of sequence pattern that may be present in a data set.

When the our sequences are "line oriented", that is the entire sequence is on a single line, the `grep` with the `--color=always` parameter will allow us to extract and color the matches:

```
fastq-dump -X 10000 SRR1553606 --split-files

cat SRR1553606_1.fastq | grep --color=always CTGTCTCTTATACA | head -2
```

will produce:

Most FASTQ files can be processed with line oriented tools.

> Do not use `--color=always` when saving data in a new file for other downstream processing. When you use coloring, additional color-related data is added to the file, and that will affect downstream processing.

## How can I search genomic sequences for patterns?

`grep` and extended grep `egrep` are tools that operate on one line at a time and can be used to identify lines with patterns. When sequences wrap over many lines we need dedicated tools like dreg or fuzznuc (Emboss Tools). See the following chapter [Regular expressions][regexp.md]

```
# This will miss patterns that wrap around new lines.
cat KU182908.fa | grep  AAAAAA

# The dreg tool matches and reports the locations.
cat KU182908.fa | dreg -filter -pattern  AAAAAA

# Search a pattern with ambiguous N bases.
cat KU182908.fa | fuzznuc -filter -pattern 'AANAA'
```

# Regular expressions

dreg or fuzznuc (Emboss Tools) are tools that allow you to search the genome for complex patterns.

## What are regular expressions?

Regular expressions are special sequences of characters that define a possibly complex search pattern.

For example

- `^` matches the beginning of the line
- `.` matches any character
- `{m,n}` matches the preceeding elements at least `m` but not more than `n` times.

The Regexp pattern language is unlike most other computing languages. It can be best learned via an interactive service like https://regexone.com/ or many others where the pattern and its effect are instantly visualized.

Example regular expression searches:

```
# Get a FASTQ dataset.
fastq-dump --split-files SRR519926

# Find an ATG anchored at the start of the line
cat SRR519926_1.fastq | egrep "^ATG" --color=always | head

# Find an ATG anchored at the end of the line
cat SRR519926_1.fastq | egrep "ATG\$" --color=always | head

# Find TAATA or TATTA patterns, this is a range of characters
cat SRR519926_1.fastq | egrep "TA[A,T]TA" --color=always | head

# Find TAAATA or TACCTA, these are groups of words
cat SRR519926_1.fastq | egrep "TA(AA|CC)TA" --color=always | head

# Quantify matches with metacharacters
# Find TA followed by zero or or more A followed by TA
cat SRR519926_1.fastq | egrep "TA(A*)TA" --color=always | head

# Find TA followed by one or or more A followed by TA
cat SRR519926_1.fastq | egrep "TA(A+)TA" --color=always | head

# Find TA followed by two to five As followed by TA
cat SRR519926_1.fastq | egrep "TAA{2,5}TA" --color=always | head

# Practice RegExp matches on online regexp testers
# http://regexpal.com/

# Match Ilumina adaptors at the end of the reads
# Match AGATCGG anywhere followed by any number of bases
cat SRR519926_1.fastq | egrep "AGATCGG.*" --color=always | head
```

# What are some challenges I may face when using regular expressions?

Think of a regular expression as a computing engine. There are different implementations of it, and each may operate with slightly different syntax, and at different speeds or efficiencies.

Get chromosome 22 of the Human Genome

```
curl http://hgdownload.cse.ucsc.edu/goldenPath/hg38/chromosomes/chr22.fa.gz | gunzip > chr22.fa
```

Let's find the telomeric repeats `TTAGGG` in the human genome. In principle this is easy: build and match the pattern. In practice, when we search at a genomic scale, things don't always go as planned.

Let's first check that this pattern is present. Use the `-i` flag to make the match case insensitive since the genome may lowercase regions (for example the repeating regions are marked as such in the human genome):

```
cat chr22.fa | egrep -i '(TTAGGG)' --color=always
```

The above won't work perfectly since this is line oriented matching and the genome wraps over many lines. But it is a good start. We can refine it more

```
cat chr22.fa | egrep -i '(TTAGGG){3,10}' --color=always
```

Then we can "linearize" the genome by removing the new line wrapping characters from it. That way we can look for the actual pattern:

```
cat chr22.fa | tr -d '\n' | egrep -o -i '(TTAGGG){20,30}' --color=always
```

To produce:

```
tagggttagggttagggttagggttagggttagggttagggttagggttagggttagggttagggttagggttagggttagg
gttagggttagggttagggttaggg
```

The same can be done with `dreg`, though note that its regular expression engine is less powerful and may not be able to process all patterns:

```
cat chr22.fa | dreg -filter -pattern '(TTAGGG){20,30}'
```

# K-Mers

## What is a k-mer?

A `k-mer` typically refers to all the possible substrings of length `k` that are contained in a string. For example if the sequence is `ATGCA` then

- The 2 base long `k-mers` ( `2-mers` ) are `AT` , `TG` , `GC` and `CA`
- The 3 base long `k-mers` ( `3-mers` ) are `ATG` , `TGC` and `GCA`
- The 4 base long `k-mers` ( `4-mers` ) are `ATGC` , `TGCA`
- The 5 base long `k-mer` ( `5-mer` ) is `ATGCA`

## What are k-mers good for?

The potential information gained from k-mers evolving quickly. Interesting applications using k-mers are:

- Error correction: rare k-mers are more likely to be caused by sequence errors.
- Classification: certain k-mers may uniquely identify genomes.
- Pseudo-alignment: new pseudo-aligners can match reads to locations based solely on the presence of common k-mers.

Computing k-mers is much faster than producing alignments -- hence they can massively speed up data interpretation.

Using the jellyfish k-mer counter:

```
# Get some sequence data.
efetch -id KU182908 -db nucleotide -format fasta > KU182908.fa

# Count the k-mers up to size 10.
jellyfish count -C -m 10 -s10M KU182908.fa

# Show a histogram of k-mers.
jellyfish histo mer_counts.jf

# The k-mers present at least 7 times.
jellyfish dump -L 7  mer_counts.jf

# Pick one k-mer, say TTAAGAAAAAA - is it present 7 times?
cat KU182908.fa | dreg -filter -pattern TTAAGAAAAA
```

## Should I use the k-mer report produced by FastQC?

The k-mer plot that FastQC produces should be interpreted with caution as it can be misleading. Both the scaling of the plot and the ratios of observed versus expected counts are flawed. Longer k-mers will be reported many times, but shifted by one base.

# Sequence Alignments

Sequence alignment is an essential concept for bioinformatics, as most of our data analysis and interpretation techniques make use of it. It is important to understand it well.

## What is a sequence alignment?

Sequence alignment (also called pairwise alignment) means arranging two sequences so that regions of similarity line up:

```
THIS-LI-NE-
--ISALIGNED
```

The way an alignment works makes sense intuitively, but be warned that this quality may also be problematic. Our intuition and the apparent simplicity of the process may mask what is taking place.

Our brains are hardwired to see patterns and similarities whenever possible, and once a similarity is shown, we are usually unable to reliably notice other alternatives that may be just as good (if not better) as the one we are currently observing. It is hard to "un-see" it.

## How are alignments displayed?

There are several ways that alignments can be reported and there is no simple, universal format that can present all the information encoded in an alignment.

Often, we use a visual display that employs various extra characters to help us "interpret" the lineup. For example:

- the `-` character may indicate a "gap" (space),
- the `|` character is used to display a match,
- the `.` character may be used to display a mismatch.

For example the following is an alignment between two sequences `ATGCAAATGACAAATAC` and `ATGCTGATAACT` :

```
ATGCAAATGACAAATAC
||||   |||.||.|
ATGC---TGATAACT--
```

Usually, if not stated explicitly otherwise, we read and interpret the alignment as if we were comparing the bottom sequence against the top one. In the case above we could say that it is an alignment with "deletions." This means that the second sequence has missing bases relative to the first.

We could display this same alignment the other way around, in which case the bottom sequence would have "insertions" relative to the top one.

```
ATGC---TGATAACT--
||||   |||.||.|
ATGCAAATGACAAATAC
```

# What is a CIGAR string?

The `CIGAR` string (Compact Idiosyncratic Gapped Alignment Report) is an alignment format used within the Sequence Alignment Map (SAM) files that form the backbone of most bioinformatics high-throughput analyses. For the same alignment from above:

```
ATGCAAATGACAAATAC
||||   |||.||.|
ATGC---TGATAACT--
```

the reported `CIGAR` format would be:

```
4M3D3M1X2M1X1M2D
```

We read out this "idiosyncratic" construct like so:

- 4 matches followed by
- 3 deletions,
- 3 matches,
- 1 mismatch,
- 2 matches,
- 1 mismatch,
- 1 match,
- 2 deletions.

The format above is in a so-called "Extended CIGAR," meaning that it employs the `X` symbol for mismatches.

Another variant where *both* matches and mismatches are designated by an `M` (as odd as that might sound) is the actual format used within the Sequence Alignment Map (SAM).

# How are alignments computed?

Suppose you had the "query" sequence `ATGAA` and some other short "target" sequences that appeared to be similar to it. Presume you saw four candidates that could be aligned like this:

```
  1       2       3       4

ATGAA   ATGAA   ATGAA   AT-GAA
|.|.|   |||.|   |||.|   || |||
ACGCA   ATGCA   ATGTA   ATCGAA
```

How do you pick the BEST alignment? As you can imagine, the best alignment will depend on what you "like" and how you "value" the way bases line up. Do you consider one, or even two mismatches less disruptive than a gap? The value you associate with a match, mismatch or a gap is called scoring.

Essential concept:

> There is no universally BEST alignment. There is only a BEST alignment relative to a SCORING choice. Changing the score will usually change what is called best alignment.
>
> Alignment algorithms find the arrangement that produces the maximal score.

# What is alignment scoring?

The scores are the values, both positive and negative, that we assign to various bases when they line up a certain way. An aligner attempts to produce an arrangement that maximizes the score.

For example, you could choose the following:

- `5` points for a match.
- `-4` points for a mismatch.
- `-10` points for opening a gap.
- `-0.5` points for extending an open gap.

With that scoring system, the alignments above would be scored as (scores shown at the bottom):

```
  1       2       3       4

ATGAA   ATGAA   ATGAA   AT-GAA
|.|.|   |||.|   |||.|   || |||
ACGCA   ATGCA   ATGTA   ATCGAA

7       16      16       15
```

Alignments 2 and 3 have identical scores of `16` and would be the "best" (highest scoring) alignments.

# How do I choose the scores?

The scores are typically computed by observing the actual substitution rates across evolutionary history. There are multiple ways we can consolidate these observed sequences - hence there are multiple possible scoring matrices.

Good starting points already exist. For example, the EDNAFULL scoring matrix is the default nucleotide scoring choice for just about all aligners. Typically, unless you set the scores yourself, you will use these values:

```
curl -O  ftp://ftp.ncbi.nlm.nih.gov/blast/matrices/NUC.4.4
```

then print it with:

```
cat NUC.4.4
```

The actual matrix is large, as it includes all the ambiguous bases as well (refer to the biology intro for information on those); the following section is relevant:

```
   A   T   G   C
A  5  -4  -4  -4
T -4   5  -4  -4
G -4  -4   5  -4
C -4  -4  -4   5
```

This section shows that the score contribution from matching an `A` with `A` is 5 points, whereas the score contribution from matching an `A` with a `T` is -4 points (i.e. a score penalty of 4 points).

But then, of course, picking the right matrix is more complicated. For a quick overview see:

- Selecting the Right Similarity-Scoring Matrix by William Pearson, the author of the FASTA program.

# What kind of scoring matrices are there?

There are two types: nucleotide scoring matrices and protein scoring matrices. The protein scoring matrices come in many variants computed with different assumptions on what similarity means. Also, (if that wasn't enough) the scoring matrices can be normalized and non-normalized.

Care must be taken when comparing alignment scores computed with different scoring matrices, even if they seem to have the same name! One could be normalized and the other not.

Visit: ftp://ftp.ncbi.nlm.nih.gov/blast/matrices for a list of matrices.

# What other properties do scoring matrices have?

Observe that the scoring matrix does not include the information on the gap penalties. It is also worth noting that the gap opening and extension penalties are typically different. Specifically, the gap extension penalty is usually much smaller than the gap opening one. This scoring (also called affine gap scoring) has a logical and biologically relevant rationale that we invite you to investigate on your own.

It is essential to understand just how impactful the choice of scoring is and how profound its effects are because it governs how many gaps versus mismatches the aligner produces.

A final note: a custom scoring matrix can only work in a biologically meaningful way if it sums up to negative values along both of its rows and columns. It is also worth keeping in mind that the scores are typically on a logarithmic scale.

# Are there other ways to quantify alignments?

Beyond the score of an alignment, scientists may also make use of concepts such as

- Percent Identity: What percentage of the two sequences are the same
- Percent Similarity: What proportion of the two sequences have similar bases/aminoacids
- EValues: Number of observed alignments by chance
- Mapping quality: The likelihood of incorrect alignment

## How do I perform a pairwise alignment?

[TODO]: list of tools

## Where can I learn more about alignments?

There is a large body of work and information on alignments and scoring that are beyond the scope of this book. There are good starting points in Wikipedia, as well as in many other easy-to-find web resources.

- Wikipedia: Sequence Alignment
- Wikipedia: Gap Penalty

# Global Alignments

To run the examples below, first install the Emboss package, then install the two simple tool wrappers that we have prepared; they are available from the data site. The same alignments can also be performed online via the Ensembl Pairwise Alignment Tool webpage.

```
# Store the program in the bin folder.
mkdir -p ~/bin

# Install the wrapper for the EMBOSS alignment tools.
curl http://data.biostarhandbook.com/align/global-align.sh > ~/bin/global-align.sh
curl http://data.biostarhandbook.com/align/local-align.sh > ~/bin/local-align.sh

# Make the scripts executable.
chmod +x ~/bin/*-align.sh
```

In the examples below, we will use hypothetical protein sequences `THISLINE` and `ISALIGNED`, as these form real words that are easy to read and so better demonstrate the differences between alignments. These sequences where first used for a similar purpose, albeit in a different context, in *Understanding Bioinformatics* by *Marketa Zvelebil* and *Jeremy Baum*

# What is global alignment?

In global alignment, every base of both sequences has to align to another matching base, to another mismatching base, or to a gap in the other sequence. For example,

```
global-align.sh THISLINE ISALIGNED
```

produces:

```
THISLI--NE-
  ||.:  ||
--ISALIGNED
```

Many global alignments do not penalize gaps at the end of the sequences. This is because the tools may have been built to expect incomplete measurements that would manifest themselves with missing values at the end.

We may also override the gap open and gap extension penalty:

```
global-align.sh THISLINE ISALIGNED -gapopen 7
```

This produces a different alignment:

```
THIS-LI-NE-
  || || ||
--ISALIGNED
```

Note how radically different the second alignment is from the first. We reduced the penalty of opening a gap to from `10` to `7`; hence, the alignment produced more gaps, but it also recovered more matches. The tradeoff is readily apparent.

The full list of parameters are:

```
-gapopen
-gapextend
-data
```

Here the `-data` parameter is used to pass scoring matrices to the aligner. Global alignments are used when we need to find the similarity over the entire length of both sequences.

# Are scores computed the same way?

One commonly used variant of both global and semi-global alignments is the free-end-gap method, where gaps at the end of the alignment are scored differently and with lower penalties.

It is not always clear from the description of the software when this choice is made. Do note that most tools used to investigate biological phenomena use free-end-gap scoring.

# Local alignments

To run the examples, please install the alignment tools as instructed at the beginning of section Global Alignment

## What is a local alignment?

When performing local alignments, the algorithms look for the highest scoring subregions (or single region):

```
local-align.sh THISLINE ISALIGNED
```

The default local alignment will be surprisingly short:

```
NE
||
NE
```

The algorithm is telling us that these two matching amino acids produce the highest possible score ( `11` in this case) and any other local alignment between the two sequences will produce a score that is worse than `11` .

We can use other scoring matrices:

```
# DNA matrices
# This matrix is the "standard" EDNAFULL substitution matrix.
wget ftp://ftp.ncbi.nlm.nih.gov/blast/matrices/NUC.4.4

# Protein matrices
# Get the BLOSUM30, BLOSUM62, and BLOSUM90 matrices.
wget ftp://ftp.ncbi.nlm.nih.gov/blast/matrices/BLOSUM30
wget ftp://ftp.ncbi.nlm.nih.gov/blast/matrices/BLOSUM62
wget ftp://ftp.ncbi.nlm.nih.gov/blast/matrices/BLOSUM90
```

Note how the local alignment changes based on the scoring scheme.

```
local-align.sh THISLINE ISALIGNED -data BLOSUM90
```

With this scoring scheme, for example, a much longer alignment is produced.

```
SLI-NE
:|| ||
ALIGNE
```

Local alignments are used when we need to find the region of maximal similarity between two sequences.

# What is a semi-global alignment?

A semi-global alignments (global-local, glocal) is a method mixture between the global and local alignments.

Semi-global alignments attempt to fully align a shorter sequence against a longer one. We don't have a simple command line semi-global aligner; if we did have one, though, its aligning of `THISLINE` and `ISLI` would produce

```
THISLINE
  ||||
  ISLI
```

Semi-global aligners are used when matching sequencing reads produced by sequencing instruments against reference genomes. Most data analysis protocols that we cover in this book rely on tools that use this type of alignment.

# Misleading alignments

## Will alignments find the "real" changes?

Finally, we want to demonstrate a common situation that you may encounter to demonstrate the limitations of using mathematical concepts to identify biological phenomena.

Imagine that the sequence below is subjected to two insertions of `C`s at the locations indicated with carets:

```
CCAAACCCCCCCTCCCCCGCTTC
     ^       ^
```

The two sequences, when placed next to one another, would look like this:

```
CCAAACCCCCCCTCCCCCGCTTC
CCAAACCCCCCCCTCCCCCCGCTTC
```

A better way to visualize what's happening in this example is shown in an alignment that reflects the changes that we have introduced:

```
CCAAA-CCCCCCCT-CCCCCGCTTC
||||| |||||||| ||||||||||
CCAAACCCCCCCCTCCCCCCGCTTC
```

But suppose we did not know what the changes were. Obviously, we would like to recover the alignment above to identify the two insertions. Can we discover this same by using a global aligner? Let's see:

```
global-align.sh CCAAACCCCCCCTCCCCCGCTTC CCAAACCCCCCCCTCCCCCCGCTTC
```

Here is what we obtain:

```
CCAAACCCCCCC--TCCCCCGCTTC
||||||||||||  .||||||||||
CCAAACCCCCCCCTCCCCCCGCTTC
```

Whoa! What just happened? Our aligner reports the two separate insertions of `C`s as a single insertion of a `CT` followed by a mismatch of `T` over `C`. To add insult to injury, the change is shown to be in a completely *different location*, not where we have actually modified the sequence! That's just crazy! How did that happen? What can we do about it?

In a nutshell, what happened is that there was a "simpler" explanation to reconcile the differences between the two sequences - and the aligner, using mathematical reasoning, produces the simplest explanation for us - even though this explanation does not reflect what actually happened. Some people use the word

"misalignment", but that would be incorrect - the aligner did its job correctly, but there was no way for it to know that a more complicated explanation was the correct one.

The main reason for the existence of this "simpler explanation" lies in the fact that the region between the two insertions has low information content; the same base is repeated: `ccccccc`. This throws off the neatly-crafted mathematical alignment concept of rewarding matching bases. When shifted by one base, we still get the same type of bases lined up, hence producing a positive score even though it is the "newly inserted" `C` that matches the "original" `C`. Let's compute the scores.

In our "correct" alignment, and when using the `EDNAFULL` matrix, the score will be formed from 23 matches and 2 gap openings:

```
(23 * 5) - (2 * 10) = 95
```

In the "incorrect" second alignment we have only 22 matches with 1 gap open, 1 gap extension and 1 mismatch leading to a score of

```
(22 * 5) - 10 - 0.5 - 4 = 95.5
```

An optimal aligner finds the alignment that maximizes the score. We now see that the second, "incorrect" alignment produces a slightly better score; hence, it will be reported as the most likely match between the two sequences.

We could recover the "correct" alignment by modifying the parameters to reduce the gap open penalty:

```
global-align.sh CCAAACCCCCCCTCCCCCGCTTC CCAAACCCCCCCCTCCCCCCGCTTC -gapopen 9
```

Now it will produce the expected output

```
CCAAA-CCCCCCCT-CCCCCGCTTC
||||| |||||||| ||||||||||
CCAAACCCCCCCCTCCCCCCGCTTC
```

Does this mean that from now on we should run all alignments with `gapopen=9`?

Absolutely not!

As we mentioned before, tweaking alignment parameters can have far-reaching consequences on all other alignments that are typically correct. Using this setting means that two matches `5 + 5 = 10` will overcome the penalty of a gap open `9`; hence, the aligner will open gaps any time it can find two matches later on. Sometimes that is a good idea, but that is most likely not what we intended to do.

The problem above was caused not by the incorrect alignment parameters but by the reduced information content of the "all-`C`" (aka homopolymeric) region.

Situations such as the above will produce incorrect polymorphism calls, and are an ongoing, profound, and ubiquitous challenge when identifying genomic variations. As we shall see, the latest SNP calling tools have the ability to recognize and correct mis-alignments.

A universal take-home message:

> Alignment reliability depends on the information content of the aligned sequence itself. Alignments that include low complexity regions are typically less reliable. Additional analysis is typically needed to confirm these results.

# Multiple sequence alignments

A multiple sequence alignment is one that uses three or more sequences. Since you already saw that even pairwise alignments can be substantially customized it should come at no surprise that since multiple sequence alignment involves reconciling differences across more than two sequences there are several different ways to go about it.

Just as with pairwise alignment there are optimal and non-optimal but much speedier methods to perform the same analysis.

## How do I align more than two sequences?

Suppose we wanted to align several full length ebola viruses.

```
# Store the genomes in this folder.
mkdir -p genomes

# Get all genomes for the ebola project.
esearch -db nuccore -query PRJNA257197 | efetch -format fasta > genomes/ebola.fa
```

The file contains 249 full length genomes:

```
seqkit stat genomes/ebola.fa
```

that produces:

```
file              format  type  num_seqs    sum_len  min_len   avg_len  max_len
genomes/ebola.fa  FASTA   DNA        249  4,705,429   18,613  18,897.3   18,959
```

Initially let's just align the first ten full ebola sequences against one another. Find the ID of the first ten sequences:

```
seqkit seq -n genomes/ebola.fa | cut -f 1 -d  ' ' | head -10 > ids.txt
```

The `ids.txt` file now contains ten accession numbers:

```
KR105345.1
KR105328.1
KR105323.1
...
```

Using these accession numbers we can extract the sequences that correspond to these ids:

```
seqkit grep --pattern-file ids.txt  genomes/ebola.fa > small.fa
```

Perform a multiple sequence alignment with the `mafft` tool:

```
mafft --clustalout small.fa > alignment.maf
```

View the alignment:

```
head -20 alignment.maf
```

Displays a visual alignment of all sequences

```
KR105345.1      -------ataattttcctctcattgaaatttatatcggaatttaaattgaaattgttact
KR105328.1      --gattaataattttcctctcattgaaatttatatcggaatttaaattgaaattgttact
KR105323.1      --gattaataattttcctctcattgaaatttatatcggaatttaaattgaaattgttact
KR105302.1      --gattaataattttcctctcattgaaatttatatcggaatttaaattgaaattgttact
KR105295.1      ---attaataattttcctctcattgaaatttatatcggaatttaaattgaaattgttact
KR105294.1      --gattaataattttcctctcattgaaatttatatcggaatttaaattgaaattgttact
KR105282.1      --gattaataattttcctctcattgaaatttatatcggaatttaaattgaaattgttact
KR105266.1      ---attaataattttcctctcattgaaatttatatcggaatttaaattgaaattgttact
KR105263.1      aagattaataattttcctctcattgaaatttatatcggaatttaaattgaaattgttact
KR105253.1      ---attattaatyttcctctcattgaaatttatatcggaatttaaattgaaattgttact
                       **** **********************************************
```

We can see that one difference between the genomes is how complete the assemblies are at their edges.

```
tail -20 alignment.maf
```

Similarly towards the ends:

```
KR105345.1      ggaaaaatggtcacacacaaaaatttaaaaataaatctatttcttcttttttgtgtgt
KR105328.1      ggaaaaatggtcgcacacaaaaatttaaaaataaatctatttcttcttttttgtgtgt
KR105323.1      ggaaaaatggtcgcacacaaaaatttaaaaataaatctatttcttcttttttgtgtgt
KR105302.1      ggaaaaatggtcgcacacaaaaatttaaaaataaatctatttcttcttttttgtgtg-
KR105295.1      ggaaaaatggtcgcacacaaaaatttaaaaataaatctattt----------------
KR105294.1      ggaaaaatggtcgcacacaaaaatttaaaaataaatctatttcttcttttttgtgtgt
KR105282.1      ggaaaaatggtcgcacacaaaaatttaaaaataaatctatttcttcttttttgtgtg-
KR105266.1      ggaaaaatgg------------------------------------------------
KR105263.1      ggaaaaatggtcgcacac-----------------------------------------
KR105253.1      ggaaaaatggtcgcacacaaaaatttaaaaataaatctatttcttcttt---------
                **********
```

The `*` character indicates a consensus.

Play around and perform multiple sequence alignments on 10% of the genomes:

```
seqkit sample --proportion 0.1 genomes/ebola.fa  > small.fa
mafft --clustalout small.fa > alignment.maf
```

# What programs can be used to align multiple sequences?

Tools include `clustal-omega`, `mafft`

(TODO: COMING-SOON)

# Basic Local Alignment Search Tool (BLAST)

A suite of tools

- `blastn`, `blastp`, `blastx`, `tblastn`, `tblastx` and others.
- Each can be further tuned via so-called tasks `-task` an tuning of parameters: `megablast`, `blastp-short` etc.

## What is BLAST?

BLAST (Basic Local Alignment Search Tool) is an algorithm and a suite of tools named by the algorithm that allow very fast searching for similarities of a so called "query" sequence against a database that may potentially contain a very large number of sequences (called subjects or targets).

The results of a BLAST search are local alignments.

BLAST is available both as a web interface from NCBI and as standalone downloadable tools. NCBI also maintains the Blast Handbook.

BLAST is not an optimal aligner. For example, there are limits for how short sequences can be for BLAST to operate correctly. In addition, BLAST is tuned for performance and efficiency and may not report all possible alignments. Its purpose is to search a large body of known information for similarities (hits).

## What are the key concepts of BLAST?

- Search may take place in nucleotide and/or protein space or translated spaces where nucleotides are translated into proteins.
- Searches may implement search “strategies”: optimizations to a certain task. Different search strategies will return different alignments.
- Searches use alignments that rely on scoring matrices
- Searches may be customized with many additional parameters. BLAST has many subtle functions that most users never need.

Knowing BLAST can be a “standalone job”. There are books written on just BLAST.

## What are the basic steps for using BLAST?

1. Prepare a BLAST database with `makeblastdb`. This only needs to be done once.
2. Pick a blast tool: `blastn`, `blastp` as appropriate (you may need to tune the parameters).
3. Run the tool and format the output as needed.

## What are the blast tools?

Blast is is an entire suite of somewhat confusingly named programs.

- `blastn`, `blastp`, `tblastn`, `blastx` etc. Each tool perform a certain type of query (nucleotide or peptide) versus a certain type of target (nucleotide or peptide) using a certain type of translation (nucleotide or peptide). It is often difficult to remember which blast tool does what as the naming is not all that logical.

As strange as that might sound, when figuring out which blast tool is more appropriate, it makes more sense to try to remember what each tool should have been named and instead of what they are actually named.

1. A blast search tool where both the query and target are in *nucleotide* space should have been called `blastNN` (but instead it is called just `blastn`).
2. A blast tool that searches a *peptide* sequence against a *nucleotide* sequence in the translated space should have been called `blastPN` but instead it is called `tblastn`

Here's a helpful table that clarifies the blast tools:

| Query sequence type | Database sequence type | Alignment level type | What the program should be called | What the program is actually called |
|---|---|---|---|---|
| nucleotide | nucleotide | nucleotide | **blastNN** | **blastn** |
| peptide | peptide | peptide | **blastPP** | **blastp** |
| nucleotide | peptide | peptide | **blastNP** | **blastx** |
| peptide | nucleotide | peptide | **blastPN** | **tblastn** |
| nucleotide | nucleotide | peptide | **blastNNP** | **tblastx** |

# What is the Blast terminology?

- **Query**: the sequence that we are looking to match.
- **Target**: the database (collection of sequences) that we are searching for matches.
- **Subject**: how we refer to an entry that matches.
- **Score**: the alignment score.
- **E-Value**: the Expect value (E) describes the number of hits better than the current hit that one can "expect" to see just by chance when searching a database of a particular size.

# How do I build a BLAST database?

Get a genome sequence for the Ebola virus. Indexing will produce several new files that we don't otherwise need. It is best if we place these in a separate folder outside of our main project folder. We put them into `~/refs`

```
# Store all references here.
mkdir -p ~/refs/ebola

# Get an ebola genome.
efetch -db nucleotide -id KM233118 --format fasta > ~/refs/ebola/KM233118.fa
```

The reference may contain one or more FASTA records. To turn it into a blast database, we need to transform it to a format the program is able to operate on. The blast database indexer is called `makeblastdb`

```
# See long and short help for the blast database indexer.
makeblastdb -h
makeblastdb -help

# Run the blast indexer.
makeblastdb -in ~/refs/ebola/KM233118.fa -dbtype nucl -out ~/refs/ebola/KM233118
```

If our FASTA file identifiers are formatted via the NCBI specifications, we may run this same blast database maker but with `-parse-seqids` option. It will then read out and store more information from the sequence ids.

```
makeblastdb -in ~/refs/ebola/KM233118.fa -dbtype nucl -out ~/refs/ebola/KM233118 -parse_seqids
```

Note that we could have named the index as something else, as well.

```
makeblastdb -in ~/refs/ebola/KM233118.fa -dbtype nucl -out ~/refs/ebola/FOO

# Now look at what files it has created.
ls ~/refs/ebola
```

We can run a query against the database. Make the query the start of the genome.

```
echo ">test" > query.fa
echo "AATCATACCTGGTTTGTTTCAGAGCCATATCACCAAGAT" >> query.fa
```

Run the `blastn` tool

```
blastn -db ~/refs/ebola/KM233118 -query query.fa
```

It generates a fairly lengthy output. The alignment specific section is:

```
 Score = 73.1 bits (39),  Expect = 5e-17
 Identities = 39/39 (100%), Gaps = 0/39 (0%)
 Strand=Plus/Plus

Query  1   AATCATACCTGGTTTGTTTCAGAGCCATATCACCAAGAT  39
           |||||||||||||||||||||||||||||||||||||||
Sbjct  1   AATCATACCTGGTTTGTTTCAGAGCCATATCACCAAGAT  39
```

# How do I format the output differently?

The need to extract information on alignments is very common. BLAST allows us to format the outputs into tabular and other formats. Change it to output format `6` or `7` (tabular form with or withouth comments and headers):

```
blastn -db ~/refs/ebola/KM233118 -query query.fa  -outfmt 7
```

We can even add custom fields to the output. Run `blastn -help`, then scroll through the output to find more details:

```
...
   qseqid means Query Seq-id
      qgi means Query GI
        qacc means Query accesion
     qaccver means Query accesion.version
        qlen means Query sequence length
      sseqid means Subject Seq-id
   sallseqid means All subject Seq-id(s), separated by a ';'
         sgi means Subject GI
      sallgi means All subject GIs
...
```

The command now reads:

```
blastn -db ~/refs/ebola/KM233118 -query query.fa  -outfmt "6 qseqid sseqid pident"
```

That produces a reduced information, query id, subject id, percent identity:

```
test    gi|667853353|gb|KM233118.1|    100.000
```

# What are blast tasks?

Tasks are algorithmic modification of the method (or its parameters) that make a blast tool more suited for finding certain types of alignments.

Confusingly, the task itself may be named the same as the tool. For example, `blastn` can have the following tasks:

- `blastn` - finds more divergent sequences
- `megablast` - finds less divergenent sequences
- `blastn-short` - short queries

Even more confusingly, the default task for the `blastn` tool is the `megablast` task. Note how this, again, is the beginning of the genome:

```
echo ">short" > short.fa
echo "AATCATACCTGG" >> short.fa
```

Yet this search will find no hits:

```
blastn -db ~/refs/ebola/KM233118 -query short.fa
```

We have to run `blastn` with a different search strategy `blastn` to get a hit:

```
blastn -db ~/refs/ebola/KM233118 -query short.fa -task blastn
```

Make the sequence even shorter:

```
echo ">mini" > mini.fa
echo "AATCATA" >> mini.fa
```

Now, only the `blastn-short` search strategy will produce results:

```
blastn -db ~/refs/ebola/KM233118 -query mini.fa -task blastn-short
```

# How do I perform pairwise alignments with BLAST?

If only two (relatively short) sequences need to be aligned, the database building does not need to be done

```
blastn -query query.fa -subject ~/refs/ebola/KM233118.fa
```

# Will blast find all alignments?

We need to remember that the default operation of blast is tuned to the most common use cases. For example, it will filter out all so-called "low complexity" regions.

```
# Chromosome 1 of the yeast genome.
efetch -id NC_001133 -db nucleotide -format fasta > NC_001133.fa

# Keep the first line (70 bases) of the genome.
head -2 NC_001133.fa > start.fa
```

But now look what happens; this produces no hits:

```
blastn -query start.fa -subject NC_001133.fa
```

The query sequence has some repetitive elements (low complexity) and gets filtered out automatically.

```
CCACACCACACCCACACACCCACACACCACACCACACACCACACCCACACACACACATCCTAACA
```

Most of the time, this is a good idea, but not this time. We need to turn off low complexity filtering corrections that just happen to be called `-dust`:

```
blastn -query start.fa -subject NC_001133.fa -dust no
```

# BLAST Databases

For BLAST to operate successfully, it needs to "index" the target sequences. The "index" is a BLAST-specific database that the BLAST tools need to perform their work. Different indices are necessary for nucleotide and peptide sequences.

- `makeblastdb` creates blast databases.
- `blastdbcmd` queries blast databases.
- `update_blastdb.pl` updates prebuilt blast databases.

## How do we make BLAST databases?

Get all proteins for project `PRJNA257197` :

```
# Make a directory to store the index
mkdir -p index

# Download the proteins from NCBI.
esearch -db protein -query PRJNA257197 | efetch -format fasta > index/all-proteins.fa

# Build the blast index.
makeblastdb -in index/all-proteins.fa -dbtype prot -out index/all -parse_seqids

# List the content of the blast database.
blastdbcmd -db index/all -entry 'all' -outfmt "%a" | head
```

## Can we download BLAST databases?

While we can always create our own blast databases with `makeblastdb` , this task may take a considerable amount of time (days or even weeks) when the total number of bases to be indexed is very large.

To facilitate the process, prebuilt databases can be downloaded from the NCBI webpage.

ftp://ftp.ncbi.nlm.nih.gov/blast/db/

## Can we automate the download of BLAST databases?

NCBI offers ready-made databases for many organisms and even the entire non-redundant database of all known sequences is available for download.

Make a dedicated storage for your blast databases:

```
# This will store all downloaded databases.
mkdir -p ~/refs/refseq

# Switch to that folder.
cd ~/refs/refseq/

# The blast package includes a script that can be used to download
# ready made databases. List them all:
update_blastdb.pl --showall

# Download the 16 microbial database.
update_blastdb.pl 16SMicrobial --decompress

# Download the taxonomy database. This can be helpful later.
update_blastdb.pl taxdb --decompress
```

We can also tell blast where to look for files. Add this to your bash profile like other variables. Then you would not need to add the full path to the BLAST database. In addition, this is required for the taxonomy searches to work.

```
# Add this to your profile.
export BLASTDB=$BLASTDB:~/refs/refseq/

# If the above is set then you can just type:
blastdbcmd -db 16SMicrobial -info
```

The taxonomy information will be automatically accessed by BLAST.

# What information can be obtained from a BLAST database?

We can extract a wide variety of interesting information from blast databases. Sometimes it is worth creating a BLAST database just to extract specific information from sequences.

```
# Get some information on the 16S Microbial database.
blastdbcmd -db ~/refs/refseq/16SMicrobial -info

# What is the first sequence in the file
blastdbcmd -db ~/refs/refseq/16SMicrobial -entry 'all' | head

# Which publication links to  this sequence?
esearch -db nuccore -query NR_118889.1 | elink -target pubmed | efetch
```

`blastdbcmd` is a command with high utility. Read the help on it:

```
blastdbcmd -help | more
```

# How do I reformat sequences in a BLAST database?

The `-outfmt` parameter of the `blastdbcmd` allows printing a variety of information. For example, the following will print the sequence ids and their length:

```
blastdbcmd -db ~/refs/refseq/16SMicrobial -entry 'all' -outfmt '%a %l'
```

There are many more formatting options (see the detailed help for the tool) that can be used to answer potentially interesting questions:

We replace the ',' (comma) separator with tabs since it is easier to manipulate files with tabs than with commas. The `tr` command translates each comma into a tab

```
blastdbcmd -db ~/refs/refseq/16SMicrobial -entry 'all' -outfmt '%a,%l,%T,%L' | tr ',' '\t' > 1
6genes.txt
```

This file lists the genes, lengths, and common taxonomical ranks (this last part will only work if your taxonomy file has been downloaded as above and if the `BLASTDB` variable has been set properly).

```
NR_118889.1    1300    36819    Amycolatopsis azurea
NR_118899.1    1367    1658    Actinomyces bovis
NR_074334.1    1492    224325    Archaeoglobus fulgidus DSM 4304
NR_118873.1    1492    224325    Archaeoglobus fulgidus DSM 4304
NR_119237.1    1492    224325    Archaeoglobus fulgidus DSM 4304
NR_118890.1    1331    1816    Actinokineospora fastidiosa
NR_044838.1    1454    1381    Atopobium minutum
NR_118908.1    1343    1068978    Amycolatopsis methanolica 239
NR_118900.1    1376    1655    Actinomyces naeslundii
NR_044822.1    1255    2075    Pseudonocardia nitrificans
```

What organism has the longest 16S gene?

```
cat 16genes.txt | sort -k2,2 -rn | head
```

This sorts the file above by the second column:

```
NG_046384.1    3600    1104324    Pyrobaculum sp. 1860
NG_041958.1    2211    178306    Pyrobaculum aerophilum str. IM2
NG_041961.1    2207    121277    Pyrobaculum arsenaticum
NG_042068.1    2197    56636    Aeropyrum pernix
NG_042067.1    2192    70771    Pyrobaculum neutrophilum
NG_041957.1    2192    698757    Pyrobaculum oguniense TE7
NG_041951.1    2168    698757    Pyrobaculum oguniense TE7
NG_041954.1    2130    70771    Pyrobaculum neutrophilum
NG_044969.1    2089    477696    Thermogladius shockii
NG_042881.1    1833    215    Helicobacter fennelliae
```

# How do I extract a specific entry from a database?

Pass the `-entry accession` parameter to `blastdbcmd`. Here, the particular blast database construction choices become essential, specifically whether or not the `-parse_seqids` was used. Accession numbers are only available in the latter case. Otherwise, the full sequence name needs to be used.

Get the first 20 bases of a specific 16S gene. We are asking for one specific accession number:

```
blastdbcmd -db ~/refs/refseq/16SMicrobial -entry 'NR_118889.1' -range 1-20
```

This produces the output:

```
>gi|645322056|ref|NR_118889.1|:1-20 Amycolatopsis azurea strain NRRL 11412 16S ribosomal RNA g
ene, partial sequence
GGTCTNATACCGGATATAAC
```

We can format the output for it as well.

```
blastdbcmd -db ~/refs/refseq/16SMicrobial -entry 'NR_118889.1' -range 1-20 -outfmt "%s"
```

This produces the following output:

```
GGTCTNATACCGGATATAAC
```

# How do I process all entries in a database?

Pass the `-entry all` parameter to `blastdbcmd` to get, for example, the first 50 bases of each gene.

```
blastdbcmd -db ~/refs/refseq/16SMicrobial -entry 'all' -range 1-50 -outfmt "%s"  > starts.txt

cat starts.txt | head
```

This will produce a file containing the first 50 bases of each 16S gene, like so:

```
GGTCTNATACCGGATATAACAACTCATGGCATGGTTGGTAGTGGAAAGCT
GGGTGAGTAACACGTGAGTAACCTGCCCCNNACTTCTGGATAACCGCTTG
ATTCTGGTTGATCCTGCCAGAGGCCGCTGCTATCCGGCTGGGACTAAGCC
ATTCTGGTTGATCCTGCCAGAGGCCGCTGCTATCCGGCTGGGACTAAGCC
ATTCTGGTTGATCCTGCCAGAGGCCGCTGCTATCCGGCTGGGACTAAGCC
TACTTTGGGATAAGCCTGGGAAACTGGGTCTNATACCGGATATGACAACT
TTGAACGGAGAGTTCGANCCTGGCTCAGGATGAACGCTGGCGGCGCGCCT
GTGAGTGGCGAACGGGTGAGTAACACGTGGGTAACCTTCNNTGTACTTTG
GTGAGTAACCTGCCCCTTCTTCTGGATAACCGCATGAAAGTGTGGCTAAT
ACCGGATANGACCACTNATCGCATNTCGGTGGGTGGAAAGTTTTTTCGGT
```

How many unique gene starts of length 50 are there?

```
cat starts.txt | sort | uniq -c | sort -rn | head
```

The output of the above is:

```
298 AGAGTTTGATCCTGGCTCAGGACGAACGCTGGCGGCGTGCTTAACACATG
161 AGAGTTTGATCCTGGCTCAGGACGAACGCTGGCGGCGTGCCTAATACATG
126 GACGAACGCTGGCGGCGTGCTTAACACATGCAAGTCGAGCGGTAAGGCCC
102 ATTCCGGTTGATCCTGCCGGAGGCCATTGCTATCGGAGTCCGATTTAGCC
101 AGAGTTTGATCATGGCTCAGATTGAACGCTGGCGGCAGGCCTAACACATG
 91 GACGAACGCTGGCGGCGTGCTTAACACATGCAAGTCGAGCGGAAAGGCCC
 84 AGAGTTTGATCCTGGCTCAGGACGAACGCTGGCGGCGTGCCTAACACATG
 75 AGAGTTTGATCCTGGCTCAGAACGAACGCTGGCGGCAGGCTTAACACATG
 72 TAGAGTTTGATCCTGGCTCAGGACGAACGCTGGCGGCGTGCTTAACACAT
 69 AGAGTTTGATCCTGGCTCAGATTGAACGCTGGCGGCAGGCCTAACACATG
```

# BLAST use cases

## What are typical uses of BLAST?

Since blast contains such a wide variety of tools, the uses are extremely varied. A large subset of them, however, could be described simply as searching through a very large database for changes relative to a known sequence.

## What is a realistic use case of BLAST?

Below, we will walk through a solution to the following problem:

- How does the sequence of the polymerase genes in the many strains observed during the 2014 outbreak compare to that observed during the 1976 outbreak. The data from 1976 is deposited under the accession number `AF086833`

Get the ebola genome for the 1976 outbreak in both Genbank and FASTA formats.

```
efetch -db nuccore -id AF086833 -format gb  > AF086833.gb
efetch -db nuccore -id AF086833 -format fasta  > AF086833.fa
```

First, we need to get the sequence for the polymerase gene. Perhaps we already know its accession number, or we could search NCBI, but we could also find that out from the genome information.

The `extractfeat` tool has many (very obscure and hard to use) options that we could use. Alas, the documentation for this tool (and the EMBOSS tools in general) are extremely lacking, and the best way to learn is by seeing examples of its usage. We could do the following to extract certain feature names and ids:

```
cat AF086833.gb | extractfeat -filter  -describe 'gene|protein_id|product' | grep polymerase
```

Or, we could get the actual sequence in one shot that involves both `extractseq` and `transeq`:

```
cat AF086833.gb |  extractfeat -filter -type CDS -tag gene -value L | transeq -filter > L.fa
```

But there are other, simpler ways to find this information. A brute force approach that typically works well is look for its accession number within this Genbank file.

```
# -A and -B makes grep print lines before and after a match.
cat AF086833.gb | grep -B 6 -A 6 polymerase
```

From which we can see the accession number `AAD14589.1` for the polymerase gene. Then we can just do:

```
efetch -db protein -id  AAD14589 -format fasta  >  L.fa
```

As it happens, the two sequences obtained in different ways (extracted and downloaded) are not exactly the same: one contains a stop codon `*` at the end of the sequence and the other does not. BLAST will ignore and work in the presence of the stop codon `*`. Other tools may not operate the same way.

Obtain all the proteins submitted for the 2014 Ebola project `PRJNA257197`. Place the results in a different directory so that blast index files don't get in the way.

```
mkdir -p index
esearch -db protein -query PRJNA257197 | efetch -format fasta > index/all-proteins.fa

# How many sequences match the word polymerase?
infoseq index/all-proteins.fa | grep polymerase | wc -l

# Build the blast index.
makeblastdb -in index/all-proteins.fa -dbtype prot -out index/all -parse_seqids

# Run the blast queries.
blastp -db index/all -query L.fa

# Customize the output even more.
blastp -db index/all -query L.fa -outfmt "7 qseqid sseqid pident"

# Three best alignments
blastp -db index/all -query L.fa -outfmt "6 qseqid sseqid pident" | sort -k3 -rn | head -5

# The worst alignments.
blastp -db index/all -query L.fa -outfmt "6 qseqid sseqid pident" | sort -k3 -rn | tail -5
```

Investigate the last alignment a little better (we can see its accession number `AKC37152.1`):

```
# Get the sequence
efetch -db protein -id AKC37152 -format fasta > AKC37152.fa

# Align it against L.fa
blastp -query AKC37152.fa -subject L.fa
```

# Accelerated BLAST-like alignment tools.

## When would I need a faster BLASTX like tool?

When a data set consists of millions of reads and its taxonomic or functional content needs to be analyzed, the usual approach is to align these sequences using BLASTX. Over the years faster alternatives to BLASTX have been created.

## What tools can be used instead of BLASTX?

Faster alternatives to BLASTX which can produce BLAST-like alignments include

- Diamond - Tool for aligning short DNA reads to a protein database like NR with improved run time and similar sensitivity to BLASTX.
- RAPSearch2 - A memory-efficient implementation of RAPSearch algorithm for protein similarity search with a large database and a large queryset.
- MALT - Sequence alignment tool for high-throughput data against databases like NR, Silva etc. - designed to provide input for MEGAN6.

## What type of information is needed to run Diamond?

To run an alignment task with Diamond, one would need

- A protein database file in FASTA format.
- A FASTA file of sequence reads that we want to align.

## How do I run Diamond?

The first step is to set up a reference database for Diamond. `makedb` command needs to be executed for this. The following command creates a binary Diamond database for NR data.

```
diamond makedb --in nr.faa -d nr
```

The next step is to align the reads to this database.

```
diamond blastx -d nr -q reads.fa -o align.txt -f 6
```

`-f,--format` specifies the output format to be BLAST tabular.

If the input file contains protein data and we want to run a blastp, the command is

```
diamond blastp -d nr -q prot.fa -o align.txt -f 6
```

## What output formats are available in Diamond?

Diamond can produce BLAST-tabular, BLAST XML or SAM alignment outputs. The BLAST-tabular output can be configured with keywords just like in NCBI-BLAST. It can also produce Diamond alignment archive (DAA) which can be an input to programs like MEGAN.

## What is the e-value threshold used in Diamond?

Evalue threshold can be set with `-evalue` parameter. The default e-value cutoff of DIAMOND is 0.001 while that of BLAST is 10.

## How long does a Diamond run take?

When used with 24 CPUs in threaded mode the index for NCBI-NR database could be created in just 11 minutes. It took 45 minutes to align 470,000 reads with a maximum length of 300 base pairs to this database.

# Short Read Aligners

Sequencing used to be a very expensive proposition. The resulting few, and relatively long, sequences (1000 bp) were very "precious" and were meant to be studied extensively. For that reason, aligners were initially designed to perform near-optimal alignments and to enumerate most if not all alternative arrangements that all fell within a level tolerance. The landscape changed dramatically around 2005 with the arrival of the so-called high throughput short-read technologies.

## How are short reads different from long reads?

Modern sequencing instruments brought about two significant changes:

1. The read lengths became very short (50-300bp)
2. The number of reads became extremely high (hundreds of millions)

Hence, whereas initial algorithms were designed to find most (if not all) alignments, scientists needed newer and faster methods that could very quickly select the best location for each of the measurements at the expense of not investigating all potential alternative alignments.

The rationale behind the new paradigm was to treat sequencing reads as small, independent measurements to be subsequently matched either against a known genome (called re-sequencing) or against one another (called assembly). This new use case and its requirements lead to the development of an entire subfield of alignment algorithms, often referred to as "short read aligners" or "short read mappers."

## What are short read aligners?

Short read aligners are commonly used software tools in bioinformatics, designed to align a very large number of short reads (billions). The dominance of the Illumina instrumentation typically means that most short-read software will work in the range that this instrument produces its data (meaning in the range of 50-300bp).

## How do short read aligners work?

First, let us make something abundantly clear: short read aligners are a marvel of modern scientific software engineering. The field of Bioinformatics is lucky to have so many choices of software at this level of complexity and performance.

A well-optimized short read aligner can match over ten thousand sequences per second (!) against the 3 billion bases of the human genome. Their performance, implementation, and level of execution is nothing short of astounding.

But there is another side to this complexity. The rational expectation is that the published aligners will produce reasonably similar alignments and that the difference between them will manifest primarily in their performance, hardware requirements, and small fluctuations in accuracy and precision.

In reality, the results that two aligners produce can be substantially different. Surprisingly, it seems almost impossible to unambiguously understand, describe, and characterize the difference in results that various algorithm's implementations provide. While there is no shortage of scientific publications that claim to compare the accuracy and performance of various tools, most of these papers fail to capture the essential differences between the methods. It is not for lack of trying - accurately characterizing the "quality" of alignments is a lot more complicated that we had anticipated.

A typical bioinformatics analysis requires finding exact matches, missing locations, and partial and alternative matches/overlaps. A tool that is efficient at one of these tasks trades that performance gain for inefficiency in another requirement.

Hence, the performance of software methods depend critically on the type of data they are used to analyze, which, in turn, is domain-specific. Different alignment methods will, therefore, be best suited to different domains.

# What are the limitations of short read aligners?

1. Most short read aligners will find only alignments that are reasonably similar to the target. This means that the algorithm "gives up" searching beyond a certain threshold.
2. Short read aligners are designed to find regions of high similarity.
3. Most short read aligners typically cannot handle long reads or become inefficient when doing so.
4. There is also a limit to how short a read can be.

The minimum length for the read is algorithm- and implementation-dependent; it is rarely (if ever) mentioned in the documentation. Many tools stop working properly when the read lengths drop under 30 bases. When studying small RNAs, for example, microRNAs, we have to look for tools that can align very short reads; the selection is surprisingly sparse.

# What is read mapping?

Conceptually, alignment and mapping appear to mean the same thing, but there are subtle differences. The word "mapper" is often used to emphasize that the optimal alignment of a read is not guaranteed. The purpose of a mapper tool is locating a region in a genome, not producing an optimal alignment to that region.

The following definition is taken from a presentation by Heng Li:

> Mapping
>
> - A mapping is a region where a read sequence is placed.
> - A mapping is regarded to be correct if it overlaps the true region.
>
> Alignment
>
> - An alignment is the detailed placement of each base in a read.
> - An alignment is regarded to be correct if each base is placed correctly.

Interestingly, and we invite the reader to come up with their own examples, we could have a situation where a read has a correct mapping with an incorrect alignment or vice versa - has an incorrect mapping with a proper alignment.

Most of the time, of course, the two match —but it's important to remember that this is not always the case.

# How do I distinguish between a mapper and an aligner?

It used to be that tools could be readily categorized into the two classes of aligner vs. mapper. As the field advanced and matured, however, the read lengths became longer and the software tools started to make use of combined techniques that made some tools behave as both mappers and aligners.

It is important, however, to remember that the distinction between mapping and alignment does exist, and to recognize further that different studies have different requirements in regards to the use of the two tools. You will need to grasp the difference between mapping and alignment to judge which is most applicable to your study.

For example, studies examining SNPs and variations in a genome would be primarily alignment-oriented. By contrast, studies focusing on RNA-Seq would be essentially mapping-oriented. The reason for this difference is that, in the latter case, you would need to know where the measurements come from, but you would be less concerned about their proper alignment.

# How do we pick the best short read aligner?

So how do we pick the best aligner? There are good starting choices, such as `bwa` and `bowtie2`, that will most likely perform well. Later, we may pick other tools based on observed performance of the chosen aligner, reviews of scientific literature on the performance of various aligner tools, and the experience of other scientists studying the same type of data. Remember, this is always specific to the data you are analyzing.

The choice of the aligner is a bit "faith-based," coupled to a combination of personal observations and empirical evidence. There is also a cultural and historical element to it. Scientists at the Broad Institute will likely use `bwa` because it was developed there; others who work at the Beijing Genomics Institute will probably prefer the `novoalign` tool because as it was created at that institute, and so on.

There is a strong, individualistic feel to almost every high-throughput aligner. Typically, a sole genius is behind each such tool, an individual with uncommon and extraordinary programming skills that has set out to solve a particular problem important to them. Due to the sheer complexity of the problem and required feature set, the software's unique characteristics will reflect their personality, virtues, and some of their personal quirks, as well.

You don't just run an aligner; you perform a Vulcan mind-meld (see Star Trek) with the author who wrote it.

# What features do we look for in a short read aligner?

When choosing an aligner, we need to perform a mental checklist of what features the aligner of our choice should provide:

- Alignment algorithm: global, local, or semi-global?
- Is there a need to report non-linear arrangements?
- How will the aligner handle INDELs (insertions/deletions)?
- Can the aligner skip (or splice) over large regions?
- Can the aligner filter alignments to suit our needs?
- Will the aligner find chimeric alignments?

Looking at this list above, it is probably clear what makes comparing different tools so difficult. Various techniques excel for different requirements; moreover, as we shall see, we are only scratching the surface when it comes to the information that alignment record may contain.

# The bwa aligner

The `bwa` (aka Burrows-Wheelers Aligner) aligner was written by Heng Li, a research scientist at the Broad Institute. This aligner is possibly the most widely used short read alignment tool and is well suited for a broad number of applications.

Interestingly, the algorithm that the latest version of `bwa` implements has not yet been published in a peer reviewed journal (as of 2016). There is a fascinating story of why that is so, and it illustrates the shortcomings of the so-called "scientific method" itself.

In bioinformatics, a tool's usage may change in time, often radically. This is applicable to `bwa`, as well. New releases of the same tool implement a different, more general alignment algorithm. The new algorithm is the so-called `bwa mem` algorithm (where `mem` stands for Maximally Exact Matches).

But note that there may be quite a bit of obsolete documentation on the previous alignment method called `bwa aln`. In the following, we will focus solely on the `bwa mem` algorithm, the results of which will differ (often substantially) from the results that of the `bwa aln` algorithm. See the pitfalls of comparing tools? Even the same software may in time choose to implement different algorithms that may produce very different results.

# How does bwa work?

In general, all short read aligners operate on the same principles:

1. They first need to build an index from a known reference (this only needs to be done once).
2. The reads in FASTA/FASTQ files are then aligned against this index.

Index building consists of simply preparing and reformatting the reference genome so that the program can search it efficiently. Each program will build a different type of index; sometimes it may produce multiple files with odd looking names or extensions. For this reason, it is best to place the reference genome in a separate folder and store the indices there as well.

Depending on the program, the original FASTA file of the reference genome may need to be kept at the original location (so don't remove the FASTA file). The time and computational resources required for building an index will depend on the genome size. These can be substantial (many days or more) for large genomes. For some tools it is possible to download pre-built indices from their websites.

# How do I use bwa?

We will align ebola sequencing data against the 1976 Mayinga reference genome. We recommend to keep reference genomes in a more general location rather than the local folder since we will reuse them frequently. For example, use `~/refs/ebola` for all ebola genomes and annotations.

This directory will hold the reference genome and all indices:

```
mkdir -p ~/refs/ebola
```

Get the ebola genome in FASTA format. We will rename it as `1976.fa`. Usually meaningful and readable names help quite a bit.

```
efetch -db=nuccore -format=fasta -id=AF086833 > ~/refs/ebola/1976.fa
```

Build an index with `bwa`:

```
bwa index ~/refs/ebola/1976.fa
```

This will finish rather quickly, as the genome is a mere 19K base long. We can make use of environment variables to store names that are to be reused. This is handy, as it allows us to keep the command the same and only change the value of the variable. For example, we could have written,

```
# Assign the file name to a variable
REF=~/refs/ebola/1976.fa

# Build the index
bwa index $REF

# This process created quite a few files.
ls ~/refs/ebola
```

The `bwa` program provides a neat self-discovery of its parameters. It is run as `bwa command`, and it will perform very different tasks depending on which command is used. To discover all of its commands, run it just as

```
bwa
```

Then we can find out more on each command, like so:

```
bwa index
bwa mem
```

and note the instructions that it provides. To align reads, we need data files. Let's obtain data from SRA. Find out all runs for the Ebola project.

```
esearch -db sra -query PRJNA257197  | efetch -format runinfo > runinfo.csv
```

We can pick a run from this file, say `SRR1972739`, and we'll only subset the data to 10K reads to get through quickly:

```
fastq-dump -X 10000 --split-files SRR1972739
```

We can create shortcuts to each of our files. This will allow us to shorten and simplify the commands greatly.

```
R1=SRR1972739_1.fastq
R2=SRR1972739_2.fastq
```

You can always check what the variable stands for with,

```
echo $R1
```

To align one of the read pairs with `bwa mem` as single end, we could run:

```
bwa mem $REF $R1 > output.sam
```

The resulting file is in a so-called SAM (Sequence Alignment Map) format, one that you will undoubtedly love and hate (all at the same time). It is one of the most recent bioinformatics data formats, one that by today has become the standard method to store and represent all high-throughput sequencing results. It looks like this:

```
@SQ    SN:gi|10141003|gb|AF086833.2|    LN:18959
@PG    ID:bwa    PN:bwa    VN:0.7.12-r1039    CL:bwa mem /Users/ialbert/refs/ebola/1976.fa SRR
1972739_1.fastq SRR1972739_2.fastq
SRR1972739.1    83    gi|10141003|gb|AF086833.2|    15684    60    69M32S    =    15600    -15
3    TTTAGATTTAACAAGATACCGAGAAAATGAATTGATTTATGACAATAATCCTCTAAAAGGAGGACTCAAATGAGATATTGCAATTGAGT
CCTCCTTTTAGA    DDDDDEEEEEDEEFFFEDHHHHIJJJJJJJJJJJJJJJJJJIGIJJJJJJJJJJJJJJJJJJJIGIGJJJJJJJJJJJJ
JJIJJJJJJIFHHHHFFFFFCCC    NM:i:2    MD:Z:27C16G24    AS:i:59    XS:i:0    SA:Z:gi|10141003|gb|
AF086833.2|,15735,+,33M68S,60,0;
```

A SAM file encompasses all known information about the sample and its alignment; typically, we never look at the FastQ file again, since the SAM format contains all (well almost all) information that was also present in the FastQ measurements. We will have an entire chapter on SAM formats later.

Since this is a paired end dataset, the proper way to align the data is in paired end mode. For that, we list both files on the command line:

```
bwa mem $REF $R1 $R2 > bwa.sam
```

Note how the same data can be aligned in either single end or paired end mode though this latter alignment will contain more information as pair related information will be added to the alignment file.

# How do I find help on bwa?

To see all the ways that we could tune `bwa mem` alignment run

```
bwa mem
```

Among the many options, note those that set the scoring matrix:

```
-A INT        score for a sequence match, [...] [1]
-B INT        penalty for a mismatch [4]
-O INT[,INT]  gap open penalties for deletions and insertions [6,6]
-E INT[,INT]  gap extension penalty; a gap of size k cost '{-O} + {-E}*k' [1,1]
-L INT[,INT]  penalty for 5'- and 3'-end clipping [5,5]
-U INT        penalty for an unpaired read pair [17]
...
```

When compared to the more traditional DNAFULL matrix discussed in the alignment chapter, we note that in the default setting, the reward for sequence match is much smaller; hence, the default alignment is tuned to find long streches if bases match.

Gap open penalties are also smaller (6 instead of 10) whereas gap extension penalities are a bit larger.

There are other parameters where a single word will set multiple values. For example:

- note how `-x ont2d` is equivalent to setting `-k14 -W20 -r10 -A1 -B1 -O1 -E1 -L0`. These are the recommended settings when aligning data from an Oxford Nanopore MinION instrument.

The way the parameters alter the final alignment gets complicated. Here is where the choice becomes more "faith based". We trust the author of the tool. In this case, we trust that Heng Li knows why these parameters are the best. Even if we ourselves could not fully explain their interplay, we have reason to trust them. What is important to keep in mind is that decisions may have been made on our behalf and if things don't go the way we expect, we know where to look for potential solutions.

# The bowtie aligner

The bowtie aligner was the first implementation of the Burrows-Wheeler algorithm for short read alignment, and it has opened a new era in processing high-throughput data.

There are two versions of `bowtie`. The latest version, `bowtie2`, is typically preferred because, as an algorithm, it is superior to `bowtie1`.

# How does bowtie work?

In general, all short read aligners operate on the same principles:

1. They first need to build an index from a known reference (this only needs to be done once).
2. The reads in FASTA/FASTQ format are then aligned against this index.

# How do I build an index with bowtie?

We will align ebola sequencing data against the 1976 Mayinga reference genome. We recommend keeping reference genomes in a more general location rather than the local folder, since you will reuse them frequently. For example, use `~/refs/ebola` for all ebola genomes and annotations.

This directory will hold the reference genome and all indices:

```
mkdir -p ~/refs/ebola
```

Get the ebola genome in FASTA format. We will rename it as `1976.fa`. Meaningful and readable names help quite a bit. We can make use of environment variables to store names that are to be reused. This is handy as it allows us to keep the command the same and only change the value of the variable. For example, we could have written,

```
# Assign the file name to a variable
REF=~/refs/ebola/1976.fa

# Obtain the genome.
efetch -db=nuccore -format=fasta -id=AF086833 > $REF
```

For bowtie2, we need to specify both the input and output prefix name on the command line. Confusingly, these can also be the same because many prefixes are the same. The first `$REF` is the reference file; the second `$REF` indicates the prefix name of the index; it could be anything, but it is a good habit to keep it the same or similar to the actual reference.

```
bowtie2-build $REF $REF
```

See for yourself how the ancillary files multiply like rabbits.

```
ls $REF.*
```

# How do I align with bowtie?

We can pick an ebola sequencing run id, for example, `SRR1972739`, and we'll select just 10K reads to get through quickly:

```
fastq-dump -X 10000 --split-files SRR1972739
```

We can create shortcuts to each of our files. This will allow us to shorten and simplify the commands greatly.

```
R1=SRR1972739_1.fastq
R2=SRR1972739_2.fastq
```

To align with bowtie,

```
bowtie2 -x $REF -1 $R1 -2 $R2  > bowtie.sam
```

When bowtie runs, it also gives us a report:

```
10000 reads; of these:
  10000 (100.00%) were paired; of these:
    5086 (50.86%) aligned concordantly 0 times
    4914 (49.14%) aligned concordantly exactly 1 time
    0 (0.00%) aligned concordantly >1 times
    ----
    5086 pairs aligned concordantly 0 times; of these:
      827 (16.26%) aligned discordantly 1 time
    ----
    4259 pairs aligned 0 times concordantly or discordantly; of these:
      8518 mates make up the pairs; of these:
        7463 (87.61%) aligned 0 times
        1055 (12.39%) aligned exactly 1 time
        0 (0.00%) aligned >1 times
62.69% overall alignment rate
```

Not to be outdone in the shock and awe department, bowtie2, too, can be set up in myriad ways.

```
bowtie2
```

Run and enjoy the view:

```
[ ... lots of lines ... ]
Presets:                 Same as:
  For --end-to-end:
   --very-fast            -D 5 -R 1 -N 0 -L 22 -i S,0,2.50
   --fast                 -D 10 -R 2 -N 0 -L 22 -i S,0,2.50
   --sensitive            -D 15 -R 2 -N 0 -L 22 -i S,1,1.15 (default)
   --very-sensitive       -D 20 -R 3 -N 0 -L 20 -i S,1,0.50

  For --local:
   --very-fast-local      -D 5 -R 1 -N 0 -L 25 -i S,1,2.00
   --fast-local           -D 10 -R 2 -N 0 -L 22 -i S,1,1.75
   --sensitive-local      -D 15 -R 2 -N 0 -L 20 -i S,1,0.75 (default)
   --very-sensitive-local -D 20 -R 3 -N 0 -L 20 -i S,1,0.50

[ ... lots of lines ... ]
```

# How do I compare aligners?

## How can I tell which aligner is "better"?

The answer to this is a little complicated. As you will see, we can't quite make it a "fair" comparison to begin with. Let's try to evaluate the alignment on simulated sequences:

```
# Get the reference genome and build an index from it.
mkdir -p refs
REF=refs/AF086833.fa
efetch -db=nuccore -format=fasta -id=AF086833 > $REF

# Index the reference with bwa
bwa index $REF
# Index the reference with bowtie2.
bowtie2-build $REF $REF

# Simulate 100,000 reads from the genome with 1% error rate
dwgsim -N 100000 -e 0.01 -E 0.01 $REF data

# Set up shortcuts to the reads.
R1=data.bwa.read1.fastq
R2=data.bwa.read2.fastq
```

Align with `bwa mem` while timing the run:

```
time bwa mem $REF $R1 $R2 > bwa.sam
```

Align with `bowtie2` :

```
time bowtie2 -x $REF -1 $R1 -2 $R2  > bowtie.sam
```

The result is that:

- bwa aligns `95.4%` of reads in `4` seconds
- bowtie2 aligns `94.7%` of reads in `10` seconds.

So bowtie2 was slower and a hair less efficient. Now let's raise the error rate to 10%:

```
dwgsim -N 100000 -e 0.1 -E 0.1 $REF data
```

Running the same alignments now take longer and fewer reads align:

- bwa aligns `83.3%` of reads in `6` seconds
- bowtie2 aligns `28.9%` of reads in `3` seconds.

The discrepancy is now huge. `bowtie2` is way off the mark. The first lesson to learn here is that properties of the input data may have major effects on both runtime and mapping rates that you may observe. We can tune `bowtie2` and make it more sensitive:

```
bowtie2 --very-sensitive-local -x $REF -1 $R1 -2 $R2  > bowtie.sam
```

In this case

- bwa aligns `83.3%` of reads in `6` seconds
- bowtie2 aligns `63.86%` of reads in `11` seconds.

Maybe we could tune bowtie2 even more. Hmm ... how about this for tuning:

```
time bowtie2 -D 20 -R 3 -N 1 -L 20 -x $REF -1 $R1 -2 $R2  > bowtie.sam
```

We are getting somewhere; bowtie2 now aligns more reads than bwa:

- bwa aligns `83.3%` of reads in `6` seconds
- bowtie2 aligns `87.14%` of reads in `10` seconds.

As you can see, it is not that simple to compare aligners. A lot can depend on how aligners are configured - though all things considered we like to start with and recommend `bwa` as clearly it is a lot less finicky and gets the job done without adding what seem to be elaborate decorations.

But does it mean that `bowtie2` is a second class citizen? Well, lets not rush to judgment yet. Let's see what tags are present in the bwa alignment file:

```
cat bwa.sam | cut -f 12-20 | head
```

prints:

```
AS:i:0  XS:i:0
AS:i:0  XS:i:0
NM:i:6  MD:Z:2A12G5A14G6G23A2   AS:i:44 XS:i:0
NM:i:6  MD:Z:7G8G32A8T0T3A6     AS:i:43 XS:i:0
```

the alignments made with bowtie2

```
cat bowtie.sam | cut -f 12-20 | head
```

contain:

```
AS:i:-29        XN:i:0  XM:i:10 XO:i:0  XG:i:0  NM:i:10 MD:Z:0C0A6A17C4T7T1A6G1T13A5    YS:i:-
27      YT:Z:CP
AS:i:-27        XN:i:0  XM:i:10 XO:i:0  XG:i:0  NM:i:10 MD:Z:2C8A8C5T1T5A6T14T3C2T6     YS:i:-
29      YT:Z:CP
AS:i:-18        XN:i:0  XM:i:6  XO:i:0  XG:i:0  NM:i:6  MD:Z:2A12G5A14G6G23A2   YS:i:-16
  YT:Z:CP
AS:i:-16        XN:i:0  XM:i:6  XO:i:0  XG:i:0  NM:i:6  MD:Z:7G8G32A8T0T3A6     YS:i:-18
  YT:Z:CP
```

Do you see the difference? Bowtie2 fills in a lot more information, it also it offers more ways to format the output and filter the alignments. So it has its applications and may be indispensable in some situations.

By the way, do you want to know how we figured out that it should be `-D 20 -R 3 -N 1 -L 20` and what does that even mean?

If you must know the task of finding settings where for this simulation bowtie2 outperforms bwa was a homework assignment. The help for bowtie2 lists the parameters that are set when `--very-sensitive-local` is set, so students started with those and kept tweaking and re-running the alignments. It was a brute force search, not some deep insight into what these parameters mean and how they interact. Few people (if anyone) understands these at that level of granularity.

# How do I choose the right aligner?

The most important lesson is not to treat the tools as "better" or "worse," but instead as more or less appropriate for a given task. Also, it is essential to understand that not all tools operate at the same levels of sensitivity or specificity when using them on their default setting.

Finally note that, depending on the data and its characteristics, different tools may be better suited to different types of data. The requirement to have a particular kind of information in the BAM file may also be a significant factor. The good news is that it is easy to benchmark different aligners as needed. As we've seen conceptually, the tools are very similar.

# Sequence Alignment Maps

- The SAM format specification is the official specification of the SAM format.
- The [SAM tag specification][samtags] is the official specification of the SAM tags.

## What is a SAM file?

The SAM format is a TAB-delimited, line oriented text format consisting of a

1. **Header** section, where each line contains some metadata
2. **Alignment** section where each line contains information on an alignment

The SAM format specification lists the required and optional content for each of these sections.

## What is a BAM file?

A BAM file is a a binary, compressed (and almost always sorted) representation of the SAM information. BAM files are always sorted, usually by the alignment coordinate information and more rarely by the read names.

- Sorting by coordinate allows fast query on the information by location.
- Sorting by read name is required when both read pairs (that are identically named) have to be accessed very quickly since these will then be stored in adjacent lines.

## What is the CRAM format?

The file format was designed and released in 2014 by the European Bioinformatics Institute see CRAM goes mainline

CRAM files are conceptually similar to BAM files. They represent a compressed version, where some of the compression is driven by compressing the reference genome that the sequence data is aligned to. This makes the data smaller than the BAM compressed formats.

The `CRAM` format is supported directly by the `samtools sort`:

```
bwa mem $REF $R1 $R2 | samtools sort --reference $REF -O CRAM > bwa.cram
```

The downside of this approach is that losing the reference sequence means losing the data (that is, being unable to decode it). For many cases this is not a problem. But there are situations where having the alignment data dependent on the presence and availability of the reference genome adds another layer of complexity.

A CRAM file can only be decoded if the same reference sequence is present at a given location on the target computer.

# Will the CRAM format replace BAM?

Time will tell. The added complexity of having to manage the reference files may cause some setbacks. On the other hand there is a strong concerted effort towards migrating all data to CRAM. Some projects are now releasing their data in CRAM format only.

# Is it SAM, BAM or CRAM now?

The format that the data is in is called SAM. When we store it in a binary compressed form we call it BAM or CRAM. Most of the time we exchange the data in a binary format because it is much more efficient and faster to query.

> Note that when we view or query a BAM file with a tool like `samtools` we see the results converted into the SAM format.

We will call the format and the data as SAM with the understanding that the actual representation may be BAM or CRAM.

# What are SAM files used for?

SAM files were invented to allow the representation of hundreds of millions of short alignments. Most data analytics protocols are methods for processing the information in SAM/BAM files.

While analyzing these SAM files we often end up being far removed from the representation itself, so much so that it almost feels irrelevant what format the data was in. But make no mistake, the information within this file determines the success of any analysis. Hence, producing this file so that it actually contains the information we need is and should always be our most pressing concern.

# What is stored in a SAM file?

Even practicing bioinformaticians may have misconceptions about SAM files. Among them the most prevalent is the assumption that BAM files produced by different tools will have a comparable amount of information and the differences are mostly in the *accuracy* or *performance* characteristics of the tool.

This misconception is further reinforced by the existence of the SAM specification itself. This specification is the "standard" and that seems to prescribe what goes into each SAM file: each SAM file has to have 11 columns and each column contains a specific type of information. It is now understandable why many will assume that SAM files produced by different tools ought to contain very similar information.

As it turns out there are alway substantial, relevant and essential differences between alignment information produced by different tools, and these differences can be summarized as:

1. Which alignments are (or are not) reported in the SAM file.
2. Alignment information beyond the required fields.

We'll mention here that attempting to transform one "standard" SAM file into another "standard" SAM file that contains all the information that another tool might have reported is a surprisingly challenging (and most likely impossible) task - often the only recourse is to actually use this other tool.

The take-home message is that a SAM format is very "stingy". It prescribes the presence of very little information. Each aligner will fill in as much as it knows how to.

# Why do we need the SAM format?

The SAM format was introduced to support use cases presented by the high throughput sequencing instrumentation:

- Fast access to alignments that overlap with a coordinate. For example, select alignments that overlap with coordinate `323,567,334` on chromosome `2`
- Easy selection and filtering of reads based on attributes. For example, we want to be able to quickly select alignments that align on the reverse strand.
- Efficient storage and distribution of the data. For example, have a single compressed file contain data for all samples each labeled in some manner.

# Can "unaligned" data be stored in a SAM file?

Because the BAM files are compact and can store additional information they have become a de-facto standard not only for representing and storing alignment information, but also for storing raw "unaligned" FASTQ files.

It is a somewhat idiosyncratic usage: a format that was originally designed to store alignments is now being increasingly used to represent "unaligned" data. Treat it as a testament of the shortcomings of the FASTQ files.

The advantage of the BAM file is that each read can be tagged with additional, user created information, for example the sample name. When used this way a single file can be used to store all samples from an experiment, hence greatly simplifying file management. Instead of having to deal with 100 paired end samples distributed over 200 files we can just store a single BAM file.

Here is an example of converting paired end FASTQ files in an unaligned BAM format, where each read in the pair is tagged as "FOO"

```
# Get the data.
fastq-dump -X 10000 --split-files SRR1972739

# Set up shortcuts.
R1=SRR1972739_1.fastq
R2=SRR1972739_2.fastq

# Convert to BAM file. Tag each sample as "FOO"
picard FastqToSam F1=$R1 F2=$R2 O=SRR1972739.bam  SM="FOO"
```

Look at the headers of the resulting file:

```
samtools view -H SRR1972739.bam
@HD    VN:1.5    SO:queryname
@RG    ID:A    SM:FOO
```

The so called "readgroup" A will be tagged as "FOO".

```
samtools view SRR1972739.bam | head -1
```

And the file contains (shown wrapped below):

```
#SRR1972739.1    77    *    0    0    *    *    0    0
TCTAAAAGGAGGACTCAATTGCAATATCTCATTTGAGTCCTCCTTTTAGAGGATTATTGTCATAAATCAATTCATTTTCTCGGTATCTTGTTAA
ATCTAAA
CCCFFFFFHHHHFIJJJJJJIJJJJJJJJJJJJJJGIGIJJJJJJJJJJJJJJJJJJJJIGIJJJJJJJJJJJJJJJJJJIHHHHDEFFFEEDEEE
EEDDDDD
RG:Z:A
```

Note again the `RG:Z:A` tag at the end of the read.

To analyze the data stored in an unaligned BAM file we typically need to unpack it into FASTQ format again like so (we unpack the into paired end files):

```
samtools fastq SRR1972739.bam -1 pair1.fq -2 pair2.fq
```

# How important is to understand the details of the SAM format?

On one hand, it is possible that one will not need to manipulate SAM/BAM files directly. There is a large library of programs and tools that process and analyze SAM/BAM files directly and seamlessly.

On the other hand, there are many problems that cannot be understood and solved without understanding and investigating these files in more detail.

# How to create SAM files?

In the chapter on short read aligners we produced SAM files with several tools. We'll use `bwa` as demonstrated there:

Get the sequencing reads.

```
fastq-dump -X 10000 --split-files SRR1972739
```

Get the reference genome.

```
efetch -db=nuccore -format=fasta -id=AF086833 > ~/refs/ebola/1976.fa
```

Set up shortcuts for nicer looking commands.

```
REF=~/refs/ebola/1976.fa
R1=SRR1972739_1.fastq
R2=SRR1972739_2.fastq
```

Index the genome.

```
bwa index $REF
```

Run the aligner in paired end mode.

```
bwa mem $REF $R1 $R2 > bwa.sam
```

The resulting file is in SAM format. We can repeat the same process with the bowtie2 aligner.

Build the bowtie2 index:

```
bowtie2-build $REF $REF
```

Generate the `bowtie2` specific SAM file.

```
bowtie2 -x $REF -1 $R1 -2 $R2  > bowtie.sam
```

# How to create a BAM file?

Typically you generate a SAM file, then sort that file and convert that into a `BAM` format. In addition you need to index the resulting `BAM` file.

The `samtools` package has come a long way in its usability and since version 1.3 will both convert and sort a `SAM` file into `BAM` in one step:

```
samtools sort bwa.sam > bwa.bam
```

Tip: when running in highly parallel mode (with say `parallels`) ensure that the `-T` prefix flag to `samtools sort` is set, otherwise the temporary files may be reused and produce unpredictable results!

Build an index for the `bwa.bam` file:

```
samtools index bwa.bam
```

A simpler way to produce the files is to sort and convert in one step:

```
bwa mem $REF $R1 $R2 | samtools sort > bwa.bam
samtools index bwa.bam
```

# How to create a CRAM file?

You need to add the reference to the sorting step and change the output format:

```
bwa mem $REF $R1 $R2 | samtools sort --reference $REF -O cram  > bwa.cram
samtools index bwa.cram
```

# Sequence Alignment Maps (SAM/BAM) specification

## What is a SAM file?

The SAM format is a TAB-delimited, line oriented text format consisting of a

1. **Header** section, where each line contains some metadata
2. **Alignment** section where each line contains information on an alignment

The SAM format specification lists the required and optional content for each of these sections.

```
# Get the sequencing reads.
fastq-dump -X 10000 --split-files SRR1972739

# Get the reference genome.
REF=~/refs/ebola/1976.fa
efetch -db=nuccore -format=fasta -id=AF086833 | seqret -filter -sid AF086833 > $REF

# Set up shortcuts for easier access.
R1=SRR1972739_1.fastq
R2=SRR1972739_2.fastq

# Index the genome.
bwa index $REF

# Run the aligner. Generate a SAM file from it.
bwa mem $REF $R1 $R2 > bwa.sam
```

## What is the SAM header?

Let's look at the start of the file:

```
cat bwa.sam | head
```

The lines that start with the `@` symbols are the so-called "SAM header" rows:

```
@SQ    SN:gi|10141003|gb|AF086833.2|    LN:18959
@PG    ID:bwa    PN:bwa    VN:0.7.12-r1039    CL:bwa mem /Users/ialbert/refs/ebola/1976.fa SRR
1972739_1.fastq SRR1972739_2.fastq
```

These headers are described in the SAM Specification. There is various information encoded here, for example `SN` is the sequence name that we aligned against (the name of the sequence in the Fasta reference file), `LN` is the length of this sequence, `PG` is the program version that we ran. But if you were to look at the alignment file produced with `bowtie`

```
cat bowtie.sam | head
```

you would see a slightly different header:

```
@HD    VN:1.0    SO:unsorted
@SQ    SN:gi|10141003|gb|AF086833.2|    LN:18959
@PG    ID:bowtie2    PN:bowtie2    VN:2.2.5    CL:"/Users/ialbert/src/bowtie2-2.2.5/bowtie2-al
ign-s --wrapper basic-0 -x /Users/ialbert/refs/ebola/1976.fa -1 SRR1972739_1.fastq -2 SRR19727
39_2.fastq"
```

While some information is the same, notice how the content diverges right at the header information. As we shall see it will only get more different.

# What is in the SAM alignment section?

The SAM Specification contains a very detailed description of each field and is a required read for all bioinformaticians. Whenever in doubt, consult it.

The following code also requires the presence of `samtools`, a command line utility to process SAM/BAM files.

# Column 1: Query Name (QNAME)

The simplest way to understand a SAM file is to go over it a few columns at a time.

There is a nomenclature for each field, for example column 1 is also called `QNAME`, which corresponds to the words "Query name". Let's look at the first column over five rows.

```
cat bwa.sam | grep -v @ | cut -f 1 | head -5
```

The same effect can be achieved with `samtools`:

```
samtools view bwa.sam | cut -f 1 | head -5
```

will list the name of the aligned query sequences:

```
SRR1972739.1
SRR1972739.1
SRR1972739.2
SRR1972739.2
```

# Column 2: FLAG

As it happens all the information necessary to figure out which read aligns in what way is included in the second column of the SAM format, in a "magical" number called `FLAG`:

```
cat bwa.sam | grep -v @ | cut -f 2 | head -5
```

that produces the following:

```
83
2115
115
2211
67
```

The `FLAG` is the field of the SAM spec where the designers of the format have strayed the furthest from what one might call *common sense*. It was an effort to "cram" as much information as possible into a single number and to do so they came up with a format that no biologist will ever manage to remember, hence paving the way for the long-term employment of all bioinformaticians. Ok, maybe that's not such a bad thing after all.

Here is how the rationale works: Imagine that we associate an attribute to each power of 2 starting from zero. Let's say

- `1` means "red",
- `2` means "blue",
- `4` means "tall",
- `8` means "short",
- `16` means "poor",
- `32` means "rich",

At this point it is not important what the attributes are just that they correspond to information we wish to know. If we were to describe an object as a `5` we would decompose that into powers (factors) of 2 like so `1 + 4`, in essence stating that it is "red" and "tall". Or if we had a `10` that would be `2 + 8` meaning that its attributes are "blue" and "short".

The justification behind this encoding is that instead of having to store two numbers, `2` and `8`, we can just store a single number, `10`, and still represent two attributes. Expanding this we can store up to 8 factors in a byte. And that is what a FLAG does. A single number representing 8 properties of the alignment.

Why powers of 2 and not of another number? Since the computer already stores all data and numbers in binary form, internally these are already decomposed into powers of 2. Accordingly, it is very efficient to computationally check which factors are present and which are absent (this is called a bitwise check).

It is essential to remember that if we wanted to identify the items that are "blue" (that is, `2`) we would need to select ALL the numbers that, when decomposed into powers of 2 would contain `2`. This is quite counter-intuitive as it includes the numbers `3`, `6`, `7` and `10` but would NOT include `4`, `9` or `16` etc.

See why: `1 + 2 = 3`, `2 + 4 = 6`, `1 + 2 + 4 = 7`, `2 + 8 = 10` and so on. So everything described as `3,6,7,10` is blue. But of course all these will be different in one or more of their other attributes. While elegant and fast, it is not really how we think about these problems.

First, it is very difficult to see by eye which flags are set and which are not Is flag `2` set in the numbers

`19` or `35`? Moreover, what if some of the attributes were mutually exclusive? For example, red and blue may be such attributes, but other attributes may be compatible. Are there checks and balances that help us select the right values?

One of the main challenges of the `FLAG` system is that it is exceedingly easy to mistakenly generate a wrong type of query as one adds more terms into it.

Imagine handling eight different attributes, some of which are exclusive and others not, some attributes are filled in by the aligner, others are not. Selecting or removing alignments by these attributes becomes unnecessarily complicated.

The flaw of the `FLAG` system is that it accentuates rather than solves an already difficult problem. Using the `FLAG` system correctly is surprisingly challenging.

The SAM specification lists the meaning of the flags and the `flags` command will list them at the command line:

```
samtools flags
```

generating:

```
Flags:
      0x1   PAIRED        .. paired-end (or multiple-segment) sequencing technology
      0x2   PROPER_PAIR   .. each segment properly aligned according to the aligner
      0x4   UNMAP         .. segment unmapped
      0x8   MUNMAP        .. next segment in the template unmapped
     0x10   REVERSE       .. SEQ is reverse complemented
     0x20   MREVERSE      .. SEQ of the next segment in the template is reversed
     0x40   READ1         .. the first segment in the template
     0x80   READ2         .. the last segment in the template
    0x100   SECONDARY     .. secondary alignment
    0x200   QCFAIL        .. not passing quality controls
    0x400   DUP           .. PCR or optical duplicate
    0x800   SUPPLEMENTARY .. supplementary alignment
```

Adding insult to injury, these are now listed in "octal" representation. Let's see what our flags mean, our list was:

```
83
2115
115
2211
67
```

We can ask for a description for each flag:

```
samtools flags 83
```

and that results in:

```
0x53   83   PAIRED,PROPER_PAIR,REVERSE,READ1
```

We can read this out loud. It says that a read labeled as `83` is a paired read, maps in the proper pair, it aligns onto the reverse strand and comes from the first file.

Now take a look the subsequence flags `2115` and `163` and decide which one of the three situations that we talked about in the QNAME section applies.

To generate statistics on the flags:

```
samtools flagstat bwa.sam
```

# Primary, secondary and chimeric (supplementary) alignments

The most important `FLAGS` characterizing the alignment type are:

1. **Primary alignment**: the best alignments by score. If there are multiple alignment that match equally well the aligner will designate one (software dependent) as the primary alignment.
2. **Secondary alignment**: In case of multiple matches one is designated as primary and all others are secondary.
3. **Supplementary alignmenr**: a read that cannot be represented as a single linear alignmet and matches two different locations without significant overlap. These are also known as "chimeric" alignments. What gets called as chimeric alignment is software/implementation dependent. Typically the entire read needs to be covered and the the alignments of the two halves is at considerable distance. Example: a read aligning across two exons while skipping an intron.

# Columns 3-4: Reference Name (RNAME) and Position (POS)

The 3rd and 4th columns of a SAM file indicate the reference name (RNAME) and the position (POS) of where the query, the sequence named in column 1 (QNAME) aligns:

```
samtools view bwa.sam | cut -f 1,3,4 | head -5
```

In our case, there is only one reference sequence (chromosome):

```
SRR1972739.1    AF086833    15684
SRR1972739.1    AF086833    15735
SRR1972739.1    AF086833    15600
SRR1972739.2    AF086833    4919
SRR1972739.2    AF086833    4863
```

We can see that each sequence aligns to the same reference at different positions: `15684`, `15735`, `15600` and so on.

> Everything in a SAM file is relative to the forward strand!
>
> The absolute essential thing to remember about the POS field is that it reflects the **leftmost** coordinate of the alignment. Even when the alignment matches on the reverse strand the POS coordinate is reported relative to the forward strand.

It is the `FLAG` column that tells us which strand the data aligns to `FLAG=4`

# Column 5-6: Mapping Quality (MAPQ) and Compact Idiosyncratic

Gapped Alignment Representation (CIGAR)

These columns reflect the Mapping Quality (MAPQ) and the so called Compact Idiosyncratic Gapped Alignment Representation (CIGAR).

```
samtools view bwa.sam | cut -f 5,6 | head -5
```

to produce:

```
60    69M32S
60    33M68H
60    101M
60    101M
60    47S54M
```

The values in the `MAPQ` column here are all `60`. This column was designed to indicate the likelihood of the alignment being placed incorrectly. It is the same Phred score that we encountered in the FASTQ files. And we read it the same way, `60/10 = 6` so the chance of seeing this alignment being wrong is `10^-6` or `1/1,000,000` one in a million.

There is one caveat though. The numbers that an aligner puts into the MAPQ field are typically estimates. It is not possible to mathematically compute this value. What this field actually does is to inform us on a guess by the aligner's algorithm. This guess is is a guideline that should not be treated as a continuous, numerical value. Instead it should be thought of as an ordered label, like "not so good" or "pretty good". As a matter of fact, aligner developers even generate special MAPQ qualities to mark special cases. For example `bwa` will generate a MAPQ=0 if a read maps equally well to more than one location.

The CIGAR string is a different beast altogether. It is meant to represent the alignment via numbers followed by letters:

- `M` match of mismatch
- `I` insertion
- `D` deletion
- `S` soft clip
- `H` hard clip
- `N` skipping

These are also meant to be "readable"; the `69M32S` says that `69` bases are a *match or mismatch* (and we'll discuss later why match and mismatch are counted together), then we *soft clip* `39` bases.

The CIGAR representation is a neat concept, alas it was developed for short and well-matching reads. As soon as the reads show substantial differences the CIGAR representation is much more difficult to read.

There are also different variants of CIGAR. The "default" CIGAR encoding describes both the match and mismatch the same way, with an `M`. There is some ill-fated rationale and explanation for having adopted this choice - one that complicates analysis even more.

The extended CIGAR encoding adopted by others uses the symbols of `=` and `X` to indicate matches and mismatches.

# Columns 7-9: RNEXT, PNEXT, TLEN

Columns 7,8 and 9 deal with the alignment of the mate pair. Since both reads that from a pair originated from the same fragment, it is very informative to know right away where both ends align. And that is what these columns do.

Let us also include the `QNAME` and `POS` fields and look at the data:

```
samtools view bwa.sam | cut -f 7,8,9 | head -5
```

We see the following:

```
=    15600  -153
=    15600  -136
=    15684   153
=    4863   -104
=    4919    104
```

The `=` symbol for the `RNEXT` field indicates that the query mate aligns to the same reference anchor (chromosome) as the query. The `PNEXT` fields indicates the `POS` field of the **primary(!)** alignment of the mate. (see later what primary alignment means). Note how every `PNEXT` always has a corresponding `POS` column on a different line since the mate is also in the file. Finally the `TLEN` column indicates the distance between the outer ends of mate alignments, that is, how long the fragment **appears** to be! It is very important to make this distinction. `TLEN` is not the actual length of the fragment - it is the length that it would be if the genome under study was identical to the reference and the mapping was correct. The specification calls it observed template length.

Working out how the `TLEN` is computed takes some effort. You usually don't need to do it manually but it is good to know how it works. To do it you need the `POS` and `CIGAR` information:

```
samtools view bwa.sam | cut -f 1,2,4,6,7,8,9 | head -1
```

That produces:

```
SRR1972739.1   83   15684   69M32S   =   15600   -153
```

The flag `83` indicates that the alignment is on the reverse strand and the left most position of it is at `15684` with its mate at `15600`. This means the following:

```
15600            15684
|                |
=======>         <==========
```

How long is the rightmost alignment? `69M23S` that means `69` matches, this pushes the right edge of this alignment to `15684 + 69 = 15753`,

```
15600            15684
|                |
=======>         <==========
                 |-- 69 --|
|---------  153 ----------|
```

This would be the start coordinate if this sequence had directionality. So the distance between the outer edges is at `15753 - 15600 = 153`, this value is reported as `-153` to indicate that the mate is towards the left of this query.

In our personal opion having to compute this like so is a flaw of the SAM specification. It should have just included the start and end of the alignment. It is exceedingly common to need this information and we typcically have to resort to tools such as `bedtools` to convert

```
samtools view -b bwa.sam | bedtools bamtobed  | head -1
```

That prints:

```
AF086833    15683   15752  SRR1972739.1/1   60   -
```

But of course as typical in bioinformatics this does not exactly match what we computed above. It is now a zero based coordinate system - the zero based "curse" of all who work in this field. Will the confusion ever end? Not likely.

# Columns 10-11: SEQ and QUAL

These columns store the sequence and the FASTQ quality measures associated with query.

```
samtools view bwa.sam | cut -f 10,11 | head -1
```

The most important detail (not quite explained well in the Samtool specification) that these columns store queries relative to the *forward* strand. What this means is that queries that align to the reverse strand are shown as their reverse complement. The soft and hard clip qualifiers of the `CIGAR` field describe if the

clipped and unaligned sequence is still included (soft clip) or not (hard clip) in the `SEQ` column. The `QUAL` column contains that sequence qualities that directly correspond to the `SEQ` columns.

# Optional Fields: Columns 12, 13 and on

Here is where it gets interesting and results produced by different aligners start to diverge. All optional fields follow the `TAG:TYPE:VALUE` format for example, where TAG is a two letter word, TYPE is a one character long (like i, Z, f) while the VALUE can be arbitrarily long. Lets look at some tags:

```
samtools view bwa.sam | cut -f 12,13,14,15 | head -1
```

will produce:

```
NM:i:2   MD:Z:27C16G24   AS:i:59   XS:i:0
```

As a rule TAGs that start with an X or Y are tool specific and are (should be) documented by the aligner.

# What do the SAM tags mean?

The definitions for SAM tags are distributed as a standalone document called [SAMtags.pdf][samtags]

- Tags are optional.
- Only some upper cased tags have pre-defined meanings.
- Tags that start with `X` are aligner
- Tags containing lowercase letters are always specific to the tool that filled them in.

  For example we can see there that `NM` is defined as "edit distance to the reference, including ambiguous bases but excluding clipping". And `MD` is *string for mismatching positions*, a different representation of the alignment itself, similar to CIGAR but not quite the same. There is a longer story to the `MD` tag, one that will be covered later.

The take-home lesson from the above is that the SAM format represents alignments and stores a fairly complex set of information on the way sequences align.

# Working with BAM files

The SAM/BAM format is the workhorse format of bioinformatics. The information in the SAM files (or lack thereof) may determine the success or failure of a project.

1. Convert all alignments into sorted, indexed BAM files
2. A sorted and tagged BAM file contains all the necessary information to reconstruct the sample identities
3. The information content of a BAM file will typically differ greatly depending on the software tool that created it.

Several tool sets have been created to manipulate BAM files:

1. `samtools` - the first and original tool, and one of the most powerful data analysis tools in all of bioinformatics. Many data analytics can be performed using `samtools` alone.
2. `bamtools` - originally conceived of to fill in functionality missing from `samtools`. In the meantime `samtools` has mostly caught up, but `bamtools` still offers some functions not available in `samtools`, for example, extracting alignments that match a certain tag.
3. `picard` - a toolset developed at the Broad Institute (named after Captain Picard in the Star Trek Next Generation series); Sometimes used in close conjunction with the GATK variant caller.

Many operations on BAM files fall into one of these two categories:

- Select alignments that match an attribute: strand, mate information, mapping quality
- Select alignments that align in a certain region of the genome.

The best resource to expore SAM flags is a site called Explain SAM Flags. This site allows you to not only build a series a flags but to instantly reverse their meaning. This can also be very error-prone otherwise.

# How can I select from or filter data from BAM files?

First, let's clarify some wording:

1. Selecting means to keep alignments that match a condition.
2. Filtering means to remove alignments that match a condition.

Filtering on `FLAGS` can be done via `samtools` by passing the parameters `-f` and `-F` that operate the following way:

- `-f flag` includes only alignments where the bits match the bits in the flag
- `-F flag` includes only alignments where the bits DO NOT match the bits in the flag.

Let's obtain and align a subset of the data `SRR1972739` against the genome stored under accession number `AF086833`.

While we're at it we'll simplify the sequence id of the reference genome to just `AF086833` rather than the keeping it the lengthy `gi|10141003|gb|AF086833.2|`. This latter long id can be annoying to work with as the `|` is a special character in `bash` (the pipe) and we'd have to always quote it. Here is our workflow:

```
# Obtain the data.
fastq-dump -X 10000 --split-files SRR1972739

# Use variables to make the command line nicer.
REF=~/refs/ebola/1976.fa
R1=SRR1972739_1.fastq
R2=SRR1972739_2.fastq
BAM=bwa.bam

# Obtain the reference genome and rewrite its sequence header.
efetch -db=nuccore -format=fasta -id=AF086833 | seqret -filter -sid AF086833 > $REF
head -1 $REF

# Index the reference genome.
bwa index $REF

# Create a SAM file, then sort and convert it to a BAM file.
bwa mem $REF $R1 $R2 | samtools sort > $BAM

# Index the bam file.
samtools index $BAM
```

Now onto `FLAGS`. For example, recall what flag `4` means:

```
samtools flags 4
```

It states:

```
0x4    4    UNMAP
```

This means that when this flag is set, the read is unmapped (unaligned). You may wonder why this alignment is included in the BAM file if it does not align. Sometimes we may want to know which read did not produce an alignment, so many aligners include them by default. To select and view the alignments where the read did not align you can do:

```
samtools view -f 4 bwa.bam | head
```

Or to count them you can do

```
samtools view -f 4 bwa.bam | wc -l
```

and that produces:

```
5461
```

So `5461` reads in our file are not actually aligned. Counting alignments with some property is so common that `samtools` offers a parameter `-c` to do that directly.

```
samtools view -c -f 4 bwa.bam
# 5461
```

The more important question may be -- How many valid alignments are there in the file? Use the capital `-F` parameter to count alignments that DO NOT have a given flag:

```
# How many don't have the UNMAP flag?
samtools view -c -F 4 bwa.bam
# 15279
```

# How to separate alignments into a new file?

Removing the `-c` parameter will produce the actual alignments, so if you wanted to create another BAM file that only contains the aligned reads you could do:

```
samtools view -b -F 4 bwa.bam > bwa.aligned.bam
samtools index bwa.aligned.bam
```

# How to get an overview of the alignments in a BAM file?

1. `samtools flagstat bwa.bam` - produces a report on flags
2. `samtools idxstats bwa.bam` - produces a report on how many reads align to each chromosome
3. `bamtools stats -in bwa.bam` - produces a report on flags

One observation that we made over the years is that the numbers reported by the various statistics programs never quite match! They are very similar but key aspects always seem to differ.

This is an indication that even at the basic level there is ambiguity in what the different terms mean to different people. Note, for example, how below `samtools flagstat` reports `14480` as "proper pairs" whereas `bamtools stats` reports `15216` for the same quantity.

The output of `samtools flagstat`

```
20740 + 0 in total (QC-passed reads + QC-failed reads)
0 + 0 secondary
740 + 0 supplementary
0 + 0 duplicates
15279 + 0 mapped (73.67% : N/A)
20000 + 0 paired in sequencing
10000 + 0 read1
10000 + 0 read2
14480 + 0 properly paired (72.40% : N/A)
14528 + 0 with itself and mate mapped
11 + 0 singletons (0.05% : N/A)
0 + 0 with mate mapped to a different chr
0 + 0 with mate mapped to a different chr (mapQ>=5)
```

The output of `bamtools stats`

```
*****************************************
Stats for BAM file(s):
*****************************************

Total reads:       20740
Mapped reads:      15279    (73.6692%)
Forward strand:    14393    (69.3973%)
Reverse strand:    6347    (30.6027%)
Failed QC:         0    (0%)
Duplicates:        0    (0%)
Paired-end reads:  20740    (100%)
'Proper-pairs':    15216    (73.3655%)
Both pairs mapped: 15268    (73.6162%)
Read 1:            10357
Read 2:            10383
Singletons:        11    (0.0530376%)
```

How could you verify which one is correct? Take the definition of proper pair as flag `2`:

```
samtools view -c -f 2 bwa.bam
# 15216
```

If you count the `PROPER_PAIR` flag the `bamtools stats` is correct.

So why does `samtools flagstat` come up with `14480`? It turns out that it counts only the properly mapped AND primary alignment reads. Basically, it is counting each read only once in the cases that a read produces multiple alignments:

```
samtools view -c -f 2 -F 2308 bwa.bam
# 14480

# What the heck is flag 2308?
samtools flags 2308
# 0x904    2308    UNMAP,SECONDARY,SUPPLEMENTARY
```

Note how we keep alignments that are NOT like this.

# How do I use flags?

The most important rule is to ensure that you select for flag `4`, the `UNMAP`. Even though it makes little sense that a read could be simultaneously unmapped and on a reverse strand, this can be reported as such:

```
# Reads that align to reverse strand.
samtools view -c -f 16  bwa.bam
# 6347

# Reads that align to reverse strand and are not unmapped.
samtools view -c -F 4 -f 16  bwa.bam
# 6343

# Reads that align to forwards strand and are not unmapped.
samtools view -c -F 20 bwa.bam
# 8936
```

While we are here let us note a few warning signs. We have not even started analyzing our data, but we've already run into an inconsistency in merely reporting what we have. Let alone interpreting it one way or another. Alas, these types of problems will only get worse from here on out.

- Biostar Post of the Day: How can an unmapped read align on the reverse strand?

# How do I combine multiple flags?

This gets a little complicated.

When investigating any attribute we may need to select for and against certain features. But as we combine the flags, the way we order the flags is not symmetric anymore when using `-f` and `-F` together. To select all reads that align (oppose `4`) and are mapped to the reverse strand (select for `16`) we do this:

```
# Alignments on the reverse strand.
# Oppose unmapped (4) and select for reverse strand (16).
samtools view -c -F 4 -f 16  bwa.bam
# 6343

# Alignments on the forward strand.
# Oppose unmapped (4) and oppose the reverse strand (16).
samtools view -c -F 4 -F 16  bwa.bam
# 8936

# We can combine the above into one flag: 4 + 16 = 20
samtools view -c -F 20  bwa.bam
# 8936
```

As you select for more attributes the mental gymnastics add up. Here is an example. If you wanted to separate strand-specific data so that each file only contains alignments for transcripts that come from the forward strand you would do:

```
# Alignments of the second in pair if they map to the forward strand.
samtools view -b -F 4 -f 128 -F 16 data.bam > fwd1.bam

# Alignments of the first in pair if they map to the reverse strand.
samtools view -b -F 4 -f 80 data.bam > fwd2.bam
```

Not surprisingly, this is not easy to figure out and apply correctly.

It gets even more complicated for paired-end strand-specific data. We had to write a tutorial (mainly so that we also remember it) on this:

- Biostar Post of the Day: How To Separate Illumina Based Strand Specific Rna-Seq Alignments By Strand

# How should I interpret seemingly invalid alignments?

It is possible to create selections that seemingly make no sense. For example, let us select alignments where the read and its mate both align to the reverse strand.

```
samtools flags PAIRED,PROPER_PAIR,REVERSE,MREVERSE
# 0x33    51    PAIRED,PROPER_PAIR,REVERSE,MREVERSE
```

There are `124` of these:

```
# Unique alignments, mapped, paired, proper pair, both reads
# in a pair align to the reverse strand.
samtools view -c -q 1 -F 4 -f 51 bwa.bam
# 124
```

If our genome matched the reference and the alignments are correct this would be impossible to observe. Given that we do see these, there are two explanations.

1. Incorrect alignments: The reads have been misplaced.
2. Genomic rearrangement: The real genome has a sequence inversion that makes it "look like" one read of the pair is inverted.

Remember that the real genome does not need to match the reference - and real, biological changes could look like incorrect mapping.

# How do I filter on mapping quality?

The mapping quality column contains a number that the aligner fills in. This number is rarely an objective quantity, it rather reflects a subjective value designated by the tool developer. For example, for `bwa` a mapping quality of `0` indicates a read that maps to multiple locations. Does this make sense? Not really, the likelihood that a read can be in multiple locations should be closer to `1/N` ( `N` is the number of possible alignments) - but `bwa` reports it as zero. Do other tools use the same rationale? Typically not. You have to check the documentation for the particular software you are using to know for sure.

The `-q` flag selects alignments whose mapping quality is at or above the value

```
# Select uniquely mapped reads when these were aligned with BWA.
samtools view -c -q 1 bwa.bam
# 15279
```

Selection options can be combined:

```
# Mapping quality over 1 and on the forward strand.
samtools view -c -q 1 -F 4 -F 16 bwa.bam
# 8936
```

# How can I find out the depth of coverage?

There are several tools that can compute the number of reads that overlap each base. With samtools you can do:

```
# Compute the depth of coverage.
samtools depth bwa.bam  | head
```

This produces a file with name, coordinate and coverage on that coordinate:

```
AF086833    46    1
AF086833    47    1
AF086833    48    1
...
```

Run `bamtools coverage` :

```
bamtools coverage -in bwa.bam | head
```

The output seems identical

```
AF086833    45    1
AF086833    46    1
AF086833    47    1
...
```

Except note how it is "one off"! The index starts at 0 rather than 1 in the latter case. This type of inconsistency is extraordinarily common in bioinformatics and causes lots of errors.

You could ask for all positions to be shown with `samtools` and you can verify that it starts at 1

```
samtools depth -a bwa.bam  | head
```

it produces:

```
AF086833    1    0
AF086833    2    0
AF086833    3    0
```

You could sort by depth, though this could take a long for a large genome size. To find the most covered region:

```
samtools depth bwa.bam  | sort -k 3 -rn | head
```

Produces:

```
AF086833    4609    163
AF086833    4608    163
AF086833    4611    157
...
```

# What is a "SECONDARY" alignment?

A secondary alignment refers to a read that produces multiple alignments in the genome. One of these alignments will be typically referred to as the "primary" alignment.

```
samtools flags SECONDARY
# 0x100    256    SECONDARY

samtools view -c -F 4 -f 256 bwa.bam
# 0
```

There are no reads that map in multiple locations in this data set.

The SAM specification states:

> Multiple mappings are caused primarily by repeats. They are less frequent given longer reads. If a read has multiple mappings, and all these mappings are almost entirely overlapping with each other; except the single-best optimal mapping, all the other mappings get mapping quality <Q3 and are ignored by most SNP/INDEL callers.

To be honest we don't fully understand the statement above.

# What is a "SUPPLEMENTARY" alignment?

A supplementary alignment (also known as a chimeric alignment) is an alignment that cannot be represented as a single unbroken line and where the read partially matches different regions of the genome without overlapping the same alignment:

If this the original read, say 8 bp long:

```
--------
12345678
```

Then this is a chimeric alignment:

```
---------------------
   ---        -----
   123        45678
```

But this is not a chimeric alignment:

```
----------------------
     -----
     12345
         ---
         678
```

Typically, one of the linear alignments in a chimeric alignment is considered the "representative" or "primary" alignment, and the others are called "supplementary" The decision regarding which linear alignment is representative (primary) is arbitrary.

The SAM specification states:

> Chimeric alignment is primarily caused by structural variations, gene fusions, misassemblies, RNA-seq or experimental protocols. []. For a chimeric alignment, the linear alignments consisting of the aligment are largely non-overlapping; each linear alignment may have high mapping quality and is informative in SNP/INDEL calling.

To select supplementary alignments:

```
samtools flags SUPPLEMENTARY
# 0x800    2048    SUPPLEMENTARY

samtools view -c -F 4 -f 2048 bwa.bam
# 740
```

# What is a "PRIMARY" or "REPRESENTATIVE" alignment?

Typically these represent the "best" alignment although the word "best" may not be well defined. Each read will only have one primary alignment and other secondary and supplemental alignments.

There is no flag to select for primary alignments, the opposite of "SUPPLEMENTARY" and "SECONDARY" will be the primary alignment. To select primary alignments:

```
samtools flags SUPPLEMENTARY,SECONDARY
# 0x900    2304    SECONDARY,SUPPLEMENTARY

samtools view -c -F 4 -F 2304 bwa.bam
# 14539
```

# Analyzing SAM files

You can perform sophisticated data analyses with just samtools.

The sections below assume that you obtained sequence data for a published Ebola run and prepared the data by aligning it with two short read aligners, `bwa` and `bowtie2`:

```
# Set up shorcuts.
REF=~/refs/ebola/1976.fa
R1=SRR1972739_1.fastq
R2=SRR1972739_2.fastq

# Get the reference genome.
# Rename sequence ID to the simpler AF086833.
efetch -db=nuccore -format=fasta -id=AF086833 | seqret -filter -sid AF086833 > ~/refs/ebola/19
76.fa

# Obtain the dataset.
fastq-dump -X 10000 --split-files SRR1972739

# Index reference with bwa
bwa index $REF

# Align with bwa mem.
bwa mem $REF $R1 $R2 | samtools sort > bwa.bam
samtools index bwa.bam

# Index reference with bowtie2.
bowtie2-build $REF $REF

# Align with bowtie2
bowtie2 -x $REF -1 $R1 -2 $R2 | samtools sort > bowtie.bam
samtools index bowtie.bam
```

# Are my alignment files different?

The `flagstat` command generates a quick report

```
samtools flagstat bwa.bam
# 15279 + 0 mapped (73.67% : N/A)

samtools flagstat bowtie.bam
# 12537 + 0 mapped (62.69% : N/A)
```

How about the alignment tag sections?

```
samtools view bwa.bam | cut -f 12-18 | head -3

# NM:i:1   MD:Z:81C19       AS:i:96   XS:i:0
# NM:i:3   MD:Z:65C21C5A7   AS:i:86   XS:i:0
# NM:i:3   MD:Z:47C21C5A25  AS:i:86   XS:i:0

samtools view bowtie.bam | cut -f 12-18 | head -3
# AS:i:-6    XN:i:0   XM:i:1   XO:i:0   XG:i:0   NM:i:1   MD:Z:81C19
# AS:i:-16   XN:i:0   XM:i:3   XO:i:0   XG:i:0   NM:i:3   MD:Z:65C21C5A7
# AS:i:-16   XN:i:0   XM:i:3   XO:i:0   XG:i:0   NM:i:3   MD:Z:47C21C5A25
```

What do these tags mean? See the SAM Flags chapter

# What are read groups?

Read groups are tags labeled as `RG` in a SAM file to represent sample information. Unordered multiple `@RG` records are allowed, and typically the following tags need to be specified:

- `ID` Read group identifier
- `LB` Library
- `SM` Sample

There are more tags that can be set for a read group. See the SAM Flags chapter for a list of all the options.

There may be a conceptual hierarchy between the read group tags that reflect the library preparation process. For example: a sample may be listed with multiple libraries, but a library may not be listed with multiple samples.

Read group tags are typically used when all alignments for all samples are to be stored together, but tracked separately, in the same file. When you have a separate BAM file for each sample these tags may still be helpful, but are not be required.

Some pipelines, notably the GATK variation caller, will only work if the read groups are set.

# How can I add read groups to my files?

The read groups may be be added during the alignment phase via a parameter to the aligner. For `bwa` this would look like this:

```
# Use a shortcut for clarity.
TAG='@RG\tID:xyz\tSM:Ebola\tLB:patient_100'

# Add the tags during alignment
bwa mem -R $TAG $REF $R1 $R2 | samtools sort > bwa.bam
samtools index bwa.bam
```

To see the tags you can print the BAM file header with the `-H` flag:

```
samtools view -H bwa.bam
```

prints:

```
@HD    VN:1.3  SO:coordinate
@SQ    SN:AF086833  LN:18959
@RG    ID:xyz  SM:Ebola  LB:patient_100
...
```

To see the tags, write:

```
samtools view bwa.bam | cut -f 12-16 |  head  -1
# NM:i:1    MD:Z:81C19    AS:i:96    XS:i:0    RG:Z:xyz
```

# How can I modify read groups?

The `samtools addreplacerg` command can do this:

```
# This will be the new tag.
NEWTAG='@RG\tID:abc\tSM:Ebola\tLB:patient_101'

# Replace existing readgroups.
samtools addreplacerg -m overwrite_all -r $NEWTAG bwa.bam -O BAM > newbam.bam
```

The same command can be used to add read groups to a BAM file that doesn't have them yet.

# How can I merge BAM files?

Sorted `BAM` files can be merged with the `samtools merge` command. Let's merge our `bwa` and `bowtie` alignments into a single file while still retaining the information about which alignment was produced by which tool. Read groups can do that:

```
# Create the read group that will identify
# alignments performed with bwa.
BWA_TAG='@RG\tID:bwa\tSM:bwa\tLB:bwa'

# Replace existing readgroups.
samtools addreplacerg -m overwrite_all -r $BWA_TAG bwa.bam -O BAM > newbam.bam

# Create the read group that will identify
# alignments performed with bowtie2.
BOWTIE_TAG='@RG\tID:bowtie\tSM:bowtie\tLB:bowtie'

# Replace existing readgroups.
samtools addreplacerg -m overwrite_all -r $BOWTIE_TAG bowtie.bam -O BAM > newbowtie.bam
```

Merge both files to create the `all.bam` file:

```
samtools merge all.bam newbam.bam newbowtie.bam
```

What does the header look like for the `all.bam` file?

```
samtools view -H all.bam
# @HD    VN:1.3    SO:coordinate
# @SQ    SN:AF086833    LN:18959
# @RG    ID:bwa    SM:bwa    LB:bwa
# @RG    ID:bowtie    SM:bowtie    LB:bowtie
```

From now on if we wanted to filter alignments for just one library:

```
# Filter for library tagged as bwa
samtools view -c -l bwa all.bam
# 20740

# Filter for library tagged as bowtie
samtools view -c -l bowtie all.bam
# 20000
```

# Can I create custom tags?

Yes. Any alignment may be tagged with any additional information as long as it is of the form

`TAG:FORMAT:VALUE` where

- `TAG` must be two letters (and not a predefined pair)
- `FORMAT` : must be one of 'z', 's', `s`
- `VALUE` : should not have whitespace

For example you could add `YY:s:gene` to tag reads that overlap with genes.

For more details see the SAM tag specification.

# How do I query a BAM file?

Sorted and indexed BAM files allow for very fast query over intervals.

Get a GenBank format file for the Ebola virus:

```
efetch -db=nuccore -format=gb -id=AF086833 > AF086833.gb
```

View this file:

```
more AF086833.gb
```

You can see the coordinates for various genomic features. Pick gene `NP` :

```
CDS  470..2689
     /gene="NP"
     /function="encapsidation of genomic RNA"
     /codon_start=1
     /product="nucleoprotein"
     /protein_id="AAD14590.1"
```

You can query over this location with samtools view:

```
samtools view -c bwa.bam AF086833:470-2689
# 1655
```

It prints the number of alignments that overlap with this particular coordinate `1655`. The same can be done for the bowtie alignment as well:

```
samtools view -c bowtie.bam AF086833:470-2689
1456
```

If this were an experiment where you needed to know the coverage over a certain feature, your task would now be completed.

# How can I query files over the web?

If the web-servers are set up correctly to stream data you can quickly query BAM files over the web without having to download the entire (potentially very large) dataset. The following file is located on the Biostar Handbook data site.

```
URL=http://data.biostarhandbook.com/bam/demo.bam
samtools view $URL | head
```

Some groups distribute their data over the web as well.

For example, the genome sequence of a Denisovan individual that lived 45,000 years ago was generated from a small fragment of a finger bone discovered in the Denisova Cave in southern Siberia in 2008.

- Website: http://www.eva.mpg.de/denisova
- Data: http://cdna.eva.mpg.de/denisova/

You can quickly analyze this data:

```
URL=http://cdna.eva.mpg.de/denisova/alignments/T_hg19_1000g.bam

# Show the header information for this file.
samtools view -H $URL

# How many reads overalp with the 1 million bp wide
# interval starting on chromosome 3, at 120 million.
samtools view -c $URL 3:120,000,000-121,000,000
```

Note how fast the access is. The whole file itself is over 80Gb, yet the BAM query is able to access just a small section of it.

> It is important to note that web access is only helpful when you need to check a small subset of the data. If you need some properties of the entire dataset, you are better off downloading the data first, since you would need to stream it off the web anyway.
>
> Web access can be useful to answer well-defined queries against a large number of datasets available on the web.

Other large sequencing projects have dedicated websites for data distribution. For example visit ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/data_collections/1000_genomes_project/ the directory structure contains lots of data sets. One such dataset is:

```
URL=ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/data_collections/1000_genomes_project/data/FIN/HG
00183/alignment/HG00183.alt_bwamem_GRCh38DH.20150718.FIN.low_coverage.cram
```

Note, this file is in CRAM format. When you first run `samtools` on such an alignment file it will download a reference for it. This download is about 273MB and is required only once. Subsequently the data access is much faster:

```
samtools depth $URL -r chr22 | head
```

or:

```
samtools mpileup $URL -r chr22 | head
```

# SAM Flags

- The SAM format specification is the official specification of the SAM format.
- The SAM tag specification is the official specification of the SAM tags.

## What are the required SAM columns?

1. `QNAME` String [!-?A-~]{1,254} Query template NAME
2. `FLAG` Int [0,65535] bitwise FLAG
3. `RNAME` String *|[!-()+-<>-~][!-~]* Reference sequence NAME
4. `POS` Int [0,2147483647] 1-based leftmost mapping POSition
5. `MAPQ` Int [0,255] MAPping Quality
6. `CIGAR` String *|([0-9]+[MIDNSHPX=])+ CIGAR string
7. `RNEXT` String *|=|[!-()+-<>-~][!-~]* Ref. name of the mate/next read
8. `PNEXT` Int [0,2147483647] Position of the mate/next read
9. `TLEN` Int [-2147483647,2147483647] observed Template LENgth
10. `SEQ` String *|[A-Za-z=.]+ segment SEQuence
11. `QUAL` String [!-~]+ ASCII of Phred-scaled base QUALity+33

## What do the flags mean?

```
   Binary       Integer   Name             Meaning
000000000001       1     PAIRED          Read paired
000000000010       2     PROPER_PAIR     Read mapped in proper pair
000000000100       4     UNMAP           Read unmapped
000000001000       8     MUNMAP          Mate unmapped
000000010000      16     REVERSE         Read reverse strand
000000100000      32     MREVERSE        Mate reverse strand
000001000000      64     READ1           First in pair, file 1
000010000000     128     READ2           Second in pair, file 2
000100000000     256     SECONDARY       Not a primary alignment
001000000000     512     QCFAIL          Read fails platform/vendor quality checks
010000000000    1024     DUP             Read is PCR or optical duplicate
100000000000    2048     SUPPLEMENTARY   Supplementary alignment
```

## How to build compound flags?

The best resource is a site called Explain SAM Flags. The site allows us to not only build a series a flags but to instantly reverse their meaning, a process that can be very error-prone otherwise.

It is worth reflecting for a moment that the format is so complicated that we need a separate, standalone website just to understand one of its columns.

You can also build the flag that has all the right bits at the command line:

```
samtools flags PAIRED,PROPER_PAIR,REVERSE
```

will print the integers value `19` :

```
0x13    19    PAIRED,PROPER_PAIR,REVERSE
```

You can also go in "reverse":

```
samtools flags 19
```

prints:

```
0x13    19    PAIRED,PROPER_PAIR,REVERSE
```

# What are SAM tags?

There are two types of tags:

- Header tags: These are predefined in the header and each one is present for each alignment.
- Alignment tags: These are filled in by the aligner and may be predefined or custom tags.

# What do the SAM header tags mean?

Header tags start with the `@` symbol and are at the beginning of the `SAM` file.

## @HD: The header line

The first line if present.

- `VN` Format version
- `SO` Sorting order of alignments
- `GO` Grouping of alignments

## @PG: Program

- `ID` Program record identifier
- `PN` Program name
- `CL` Command line
- `PP` Previous @PG-ID
- `DS` Description
- `VN` Program version

## @RG: Read group

This is one of the most important headers in a SAM file.

Unordered multiple `@RG` lines are allowed.

- `ID` Read group identifier
- `CN` Name of sequencing center producing the read
- `DS` Description
- `DT` Date the run was produced (ISO8601 date or date/time)
- `FO` Flow order
- `KS` The array of nucleotide bases that correspond to the key sequence of each read
- `LB` Library
- `PG` Programs used for processing the read group
- `PI` Predicted median insert size
- `PL` Platform/technology used to produce the reads
- `PM` Platform model
- `PU` Platform unit, a unique identifier
- `SM` Sample

### @SQ: Reference sequence dictionary

The order of `@SQ` lines defines the alignment sorting order.

- `SN` Reference sequence name
- `LN` Reference sequence length
- `AS` Genome assembly identifier
- `M5` MD5 checksum of the sequence in the uppercase
- `SP` Species
- `UR` URI of the sequence

## What do the SAM alignment tags mean?

The definitions for SAM tags are distributed as a standalone document called SAMtags.pdf

- Tags are optional.
- Only some upper case tags have pre-defined meanings.
- Tags that start with `X`, `Y` ' and `Z` or are in lowercase letters are always specific to the tool that filled them in. You need to consult the tool documentation for what they mean.

The current specification calls for the following pre-defined tags but please do note that no aligner fills in all and most aligners only fill very few of them:

- `AM` The smallest template-independent mapping quality of segments in the rest
- `AS` Alignment score generated by aligner
- `BC` Barcode sequence
- `BQ` Offset to base alignment quality (BAQ)
- `CC` Reference name of the next hit
- `CM` Edit distance between the color sequence and the color reference (see also NM)
- `CO` Free-text comments

- `CP` Leftmost coordinate of the next hit
- `CQ` Color read base qualities
- `CS` Color read sequence
- `CT` Complete read annotation tag, used for consensus annotation dummy features.
- `E2` The 2nd most likely base calls
- `FI` The index of segment in the template
- `FS` Segment suffix
- `FZ` S Flow signal intensities
- `GC` Reserved for backwards compatibility reasons
- `GQ` Reserved for backwards compatibility reasons
- `GS` Reserved for backwards compatibility reasons
- `H0` Number of perfect hits
- `H1` Number of 1-difference hits (see also NM)
- `H2` Number of 2-difference hits
- `HI` Query hit index
- `IH` Number of stored alignments in SAM that contains the query in the current record
- `LB` Library
- `MC` CIGAR string for mate/next segment
- `MD` String for mismatching positions
- `MF` Reserved for backwards compatibility reasons
- `MQ` Mapping quality of the mate/next segment
- `NH` Number of reported alignments that contains the query in the current record
- `NM` Edit distance to the reference
- `OC` Original CIGAR
- `OP` Original mapping position
- `OQ` Original base quality
- `PG` Program
- `PQ` Phred likelihood of the template
- `PT` Read annotations for parts of the padded read sequence
- `PU` Platform unit
- `QT` Barcode (BC or RT) phred-scaled base qualities
- `Q2` Phred quality of the mate/next segment sequence in the R2 tag
- `R2` Sequence of the mate/next segment in the template
- `RG` Read group
- `RT` Barcode sequence (deprecated; use BC instead)
- `SA` Other canonical alignments in a chimeric alignment
- `SM` Template-independent mapping quality
- `SQ` Reserved for backwards compatibility reasons
- `S2` Reserved for backwards compatibility reasons
- `TC` The number of segments in the template
- `U2` Phred probability of the 2nd call being wrong conditional on the best being wrong
- `UQ` Phred likelihood of the segment, conditional on the mapping being correct
- `X?` Reserved for end users
- `Y?` Reserved for end users
- `Z?` Reserved for end users

# What are BWA aligner specific tags?

Out of the pre-defined tags `bwa` fills the following:

- `NM` Edit distance
- `MD` Mismatching positions/bases
- `AS` Alignment score
- `BC` Barcode sequence

Lowercase tags or those that start with a `X`, `Y` or `Z` are application specific. The `bwa` aligner fills in the following `bwa` specific tags:

- `X0` Number of best hits
- `X1` Number of suboptimal hits found by BWA
- `XN` Number of ambiguous bases in the referenece
- `XM` Number of mismatches in the alignment
- `XO` Number of gap opens
- `XG` Number of gap extentions
- `XT` Type: Unique/Repeat/N/Mate-sw
- `XA` Alternative hits; format: (chr,pos,CIGAR,NM;)*
- `XS` Suboptimal alignment score
- `XF` Support from forward/reverse alignment
- `XE` Number of supporting seed

# Some Programming Required

As soon as you start learning to perform data analysis you may run into unanticipated problems:

- Existing software tools can rarely do all steps.
- You may need to bridge small formatting differences with simple transformations.
- You may need to compute values based on other values.

Not being able to get past these may bring your project to a halt.

The good news is that the programming skills required to get past these obstacles are often easy to learn! Many times writing just a few lines or applying a formula is enough. You can get a lot of mileage out of very basic skills.

## What kind of programming languages are out there?

Computer programs can be divided into two main categories:

- Compiled programs: the program is (usually) a standalone executable that can be run on any other computer of the same type.
- Interpreted programs: the program requires the presence of a specific programming language on every computer that is meant to execute it.

## What programming languages are good for Bioinformatics to learn?

Modern bioinformatics uses mainly the following languages:

- `C` - Compiled. Efficient and powerful but with high barriers of entry.
- `Java` - Interpreted. Allows building more complex software with less effort.
- `Python` - Interpreted. Simple and straightforward to learn, it can be substantially slower than C and Java in some cases.
- `Awk` - Interpreted. Ancient and somewhat simplistic, considered obsolete, its usage patterns match the bioinformatics data formats.
- `Perl` - Interpreted. Formerly the most popular language for doing bioinformatics -- by today it has fallen in popularity relative to Python.
- `R` - Interpreted. Language used mainly for data analysis and visualization.

## Which programming language should I learn?

If you are new to the field and you have the time to devote learning a programming language we would recommend starting with `Python`.

At the same time we recommend familiarizing yourself with `awk` as it both extremely simple and it can be a very handy tool for everyday data analysis.

# Programming with Awk

- Enlightened with Perl? You've ascended to a *PerlMonk*!
- Blazing trails with Python? You've become a *Pythoneer*!
- Following your dreams with Awk? You are getting *Awk-ward*!

To perform the examples in this section obtain the BAM file as;

```
URL=http://data.biostarhandbook.com/bam/demo.bam
curl $URL > demo.bam
```

# Why is Awk popular with bioinformaticians?

Awk operates on one line at a time, and the typical awk program is so short that it can be listed at the command line. While we can use `cut` to select columns we can't use it to rearrange them or to compute new or other values. Here is a comparison of `cut` and `awk` :

```
# Cutting columns 1,3, and 4.
samtools view demo.bam | cut -f 1,3,4 | head

# With awk we can cut out and rearrange the columns
# and we can compute the actual likelihoods from the MAPQ quality
samtools view demo.bam | awk '{ print $3, $2, $1, 10^(-$5/10) }'  | head
```

The expression `{ print $3, $2, $1, 10^(-$5/10) }` is an awk program.

# How does Awk work?

An awk program works on a line by line basis and for each line of a file it attempts the following:

```
awk 'CONDITION { ACTIONS }'
```

That is for each line it tries to match the `CONDITION`, and if that condition matches it performs the `ACTIONS`. Do note the curly brackets and the quotes. Note that multiple conditions and actions may be present:

```
awk 'CONDITION1 { ACTIONS1 } CONDITION2 { ACTIONS2 } CONDITION3 { ACTIONS3 }'
```

The simplest condition is no condition at all. The simplest action is no action at all. This makes the simplest awk program `awk '{}'` that you can run:

```
samtools view demo.bam | awk '{}'  | head
```

Of course that does not actually do anything to your data. A simple action could be `print`. Awk automatically splits the input (see the caveats of default splitting later) into variables named `$1`, `$2`, `$3` etc.

```
samtools view demo.bam | awk '{ print $5 }'  | head
```

We can use a condition in awk to check that column `$5` is less than `60` and print the value only in that case.

```
samtools view demo.bam | awk '$5 < 60 { print $5 }'  | head
```

The `print` command gets executed only when the fifth column is less than 60.

# How can I write more complicated Awk programs?

While awk is extremely useful for simple processing or quick answers, we would advise you to avoid writing highly complex awk (or bash) programs altogether.

Awk is strongest when the whole program is simple enough to be listed (and understood) on the command line. When written that way we can visually inspect it and we can ideally tell precisely what the command does. If the commands were in a separate file we'd have to inspect that separately and that would take away most of the readability of the process.

That being said when there are more actions these can be placed into a file like so:

```
CONDITION1 {
    ACTIONS1
}

CONDITION2 {
    ACTIONS2
}
```

and we can run the program through `awk` via the `-f` flag:

```
samtools view demo.bam | awk -f myprogram.awk  | head
```

# How is the output split into columns?

Superficially the default operation of awk appears to split lines into columns by any whitespace (spaces, tabs) -- in reality it does a lot more than that. It also collapses consecutive white spaces into a single one. This is a subtle difference that often does not matter - but when it does, the most devious and subversive of errors may occur.

With the default splitting behavior when we split the lines containing:

```
A B
A  B
```

we end up with the same result `$1=A` and `$2=B` for each line. In this case this may be what we wanted.

In general, and especially when processing tabular data, we absolutely don't want this behavior. Imagine a tab delimited file where tabs indicate columns. Say in one row two empty columns follow each other. The default awk splitting behavior would collapse these two tabs into one and subsequently shift the rest of the columns by one, producing values from the wrong column. So reading column 10 would sometimes give you the value in column 10 but other times the value in column 11, 12 etc. Needless to say this is highly undesirable behavior.

In general you always want to specify the splitting character when using awk:

```
awk -F '\t'
```

if you get tired of typing this all the time, make an alias in `.bashrc` like so:

```
alias awk="awk -F '\t'"
```

But do recognize, your commands may not work the same way for other people that do not have the same alias set.

# What are some awk special variables I should know about?

When awk runs, a number of variables are set beyond the column variables `$1`, `$2` etc.

- `$0` is the original line.
- `NF` number of fields in the current line (number of columns)
- `NR` number of records, the number of lines processed (line number)
- `OFS` output fields separator, the character placed between items when printed

As an example usage:

```
samtools view demo.bam | awk '{ print NR, NF }'  | head
```

This prints:

```
1 15
2 15
3 15
4 16
5 15
```

Note how line number 4 contains an extra column!

# Are there any special Awk patterns I should know about?

Awk has special patterns called `BEGIN` and `END` to run certain tasks only once.

```
samtools view demo.bam | awk ' BEGIN { print "Hello!" } END { print "Goodbye!" } '  | head
```

produces:

```
Hello!
Goodbye!
```

These are typically used to set up a computation and then to report results after the run has completed. For example, to compute the average template length (DNA fragment size) of an alignment file we could use (let's select the proper pairs only with -f 2):

```
samtools view -f 2 demo.bam | awk ' $9 > 0 { sum = sum + $9; count=count + 1 } END { print sum /count } '
```

That being said there is usually no need to re-implement computing sums or averages etc. The above can be solved a lot more elegantly with `datamash`:

```
samtools view -f 2 demo.bam | awk ' $9 > 0 { print $9 } ' | datamash mean 1
```

Note how we still need `awk`, it is just we use it for one thing only, selecting lengths that are positive.

# How can I build more complex conditions in Awk?

Make sure to use double quotes within patterns as to not conflict with the single quotes that the whole awk program is enclosed within.

You can use the following constructs.

- `>` or `<` comparisons: `'$1 < 60 { print 41 }'`
- `==`, `!=` for equal, not equal: `'{ $1 == "ORF" { print $1 }'`
- `~`, `!~` for pattern match, or pattern no match (regular expressions): `' $1 ~ "YALC|YALW" { print $1 }'`
- another way to match patterns is to enclose the pattern within `/`, this construct is called a regexp constant. These may be simpler to write, less putzing around with quotes: `' $1 ~ /YALC|YALW/ { print $1 }'`

# How can I format the output of Awk?

Producing properly formatted output is always one of pressing concern. By default, printing with awk will concatenate outputs with the value of the `OFS` variable (Output Field Separator):

```
echo | awk ' { print 1,2,3 }'
```

by default will print:

```
1 2 3
```

changing `OFS` can put other characters between the elements, perhaps a `TAB` :

```
echo | awk '{ OFS="\t"; print 1,2,3 }'
```

this will now print:

```
1    2    3
```

At some point you will need precise control on how values are formatted. Then you will need to use the so-called formatted print `printf` where the output is specifed via different parameters, a formatting string, and a list of values. For example, a formatting string of `%d %d %d` will mean that the data should be formatted as three integers:

```
echo | awk '{ printf("%d %d %d",1,2,3) }'
```

The `printf` approach of formatting values comes from the `C` language and has been implemented identically in numerous languages (such as Awk) so when you learn it for one language the same concepts will apply for all others. Here is a fancier printing example:

```
echo | awk '{ printf("A=%0.3f B=%d C=%03d",1,2,3) }'
```

will print:

```
A=1.000  B=2  C=003
```

See more printf examples

# How can I learn more about Awk?

- Awk One-Liners one-liner awk scripts
- Awk One-Liners Explained an explanation for the one-line awk scripts
- Awk CheatSheet - a resource that lists all awk related variables
- Gnu Awk Manual - a comprehensive, but fairly complex resource

# BioAwk

## What is BioAwk?

BioAwk is the brainchild of Heng Li, who after an online discussion where people were complaining about not having an 'awk' that is bioinformatics-aware, decided to take the source of awk and modify it to add the following features:

1. FASTA/FASTQ records are handled as if they were a single line.
2. Added new internal variables for known data formats. For example instead of having to remember that the 6th column is the `CIGAR` string in a sam file a variable `$cigar` will exist when reading SAM files.
3. Added a number of bioinformatics-relevant functions such as `revcomp` to produce the reverse complement (and others).

When bioawk is installed globally you may consult the help with:

```
man bioawk
```

To turn on the special parsing of formats pass the `-c FORMAT` flag where `FORMAT` can be one of the following:

- `sam` , `bed` , `gff` , `vcf` , `fastx`

You can get a quick reminder of the special variable names that bioawk knows about:

```
bioawk -c help
```

it will print:

```
bed:
    1:chrom 2:start 3:end 4:name 5:score 6:strand 7:thickstart 8:thickend 9:rgb 10:blockcount
11:blocksizes 12:blockstarts
sam:
    1:qname 2:flag 3:rname 4:pos 5:mapq 6:cigar 7:rnext 8:pnext 9:tlen 10:seq 11:qual
vcf:
    1:chrom 2:pos 3:id 4:ref 5:alt 6:qual 7:filter 8:info
gff:
    1:seqname 2:source 3:feature 4:start 5:end 6:score 7:filter 8:strand 9:group 10:attribute
fastx:
    1:name 2:seq 3:qual 4:comment
```

What this states is that when you are using the `sam` format you may use variable names such as `qname` , `flag` etc.

For example:

```
cat SRR1972739_1.fastq | bioawk -c fastx ' { print $name } ' | head
```

prints:

```
SRR1972739.1
SRR1972739.2
SRR1972739.3
...
```

to compute the GC content of each sequence:

```
cat SRR1972739_1.fastq | bioawk -c fastx ' { print $name, gc($seq) } ' | head
```

Above we are making use of bioawk specific variables `$name` and `$seq` as well as the bioawk specific function `gc` (GC content) to produce:

```
SRR1972739.1    0.306931
SRR1972739.2    0.49505
SRR1972739.3    0.415842
SRR1972739.4    0.514851
```

# How to use Awk/BioAwk for processing data?

The applications are broad and useful for a wide variety of tasks.

## Find alignments where the CIGAR string has deletions:

```
samtools view -f 2 demo.bam | awk '$6 ~ /D/ { print $6 }' | head
samtools view -f 2 demo.bam | bioawk -c sam '$cigar ~ /D/ { print $cigar }' | head
```

## Select alignments with an edit distance of 3

```
samtools view -f 2 demo.bam | awk '/NM:i:3/ { print $6, $12, $13, $14 }' | head
```

## How many bases have a coverage over 150x?

```
samtools depth demo.bam | awk '$3 > 150 { print $0 }' | wc -l
```

## Mutate one part of the whole line

One really handy feature of awk is that allows mutating a part of the line and printing it out in otherwise original format:

```
samtools depth demo.bam | awk ' $3 <= 3 { $1="LO" } $3 > 3 { $1="HI" } { print $0 }' | head
```

Note how we assign to `$1` but then print `$0`. This is what this does:

```
LO 46 3
HI 47 4
HI 48 4
...
```

This seemingly odd feature is *extremely useful* when the line has many columns and we need to overwrite just one while still keeping it in the same format. Imagine that you had a file with 100 columns and wanted to change just one column. It would be tedious and error prone to piece back together the entire line.

# How to write UNIX scripts

In this chapter we will cover Unix scripts from the point of view of the Bash shell. There are other, newer alternatives notably the FiSH shell.

**Note:** The examples in this chapter assume familiarity with concepts covered later in the book (data access and running tools). If you find yourself confused then it is best to revisit this chapter after covering those sections.

## What is a bash script?

A bash script is a short text file (10-100 lines) that contains a series of bash commands. Conventionally the shell file extension is `.sh` and is run through the shell either with the command:

```
bash myscript.sh
```

or, alternatively if the first line of the shell starts with a so-called "shebang" line `#!/bin/bash` and the program has been made executable with `chmod +x myscript.sh` then the script can be executed just by running it directly:

```
myscript.sh
```

Note that for your system to see and run the script, the same rules apply as with other programs.

## Why are we writing scripts?

The primary advantage of scripts is that they allow us to collect and document related functionality in a single file. Beside listing the commands, we can also explain the reasons and rationale for our work with comments.

Comments are lines starting with the `#` symbol. We recommend making comments short, complete sentences that end with a period.

```
#
# This scripts downloads a dataset identified by a SRR run id.
#
# We're getting only 1000 spots since we're only
# testing this pipeline.
#
fastq-dump -X 10000 --split-files SRR519926

# Run FastQC on the resulting datasets.
fastqc SRR519926_1.fastq SRR519926_2.fastq
```

# Who are we writing the documentation for?

The future us.

When we start a project we are in tune with it and it feels like we will never forget the small and simple decisions we made during the project. This is wishful thinking. We all have better things to do than remember minutiae. Few things are more wasteful and frustrating than having to redo something just because we forgot what and why we did it.

Documentation helps us remember the reasons that we made certain decisions. It is a major productivity boost and lets us get more done in less time.

# What is "refactoring" code?

Refactoring is an iterative process of improving code to reduce its redundancy and make it more generic and simpler. The reason we need refactoring is that typically a process becomes more well understood as we work and solve it. Periodically, we may need to revisit previous steps and rework them to match the later steps.

Refactoring may feel wasteful, as often we modify what may seem to already be working just fine. But in any analysis, complexity can be a hindrance. Refactoring takes on complexity and attempts to simplify our work and will pay dividends in the future.

Refactoring takes some practice and typically the returns are diminishing -- that is, every code can be refactored, but after a few rounds of doing so the benefits are typically much smaller.

# What is the most important improvement I can make to my code?

Move information that can change during running into so-called "variables". This helps you separate the variable sections (those that you might want to change later) from those that should always run the same way.

```
NAME=John
echo Hello ${NAME}!
```

then run it with:

```
bash sayhello.sh
```

Putting the variables first allows you to easily change them, making your code more adaptable.

```
# The selected SRR number.
RUN=SRR519926

# The number of reads to convert.
LIMIT=10000

# Get and convert the data.
fastq-dump -X ${LIMIT} --split-files ${RUN}

# Run FastQC on the resulting datasets.
fastqc ${RUN}_1.fastq ${RUN}_2.fastq
```

# Why is it good to stop a script when an error occurs?

As the saying coined in the Pragmatic Programmer goes: "Dead programs tell no lies." The best course of action on any error is to stop as soon as it happened, because that makes it the easiest to troubleshoot.

By default, shells will continue with the next command even if a previous command failed. Override that default via flags

```
set -u -e -o pipefail`
```

that can actually be shortened to:

```
set -ueo pipefail
```

Add this to every shell script you ever write to make troubleshooting easier.

# How do I add runtime parameters?

The next level of refactoring comes from moving the variable sections of a script outside the script, and having them specified at the command line like so:

```
NAME=$1
echo Hello ${NAME}!
```

and you can run it with:

```
bash sayhello.sh John
```

or

```
bash sayhello.sh Jane
```

As you can see above, this can be achieved by using the magic variables `$1`, `$2` and so on.

Of course, we may want a more complex script where we can specify the RUN ID and the number of reads to convert from it. Suppose we want to run it like:

```
getdata.sh SRR519926 1000
```

and when we do that we get `1000` reads out of run `SRR519926`. Let's write that script:

```
# Downloads an SRR run.
# Converts LIMIT number of spots.
# Assumes the data is in paired-end format.
# Runs FastQC on each resulting FastQ file.

# Stop on any error.
# Dead programs tell no lies!
set -ueo pipefail

# The first parameter is the SRR number.
RUN=$1

# The second parameter is the conversion limit.
LIMIT=$2

# Remind user what is going on.
echo "Rock on! Getting ${LIMIT} spots for ${RUN}"

# Get the data from SRA.
fastq-dump -X ${LIMIT} --split-files ${RUN}

# Run FastQC on the resulting datasets.
fastqc ${RUN}_1.fastq ${RUN}_2.fastq
```

We can now run the script like this:

```
bash getdata.sh SRR1553607 10000
```

# How can I manipulate file paths?

It is important to maintain a consistent naming scheme. This often means we have to manipulate file names within bash to remove the directory name, or to keep only the file name, or to keep only the file name with no extension, or to remove the extension, etc.

Bash offers this functionality via a series of awkward, hard-to-remember constructs that we typically have to look up every time (or Google for):

```
# Suppose this is the full path.
FULL=/data/foo/genome.fasta.tar.gz

# To make it: genome.fasta.tar.gz
NAME=$(basename ${FULL})

# To make it: fasta.tar.gz
EXT1=${FULL#*.}

# To get only the extension: gz
EXT2=${FULL##*.}

# To get the other side: /data/foo/genome.fasta.tar
START=${FULL%.*}
```

When we need a transformation not offered as a single command we can build it by successively applying multiple commands.

# How can I get dynamic data into a variable?

This is a static declaration:

```
NAME=World
echo "Hello $NAME"
```

We can also execute a command and save its results in the variable -- if we enclose it with backticks (` symbols):

```
VALUE=`ls -1 | wc -l`
echo "The number of files is $VALUE"
```

# How can I assign default values to a variable?

To assign default values to variables use the construct:

```
FOO=${VARIABLE:-default}
```

For example, to set the `LIMIT` variable to the first parameter `$1` or `1000` if that was not specified:

```
LIMIT=${1:-1000}
```

# How can I build more complicated scripts?

The best strategy to get started writing scripts is to write them such that instead of executing the commands you echo (print) them out on the screen. Basically put an `echo` command in front of each real command. So instead of:

```
fastqc $SOMETHING
```

write:

```
echo fastq $SOMETHING
```

This will allow you to first SEE what the commands will look like when executed. If it looks right, to execute your commands, pipe them to the `bash` command again.

```
bash myscript.sh | bash
```

Here is a more complicated example that operates on downloaded SRA ids. We'll want a script that takes a project ID and a count that can be run as:

```
bash getproject.sh PRJNA257197 10
```

Our script will fetch 10 sequencing data sets from the project `PRJNA257197`, then run a trimmomatic quality trim on each, and finally run a fastqc report on the resulting files.

First let's get the run info:

```
# Get the run information.
esearch -db sra -query PRJNA257197  | efetch -format runinfo > info.csv

# Cut out the first column, just the RUN ids.
cat info.csv | cut -f 1 -d ',' | grep SRR > ids.csv
```

We now have a file called `info.csv` with numbers in it. We want to use this file to download a lot more information.

Here is our script:

```
#
# Usage: getproject.sh PRJN NUM
#
# Example: bash getproject.sh PRJNA257197 3
#
# This program downloads a NUM number of experiments from an SRA Bioproject
# identified by the PRJN number then runs FastQC quality reports before
# and after trimming the data for quality.

# Immediately stop on errors.
set -ueo pipefail

# The required parameter is the run id.
PRJN=$1

# How many experimental runs to get.
NUM=$2

# What is the conversion limit for each data.
LIMIT=10000

# This is an internal variable that holds the selected ids.
SRA=short.ids

# Keep only the required number of ids.
cat ids.csv | head -${NUM} > $SRA

# Place an echo before each command to see what will get executed when it runs.
cat $SRA | xargs -n 1 -I {} echo fastq-dump --split-files -X ${LIMIT} {}

# Generate the commands for fastqc.
cat $SRA | xargs -n 1 -I {} echo fastqc {}_1.fastq {}_2.fastq

# Generate the commands for trimmomatic.
# Here we run it with the -baseout flag to generatevfile extension of .fq.
cat $SRA | xargs -n 1 -I {} echo trimmomatic PE -baseout {}.fq {}_1.fastq {}_2.fastq SLIDINGWI
NDOW:4:30

# Run FastQC on the new data that matches the *.fq extension.
cat $SRA |  xargs -n 1 -I {} echo fastqc {}_1P.fq {}_2P.fq
```

Here is how you run the script:

```
bash getproject.sh PRJNA257197 1
```

Note -- this script does not execute anything. Instead it generates the commands and prints them on the screen:

```
fastq-dump --split-files -X 10000 SRR1972917
fastqc SRR1972917_1.fastq SRR1972917_2.fastq
trimmomatic PE -baseout SRR1972917.fq SRR1972917_1.fastq SRR1972917_2.fastq SLIDINGWINDOW:4:30
fastqc SRR1972917_1P.fq SRR1972917_2P.fq
...
```

You could now copy-paste each command separately and verify that they work step-by-step. Keep fixing up the script until everything checks out. You can now either remove the `echo` commands or simply pipe the output of the script through bash:

```
bash getproject.sh PRJNA257197 5 | bash
```

The key component to remember is that you want to *see* what gets executed. That will help you troubleshoot any errors that crop up.

For more ideas, see the article Unofficial Bash Strict Mode

# Looping over files

One of the most common needs in bioinformatics is to take a group of files and apply the same command on each. There are several competing ways to do this.

We will demonstrate each by trimming adapters from a paired-end FASTQ file and running FastQC before and after the trimming.

```
fastq-dump --split-files SRR1553607
fastq-dump --split-files SRR519926
```

Generate a file containing our adapter sequence:

```
echo ">illumina" > adapter.fa
echo "AGATCGGAAGAGCACACGTCTGAACTCCAGTCAC" >> adapter.fa
```

In practice, we tend to build our script with a combination of the constructs below (depending on our needs).

# How to loop in bash?

The first method is a looping construct (a so called `for` loop that could be a list of IDs or a file pattern `*.fq` etc):

```
set -ueo pipefail

for FNAME in SRR1553607 SRR519926
do
    echo "FastQC on ${FNAME}"
    fastqc -q ${FNAME}_1.fastq ${FNAME}_2.fastq
    echo "Trimming ${FNAME}"
    trimmomatic PE -baseout ${FNAME}_trimmed.fq ${FNAME}_1.fastq ${FNAME}_2.fastq SLIDINGWINDO
W:4:30 ILLUMINACLIP:adapter.fa:2:30:5
    echo "FastQC on trimmed  ${FNAME}"
    fastqc -q ${FNAME}_trimmed_1P.fq ${FNAME}_trimmed_2P.fq
done
```

**Pros**

1. We list the commands exactly as they look on the command line.
2. We can use any bash construct that we need.
3. We can document every step in as much detail as we need.
4. Bash is widely available on Linux systems.

**Cons**

1. Bash commands are very sensitive to small formatting changes.

2. Bash is not a particularly comprehensive programming language. It is not suited for longer programs.

Writing bash scripts is a good starting point, though.

# How to execute a command on every file that matches a property?

We can locate files with the `find` command then execute a command on each file that we found:

```
find . -name '*fastq' -exec fastqc {} \;
```

**Pros**

1. Very handy when we need to run commands on files that are in multiple locations.
2. `find` has many options for finding files by date, sizes etc.

**Cons**

1. The command is not easy to document.
2. More complex commands will be hard to decipher.
3. Redirecting output with `>` gets complicated.

# How to build custom commands?

We can use the `xargs` command to build other, new commands. `xargs` has the ability to group inputs by the number of parameters `-n` :

```
echo 1 2 3 4 | xargs -n 2 -I {} echo "Hello {}"
```

will print:

```
Hello 1 2
Hello 3 4
```

To run fastqc on each fastq file:

```
ls -1 *.fastq | xargs -n 1 -I {} fastqc {}
```

(note the above is a flag minus one `-1` not an minus ell `-l` ). It lists the files in a short but line-oriented way that makes for good input for `xargs` .

**Pros**

1. Allows more control on how to format parameters for commands.
2. It is simple and available on all Unix systems.

**Cons**

1. Not easy to document.
2. Only five arguments will be substituted! This can be a major annoyance.

```
echo BAA | xargs -n 1 -I {} echo {} {} {} {} {} {}
```

will print:

```
BAA BAA BAA BAA BAA {}
```

# How should I run commands on several identical datasets?

We recommend using GNU `parallel` to run commands on multiple files. This has the added advantage (but watch out for problems!) of also running your code in parallel on all available CPUs. (do `brew install parallel`):

```
ls -1 *.fastq | parallel --verbose --eta  fastqc {}
```

This is the most advanced method of them all. Combine this with creating simple scripts.

**Pros**

1. Runs commands in parallel on each CPU at at time.
2. Offers simple methods for extracting/reformatting file names
3. Can create combinations of parameters automatically.

**Cons**

1. We must account for our programs running in parallel! We must ensure that programs do not overwrite each other's output. This can be surprisingly challenging! Some programs create temporary files without telling us they do!

Read more on GNU Parallel:

- Biostar Post of the day: Gnu Parallel - Parallelize Serial Command Line Programs Without Changing Them
- GNU Parallel Tutorail

# How do I construct commands containing special characters?

In general, when we want to build a command that in turn makes use of bash special characters such as `>` or `|`, we need to ensure that this special character is passed down to the next level intact.

We need to prevent the interpreter from acting on the special characters the first time it sees them. For example, if we wanted 5 files named `1.txt`, `2.txt` etc each containing `Hello 1`, `Hello 2` etc. Neither of the following will work:

```
# Won't do it!
echo 1 2 3 4 | xargs -n 1 -I {} echo "Hello {} >> {}.txt"

# Won't do it!
echo 1 2 3 4 | xargs -n 1 -I {} echo "Hello {}" >> {}.txt
```

What will work is building the command in a single and doubly quoted way and passing that into our own shell instance:

```
echo 1 2 3 4 | xargs -n 1 -I {} bash -c 'echo "Hello {}" > {}.txt'
```

At this point, for readability, one may be better off writing a bash script rather than using `xargs`.

Trying to quote commands properly can be surprisingly challenging. If it seems that one construct just does not seem to fit a certain need, it may be best to approach the problem in a different way.

When the filenames may contain spaces everything becomes a bit more complicated. We need to "null" separate the filenames to be able to process them, then tell `xargs` or `parallel` that the incoming strings are null terminated:

```
find . -name '*.fq' -print0 | parallel -0 "grep ATGC {} > {}.out"
```

# Advanced command line skills

## How do I minimize unwanted file removal?

It is easy to unintentionally remove a file.

To prevent this, add the following into your `~/.bashrc` file to make move, copy or delete ask for permission before removing or overwriting files:

```
alias mv="mv -i"
alias cp="cp -i"
alias rm="rm -i"
```

To temporarily override this check use the `-f` when invoking a command.

## How do I attach a number to each line?

```
cat -n somefile.txt
```

## How can I see how many unique elements are in a column?

The quintessential construct to find out how many of each elements are in column 5 of somefile.txt:

```
cat somefile.txt | cut -f 5 | sort | uniq -c
```

## How can I find files matching a given pattern?

```
find . -name '*.fq'
```

## How do I find particular files and then execute a command on each?

```
find . -name '*.fq' | xargs -n 1 head
```

The previous example won't work if the file names have spaces in them. In that case, we would need to specify extra flags:

```
find . -name '*.fq' -print0 | xargs -0 -n 1 head
```

## How can I run tasks in parallel?

Running tasks in parallel can speed up your computation significantly as each compute core on your system can work at the same time. There are different approaches to parallel computing. Some tools allow you to set the number of threads (usually `-t` or processes `-p` ) as a flag during program invocation. You can also launch several processes at the same time, each on a different file of input data. GNU Parallel can do this for you in the most elegant manner:

```
find . -type f | parallel someprogram {} \> {}.txt
```

Finally there is more advanced parallel programming where programs share memory and other resources. These typically need to be launched in a special way (via a scheduler script) and we won't be discussing them here.

### How can I simplify an NCBI header?

As shown by noirot.celine on Biostars:Renaming Entries In A Fasta File

Here is a generic way to convert an NCBI header to a simpler form

From:

```
>gi|1002620271|ref|NC_029525.1| Coturnix japonica isolate 7356 chromosome 10, Coturnix japonic
a 2.0, whole genome shotgun sequence
TACTCCCCAAGAA
```

to

```
>NC_029525.1
TACTCCCCAAGAA
```

use sed and regular expressions:

```
sed 's/^[^ ]*[|]\([^|]*\)[|] .*$/>\1/' original.fasta > updated.fasta
```

## Should I program in bash?

Even though `bash` is a full fledged programming language, its design originates from days when language design was in its infancy. It is important to learn `bash` since it is a common and useful way to interact with a computer but it is not the best choice to build more complex programs.

Programming languages like: `Perl` and `Python` were developed in part to address the shortcomings of `bash` as a programming environment. We recommend programming in `bash` for automating simple tasks like looping over files or launching programs. For more complex tasks, we recommend `Python` .

## Where to find more information?

Occasionally we need simple constructs to perform a simple action:

- Awk oneliners explained
- Sed oneliners explained
- Bash Programming How To
- Advanced Bash Programming
- Bash by Example from IBM developer works.

# Using Makefiles

Whereas writing scripts is useful and helps immensely, there may soon come a time when you want to keep all related functionality in one file, yet only run a sub-section of it.

The tool named `make` and its corresponding `Makefile` are designed to do that (and a lot more as well). `make` will allow you to set up a veritable *command and control center* for your data analysis from where you will be able to keep track of what you have done and how.

In our opinion the path to reproducible research starts with `Makefiles`. Perhaps at some point in the future when scientists submit their data analysis results, they will also be required to also attach their `Makefile`s that describe the steps they took in a clear and explicit way.

# What is a Makefile?

You can think of makefiles as scripts with "targets".

A `Makefile` is a simple text file that lists commands grouped by so called "targets", which you can think of as labels. We indicate the commands that belong to a rule by indenting them via a TAB character:

```
foo:
    echo Hello John!

bar:
    echo Hello Jane!
    echo Hello Everyone!
```

This `Makefile` has two targets: `foo` and `bar`. And once you create this file you may type:

```
make foo
```

and it will print

```
Hello John!
```

Note how `make` automatically executed the `Makefile` as long as it was called just that. If we wanted to execute a different makefile we would have to specify it like so `make -f othermakefile`. If we were to type:

```
make bar
```

it will print:

```
Hello Jane!
Hello Everyone!
```

And that's it. This one feature of Makefile is sufficient to greatly simplify your life and package your entire workflow into a single file.

For every analysis you should create a single `Makefile` and perhaps other scripts with it. Then you basically have a "central command script" from which you can orchestrate your analysis.

```
make fastqc
make align
make plots
...
```

# Why doesn't my Makefile work?

One common error to check for is neglecting to put tabs in front of the commands. Many text editors insist on inserting a number of space characters even if you press TAB and not SPACE on your keyboard. View the whitespace in your editor.

# Can I use bash shell constructs in a Makefile?

The syntax used inside makefiles may superficially look like a bash shell, but only the commands themselves are passed and executed as a bash shell. There is a fine distinction there - one that you may not ever need to bridge - if you do, remember that a `Makefile` is not a `bash` script. For example, variables in Makefiles are specified the same way as in bash but NOT executed in bash:

```
NAME=Jane
hello:
    echo Hello ${NAME}
```

# Why does my Makefile print every command?

By default `make` echoes every command. This helps you see what it is trying to do. To turn that off add a `@` symbol to the line:

```
NAME=Jane
hello:
    @echo Hello ${NAME}
```

# Why does my Makefile stop on an error?

Remember, that is a good thing. You really want to stop on any error so you can examine and fix it.

In the odd case when you really don't care whether a command succeeded or not and you don't want the make to stop add the `-` sign in front of the command:

```
hello:
    -cp this that
    echo "Done"
```

Will always print "Done" even if the file `this` is not present.

# Why is the first action executed?

If you run `make` with no task label it will execute the first label it sees. It is a good idea to place usage information there so that you can remind yourself of it just by running `make`.

```
NAME=Jane
usage:
    @echo "Usage: make hello, goodbye, ciao"

hello:
    @echo Hello ${NAME}
```

# Is there more to Makefiles?

We only covered the basic features of `make`, the ones that we believe will give you the most benefits with the least amount of overhead.

Rest assured that `make` has several other very handy features. One important feature is that it can track the so-called dependencies between files. It can be told to automatically detect which sections need to be re-run when some of the inputs change.

In our personal observation these features of `make` are only useful when data analysis is performed at very large scale. In our typical work we almost never need to specify these dependencies. That keeps the use of `make` a lot simpler. But we encourage you to explore and learn more about `make`. A good start would be the document titled Automation and Make on Software Carpentry.

But remember - the simplest feature that you learned above already saves you a lot of effort.

# Are there are alternatives to Makefiles?

The concept of `make` goes back a long way and in time many alternatives have been proposed. One that seems to have taken off with bioinformaticians is snakemake If you find yourself having to develop overly complex Makefiles it might be worth exploring the alternatives. On our end we get a lot done with simple and uncomplicated Makefiles.

# Visualizing data

For all the algorithmic power at our disposal, visually inspecting data has been and will always be an essential ingredient to scientific discovery. The human eye has pattern recognition abilities that are unmatched by any computer. Our minds are the most powerful analytic instrument that we have access to. We are just not fast enough and we're just not physically capable of processing all the data at our disposal.

If we could, we would be able analyze data far better than any computer. Keep this in mind -- eventually you'll disagree with a computed result -- and when that happens, it may be that you are correct and the computer is wrong.

## What are the challenges of visualization?

It is essential to recognize that visualizing data so that it displays the attributes that are interesting to us at any given moment is a far more complicated task than we may realize. A meaningful visualization is one that shows us *only* what we are interested in at any give moment -- but that need changes over time.

For this reason, building useful visualization software is far more challenging than it may seem at first -- and it is more difficult than building tools that have objective measures (speed/accuracy etc) of quality.

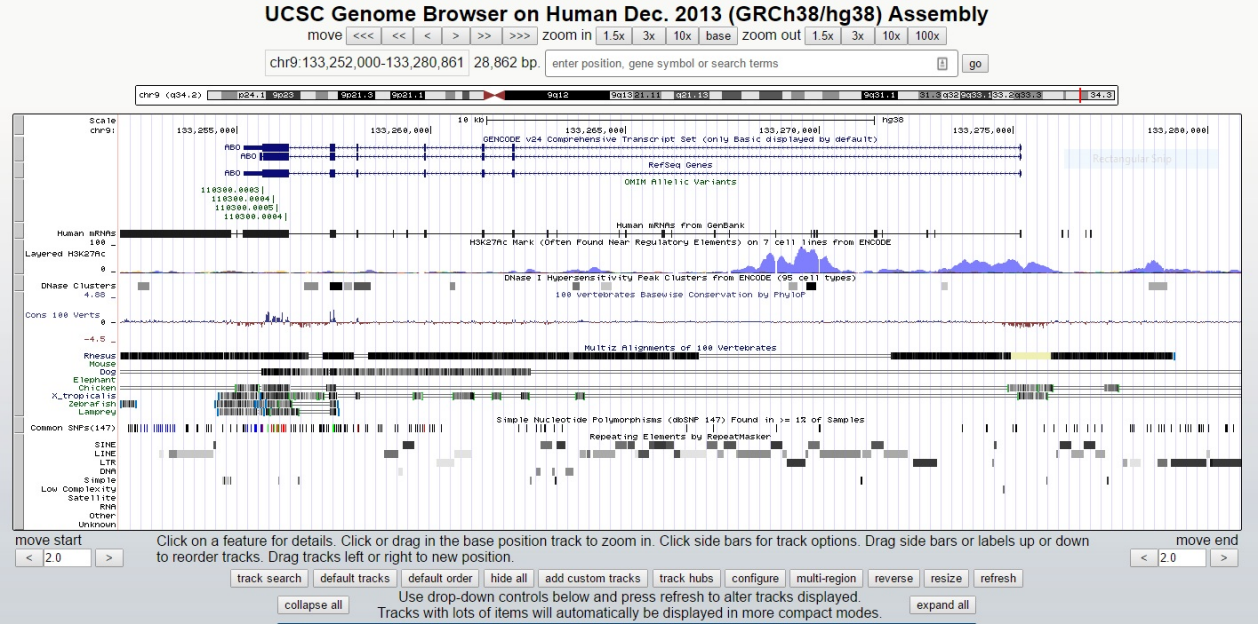
No wonder that biological visualization tools have a "graveyard" that is larger than most other tools. Many have tried to build a better, simpler and nicer visualization approach, and many have failed to deliver a long term solution. Though it is fair to say, often through no fault of their own.

## What is a genome browser?

A genome browser is a graphical interface to display information from a database of genomic data. Most genome browsers draw linear tracks that represent the forward strand of the genome from its 5' (left) towards the 3' (right) direction. The genomic features are drawn over this linear track via so-called **glyphs**.

A glyph is a pictogram that corresponds to a particular genomic feature.

There is no universal standard that all visualization tools adhere to. Interpreting glyphs comes with experience and a level of familiarity with each tool. For example, here is a default screenshot of the UCSC genome browswer:




# Why are default browser screens so complicated?

Visit any genome browser site and you may be overwhelmed by the information that it shows by default. Most of this information may not be relevant to you. Unfortunately, scientists are not always that great at creating simple and usable services, and many operate under the mistaken assumption that "more" must be "better".

Typically, there is nothing special about the data that a browser shows you by default. Note how in the tracks above some features of the elephant genome are shown without us having ever asked for it. We mean no disrespect to you, elephant, but you are not that special! Perhaps it is the lead developer's (or their boss') favorite data and they operate under the assumption that everyone wants to see the same.

Needless to say -- the arbitrary initial complexity of most browsers makes their use more difficult than necessary.

# What types of data may be visualized in a genome browser?

In most cases only data in simple line oriented formats works: FASTA, BED, GFF, SAM/BAM.

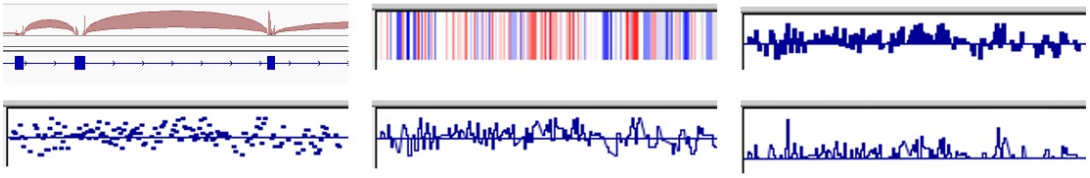
Data formats that contain more complex information such as GenBank or EMBL typically need to be transformed and simplified into a line oriented format such as BED or GFF.

Most visualization tools come with some data pre-loaded, and with different genomic builds for widely used organisms: human, mouse, fruit fly, etc. That means that data for many genomic features will likely already be present. For other organisms we may need to build our own custom genome and add our own custom annotations to the browser.

# How do I interpret glyphs?

Glyphs are a visual representation of some genomic features:

- Horizontal intervals: directions, genes, alignments
- Values over intervals: coverages, probabilities
- Attributes at locations: mutations, deletions, junctions



What you will find is that often it is not possible to locate a precise definition for the reasons that a feature is drawn a specificc way.

For example, why is one linear feature drawn thicker than the other? Probably because it is labeled differently. Perhaps it is a coding sequence and this tool chose to draw those differently. Alas, this information may not always be present or easy to find in the tool's documentation. Often you need to rely on intuition, some detective work, and your own interpretation.

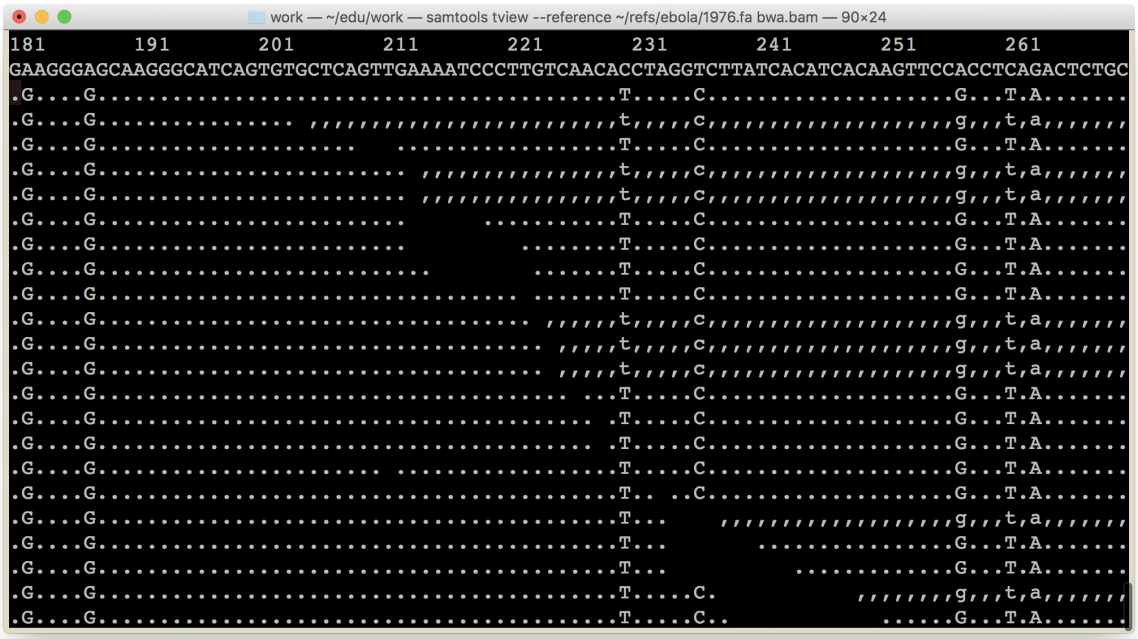
The take home message here is that behind every visual feature that you see, no matter how small, it is the visualizer trying to tell you something more. For example, is an alignment outline of an ever-so-slightly different color? Perhaps it is a chimeric read - and this is how this visualizer shows those.

# What is the simplest genome browser?

It is called `samtools tview` (text view). You already have it installed via samtools. We can visualize the BAM file created in the section Filtering SAM files with:

```
samtools tview --reference $REF bwa.bam
```

It will produce the image:




A few rules for interpreting and working with `tview`:

- Sequencing reads are displayed in a line.
- Press `?` for help, `q` or `ESC` to exit a screen.
- Space, backspace moves the screen left and right.
- The letters are in the so-called 'pileup' format
- `.` means a match on forward strand, `,` means a match on reverse strand.
- A capital letter like `T` means a substitution of T of a read mapped to the forward strand, a lowercase like letter `t` means a change in the read mapped to the reverse strand.

Samtools `tview` also works for remote files, here we go to chromosome 8, position 17,375,390:

```
samtools tview http://cdna.eva.mpg.de/denisova/alignments/T_hg19_1000g.bam -p 8:17,375,390
```

In this case, we did not pass a reference genome to it, so instead of showing `.` and `,` the full base composition of the reads is shown directly. It makes for a view that is more difficult to interpret:

# Which standalone genome browsers should I use?

A surprising number of standalone genome visualization tools become abandoned within a few years of their announcement.

We list those standalone applications that have withstood the test of time and appear to have decent institutional support:

- IGV,Integrative Genomics Viewer by the Broad Institute.
- IGB, Integrated Genome Browser from the University of North Caroline, Charlotte
- Artemis by the Wellcome Trust Sanger Institute. One of the few browsers that can handle GenBank and EMBL formats!
- Tablet from the James Hutton Institute
- SeqMonk A tool to visualise and analyse high throughput RNA-Seq alignments.
- iobio real-time genomic analysis

# What about online genome browsers?

- UCSC Genome/Table Browser from the University of California at Santa Cruz,
- Ensembl Genome Browser from the partnership of the European Bioinformatics Institute and the Wellcome Trust Sanger Institute

For each of these services it is possible to display your own data within the context of the data that is stored within that service. See next chapters for more detail.

It should be noted here, while NCBI offers many facilities to visualize data that is already deposited into the service, it offers few options to visualize our own data within the same context.

## Is it possible to run your own genome browser service?

There is software to do that, but setting it up and running it as a service is not particularly easy.

- GBrowse: Generic Genome Browser a combination of database and interactive web pages for manipulating and displaying annotations on genomes.
- JBrowse an embeddable genome browser built completely with JavaScript and HTML5, with optional run-once data formatting tools written in Perl.
- GBiB: Genome Browser in a box is a "virtual machine" of the entire UCSC Genome Browser website that is designed to run on most PCs. An alternative to look into may be "track hubs".

Many organizations built around model organisms use these the browsers: SGD, FlyBase, WormBase as well as many others.

# The Integrative Genomics Viewer

NOTE: The usage instructions for IGV and all graphical programs are being kept short intentionally.

In our opinion, when presenting graphical user interfaces, describing steps in their annoying minutiae (click here, then click there) have a questionable effectiveness. Step by step instructions that need to be followed blindly take away your ability to *understand* what is going on.

Different people may begin from different starting points and they may get stuck at different stages. To succeed you need a conceptual understanding of what you are trying to accomplish rather than a click-by-click handholding.

What we will state in this book are possible actions and we can assist you identify these actions. But you yourself have to explore and use the interface in order to successfully perform them.

Try actions:

- Explore the menu
- Right click on glyphs, windows and tabs.
- Try to drag and drop tracks or features.

## What is IGV?

IGV, Integrative Genomics Viewer by the Broad Institute is both a visualizer and a data integration platform where your data can be displayed in the context of other information.

## How to install and run IGV?

You can download and double click on IGV -- it is simple to run it that way and initially you might use it that way.

In our experience this is not the best way to run it. With large datasets, the tool may get bogged down. It can hang, and its error recovery is impacted. We have had much better luck downloading and running the binary distribution of IGV. After unpacking it into `src` we can run the shell script from the command line:

```
bash ~/src/IGV_2.3.85/igv.sh
```

Then minimize this window and come back to it only if there is trouble.

## What does the IGV interface look like?

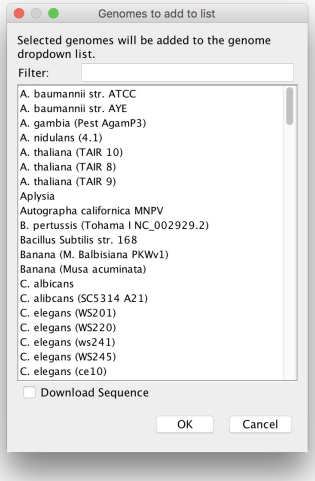
Most visualization tools (and IGV is no different) come with data preloaded, with different genomic builds of widely used organisms: human, mouse, fruit-fly etc. That means that some information on your organism may already be present. For all other organisms we typically build our own custom genome and add our

own custom annotations.

Note the colors, the orientations, some reads have different outlines, some regions have different colors. All of these are IGV specific **glyphs** by which IGV is trying to display information about the data.

# What data does IGV come with?

The Broad Institute distributes data for IGV from a wide number of organisms and data sources. Do make note -- to be able to display their data, your nomenclature for chromosomes has to match theirs (more on this in the *What is in a name* (todo) chapter):




# How do I create a custom genome in IGV?

To operate on a genome file that is not preloaded with IGV we need to build it ourselves. This will typically require the following:

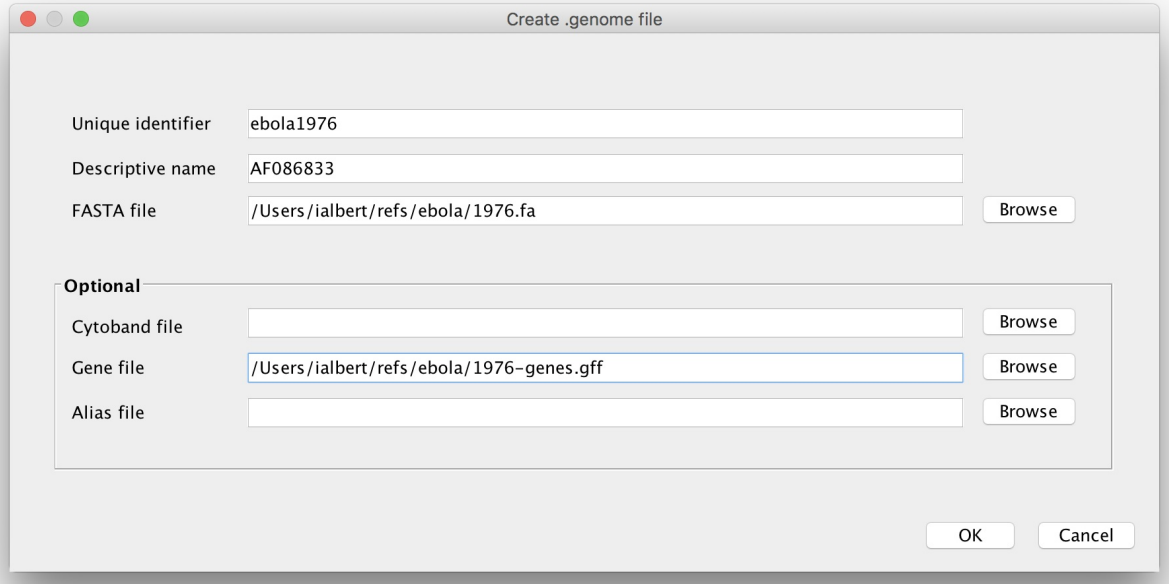
1. Identify the genome file and optionally a feature file you want to use.
2. Instruct IGV to process the genome and feature to create an IGV formatted genome out of them.
3. Load the new genome file when needed.

How to make a gene file:

```
# Get a genbank file. Unfortunately we cannot load that into IGV.
# It needs to be formatted into GFF.
efetch -db nucleotide -format=gb -id=AF086833 > AF086833.gb

# Convert that GenBank file into GFF3.
# This will contain all features.
cat AF086833.gb | seqret -filter -feature -osformat gff3 > AF086833.gff
```

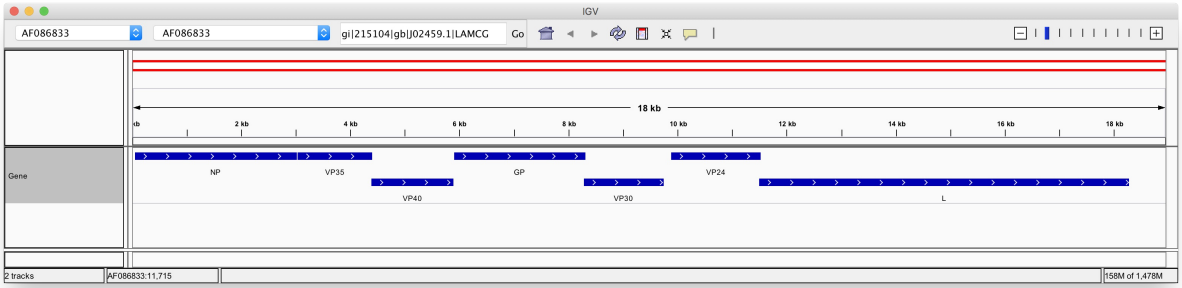
If we wanted to keep only gene lines from the GFF file:

```
# The first line of the gff file tells the version.
head -1 AF086833.gff >> 1976-genes.gff

# Keep only lines where the third column is gene.
cat AF086833.gb | seqret -filter -feature -osformat gff3 | awk ' $3 ~ /gene/ { print $0 }' >>
1976-genes.gff

# Copy these to the reference genome folder.
cp AF086833.gff 1976.gff
cp 1976-genes.gff 1976.gff ~/refs/ebola
```



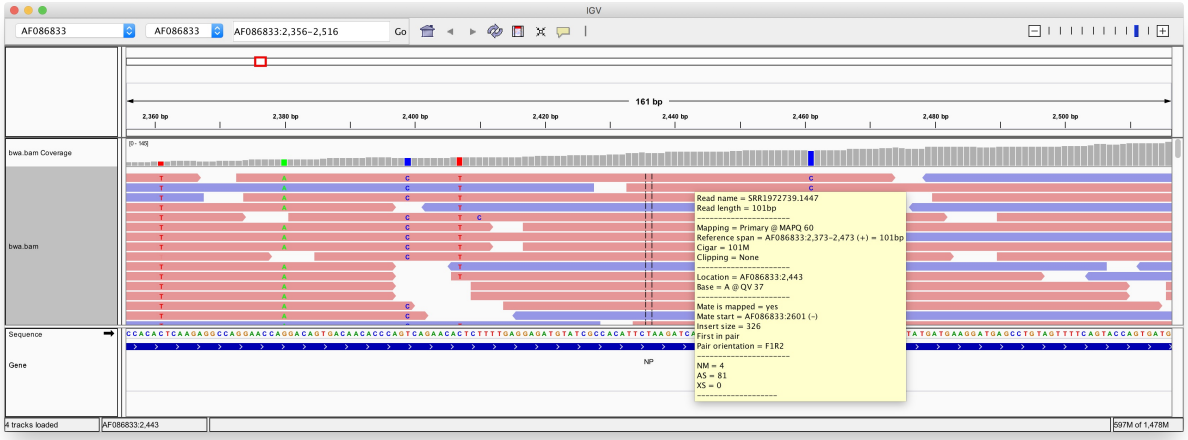

It will ask you where to save the genome file. Set this to the same location where your reference files already are.

Keep at it and you'll eventually succeed. Read the manual, explore the options, click on menus and importantly right click on fields and surfaces to see what type of contextual options are being offered. Once the genomes are built, you may drag you GFF file into the main panel or open it via the `File` menu. Success looks like this:




# How do I visualize alignments?

We can visualize the BAM file created in the section Filtering SAM files by loading it up into IGV:




To learn the interface you will need to explore it for yourself:

- Right click panels to see all your visualization options.
- Read the manual to see why a feature is drawn a certain way. Most visualizers make their own decisions of what and gets visualized and why.
- Visualizers can be quirky. They may crash or hang on you. Typically this happens when the data is invalid in some manner or if they run out of memory.
- IGV shows the locations where our alignment data matches or mismatches the genome that we aligned it against.

Each one of your aligned sequences in the FASTQ file will have a graphical representation on the interface. Try to find the first SAM record in the visual diplay.

# Can IGV work on remote files?

Oh, yes. It is one its best features. Alignment files on remote servers may behave the same way as they would on a local computer. Here is how this works in practice. A published paper states:

"The Denisovans are a Paleolithic-Era members of the genus Homo that may belong to a previously unknown species. The genome sequence of a Denisovan individual which lived about 41,000 years ago was generated from a small fragment of a finger bone discovered in Denisova Cave in southern Siberia in 2008."

The alignment file is available at

- http://cdna.eva.mpg.de/denisova/alignments/T_hg19_1000g.bam

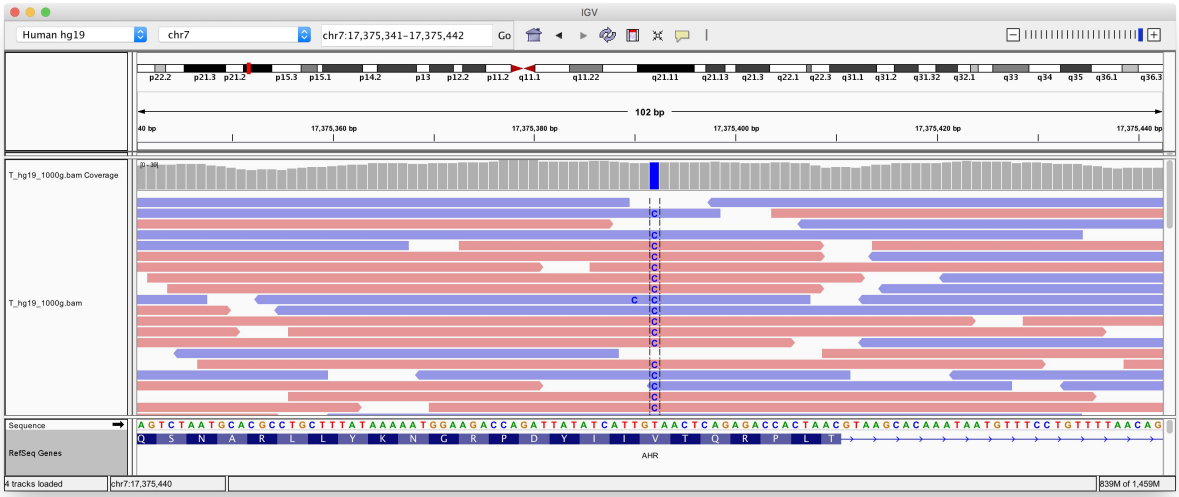
We also know that one exon of AHR, the Aryl Hydrocarbon Receptor is encoded on chromosome 7 and that one particular location of this gene at position `17,375,392` has been known to be associated with smoke sensitivity in mice:

- http://www.genecards.org/cgi-bin/carddisp.pl?gene=AHR

Homo sapiens has a Valine translated from this location. What amino acid would the protein for this Denisovan individual have?

Thankfully we would not need to download the whole BAM file (80GB) just to investigate our question. We can copy the BAM file's address only (Copy link address) then paste that into the "Open -> Load From Url" menu item of IGV.

This way, we can quickly and easily load up the data in IGV and navigate to the location of interest to see that a base change at position `17,375,392` changed the codon `GTA` (Valine) to `GCA` (Alanine) -- indicating that the Denisovans may have been more sensitive to smoke than modern humans. Suffice to say a sensitivity to smoke is not desirable when you need to live in a cave...



# Data simulation

Simulating data allows us to evaluate the performance of our tools and techniques. By knowing what the data contains, we can verify that the tools do indeed operate as we expect them to.

We recommend that you explore tools with simulated data first to see the strengths and weaknesses of various methods. While these type of tools are essential for learning data analysis, this domain of bioinformatics is unfortunately filled with neglected and abandoned software. It is not easy to find a working read simulator. Below are our recommendations.

## What types of data simulators exist?

There are simulators available for whole genome, exome, rna-seq and chip-seq data simulation. Some tools run from command line, others have graphical user interfaces.

- 'wgsim/dwgsim' simulates obtaining sequencing reads from whole genomes.
- 'msbar' EMBOSS tool can simulate mutations in a single sequence.
- 'biosed' EMBOSS tool provides control on what mutations we introduce.
- 'ReadSim' is a sequencing read simulator to target long reads such as PacBio or Nanopore.
- 'Art' is the most sophisticated and up to date simulator that attempts to mimic the errors that sequencing instruments themselves produce.
- `Metasim` simulates reads from metagenomes (from a diverse taxonomical composition present in different abundances).
- `Polyester` an RNA-Seq measurement simulator

## Are the simulations realistic?

This depends on what the simulator aims to capture. Some aspects of a realistic process are easier to reproduce, but others like various biases and fragmentation errors are not.

In general, data simulators are best suited for exploring how various events may manifest themselves within sequencing data. For example: what kind of data will a "large scale inversion" generate and what will that data look like when aligned against the non-inverted reference genome. We will demonstrate a number of examples of this type.

## What is `wgsim` and `dwgsim`?

`wgsim` and `dwgsim` are tools for simulating sequence reads from a reference genome. The tools are able to simulate diploid genomes with SNPs and insertion/deletion (INDEL) polymorphisms, and simulate reads with uniform substitution sequencing errors. While the tools do not generate INDEL sequencing errors, this can be partly compensated by simulating INDEL polymorphisms.

`wgsim` is easier to install but it only produces the mutations in text format and that is harder to visualize. `dwgsim` on the other hand will produce VCF files in addition to the text file.

- See: Whole Genome Simulator wgsim/dwgsim

# How do I simulate sequencing data with `dwgsim`?

Install `dwgsim` as instructed in the tool setup:

```
# Get a reference genome.
efetch -db=nuccore -format=fasta -id=AF086833 > genome.fa

# Simulate sequencing reads from the reference genome.
dwgsim genome.fa data
```

This will create a number of files:

- First in pair: `data.bwa.read1.fastq`
- Second in pair: `data.bwa.read2.fastq`
- Mutations as text file: `data.mutations.txt`
- Mutations as a VCF file: `data.mutations.vcf`

You may align the paired-end reads against a genome to explore aligner behavior.

# How do I mutate a sequence with `msbar`?

The `msbar` EMBOSS tool was designed to generate mutations in sequences. It can produce point mutations with the flag:

```
-point Types of point mutations to perform
       0 (None);
       1 (Any of the following)
       2 (Insertions)
       3 (Deletions)
       4 (Changes)
       5 (Duplications)
       6 (Moves)
```

Or block mutations with the flags

```
-block Types of block mutations to perform
       0 (None)
       1 (Any of the following)
       2 (Insertions)
       3 (Deletions)
       4 (Changes)
       5 (Duplications)
       6 (Moves)
```

Block mutations are a series of consecutive mutations that span from ranges specified with `-minimum` and `-maximum` .

Get a 50bp query from the reference file:

```
# Get a reference genome.
efetch -db=nuccore -format=fasta -id=AF086833 > genome.fa
cat genome.fa | seqret -filter -send 50  > query.fa

# Introduce a point deletion.
cat query.fa | msbar -filter -point 3 > mutated.fa

# Visualize the output.
global-align.sh query.fa mutated.fa
```

on our system this produced:

```
CGGACACACAAAAAGAAAGAAGAATTTTTAGGATCTTTTGTGTGCGAATA
|||||||||||||||||||||||||||||| ||||||||||||||||||
CGGACACACAAAAAGAAAGAAGAATTTTTA-GATCTTTTGTGTGCGAATA
```

The same with a 2 base block mutation:

```
cat query.fa | msbar -filter -block 3 -minimum 2 -maximum 2 > mutated.fa
```

shows up as:

```
CGGACACACAAAAAGAAAGAAGAATTTTTAGGATCTTTTGTGTGCGAATA
|||||||||||||||||||||||||||||||||||||||||||||||  |||||
CGGACACACAAAAAGAAAGAAGAATTTTTAGGATCTTTTGTGT--GAATA
```

We can introduce more than one mutation with the `-count` parameter:

```
cat query.fa | msbar -filter -count 2 -block 3 -minimum 2 -maximum 2 > mutated.fa
```

one result could be:

```
CGGACACACAAAAAGAAAGAAGAATTTTTAGGATCTTTTGTGTGCGAATA
|||||||||||||  ||||||||||||||||||  ||||||||||||||||
CGGACACACAAAA--AAAGAAGAATTTTTAGGA--TTTTGTGTGCGAATA
```

Since the tool selects mutations randomly, when choosing mutations like substitutions it is possible to mutate the sequence back to itself i.e. replacing an `A` with another `A`

One flaw of the `msbar` tool is that it does not tell us what changes it has made. Hence, it is hard to use it in a context of validating a variant caller.

# How do I simulate mutations with `biosed` ?

```
# Get a reference genome.
efetch -db=nuccore -format=fasta -id=AF086833 | head -3 > genome.fa
```

The file contains:

```
>AF086833.2 Ebola virus - Mayinga, Zaire, 1976, complete genome
CGGACACACAAAAAGAAAGAAGAATTTTTAGGATCTTTTGTGTGCGAATAACTATGAGGAAGATTAATAA
TTTTCCTCTCATTGAAATTTATATCGGAATTTAAATTGAAATTGTTACTGTAATCACACCTGGTTTGTTT
```

Now replace the first ten bases with `biosed` as:

```
cat genome.fa | biosed -filter -target CGGACACACA -replace GGGGGGGGGG
```

produces:

```
>AF086833.2 Ebola virus - Mayinga, Zaire, 1976, complete genome
GGGGGGGGGGAAAAGAAAGAAGAATTTTTAGGATCTTTTGTGTGCGAATAACTATGAGGA
AGATTAATAATTTTCCTCTCATTGAAATTTATATCGGAATTTAAATTGAAATTGTTACTG
TAATCACACCTGGTTTGTTT
```

It also will reformat the fasta file. How about that...

# How do I simulate reads `with` readsim`?

Manual: https://sourceforge.net/p/readsim/wiki/manual/

The program will only work with Python 2.

```
# Get a reference genome.
efetch -db=nuccore -format=fasta -id=AF086833 > genome.fa

# Simulate pacbio technology reads into a file named reads.fq
readsim.py sim fq --ref genome.fa --pre reads --tech pacbio
```

# How do I simulate reads with `art` ?

Manual: http://www.niehs.nih.gov/research/resources/software/biostatistics/art/

The `Art` suite is the most sophisticated simulator that we know of.

```
art_illumina -ss HS25 -sam -i genome.fa -p -l 150 -f 20 -m 200 -s 10 -o reads
```

The output contains even an "optimal" SAM alignment file for the simulated reads that we can compare against:

```
Quality Profile(s)
    First Read:   HiSeq 2500 Length 150 R1 (built-in profile)
    First Read:   HiSeq 2500 Length 150 R2 (built-in profile)

Output files

  FASTQ Sequence Files:
     the 1st reads: reads1.fq
     the 2nd reads: reads2.fq

  ALN Alignment Files:
     the 1st reads: reads1.aln
     the 2nd reads: reads2.aln

  SAM Alignment File:
    reads.sam
```

# Visualizing genomic variations

One of the most unexpectedly challenging tasks of a bioinformatics visualization is correctly interpreting the effects that large-scale genomic variations have on alignments.

The confusion starts right at the terminology -- we call one of our sequences the "reference" and we compare our data to it. The word "reference" implies that we are comparing against some sort of "ground truth" -- but it is the other way around.

Our data is the reality, and the reference is a "mirror" in which we try to look at it. But just as with a mirror, what we see often shows the opposite of what we expect.

To run the software for this chapter prepare some datasets first:

```
mkdir -p refs
efetch -db=nuccore -format=fasta -id=AF086833 | seqret --filter -sid AF086833 > refs/AF086833.
fa
bwa index refs/AF086833.fa
```

See also the data site: http://data.biostarhandbook.com/scripts/

# Which alignments are informative?

When it comes to structural variations the main rule to remember:

> Alignments that **cross over** the variation carry the most information on the exact nature of the change.

Alignments that are fully contained within a continuous segment do not know about what is "outside" of that segment, and hence are less informative. This is why paired-end sequencing is so useful. It does two things for us:

1. It performs the alignment over a longer distance.
2. The distance and orientation between read pairs can indicate the type and direction of variation.

It is also worth noting that typically the vast majority of alignments are outside of these so called junction points -- hence, when large scale genomic variation is under study, the genome may need to be covered at higher rates.

# What would perfect data look like?

Ask any bioinformatician what ideal, "perfect" data should look like - chances are you'll get very different answers. We practitioners never see ideal data -- we don't quite know how to even define it. For this book we wrote a simple Python script called the `perfect_coverage.py` that simulates sequencing every 500bp long fragment of a genome exactly once, with no errors, with an Illumina paired end protocol.

```
# Get the script
curl -O http://data.biostarhandbook.com/align/perfect_coverage.py

# Get some data.
cat refs/AF086833.fa | python perfect_coverage.py
```

This produces the files `R1.fq` and `R2.fq` . Align these reads with.

```
bwa mem refs/AF086833.fa R1.fq R2.fq | samtools sort > perfect.bam
samtools index perfect.bam
```

Visualize the output in IGV. This is what we saw:

Note the coverages at the beginning and end of the genome. These are called *edge effects* and will show up even in ideal data. In essence, even in ideal conditions there will be less coverage at the ends of a sequence, because fewer reads can possibly overlap. The range where the effect will show up is as long as the fragment size -- 500bp in this case.

Since a chromosome has only two ends and the fragments are typically much smaller than the sequence itself these artifacts are typically ignored. But that for an analysis that measures lots of short sequences (like an RNA-Seq experiment) instead of a few, there may be tens of hundreds of thousands of transcripts, and each transcript will have its ends affected. Moreover, transcripts have varying lengths whereas the edge effect is fragment size specific. Hence, shorter transcripts will be more affected and will get less coverage than expected.

Dealing with edge effects in these cases is more complicated, often solved by introducing a new concept called "effective length normalization" -- basically treating all sequences as if they were shorter than they are in reality. We'll discuss this further in the RNA-seq section. It is important to note that the effect is present even in this idealized case.

# What would realistic and good data look like?

The data simulation methods are explained in more detail in How to simulate data. For the rest of the chapter we repeat the following.

1. We will make a copy of our reference file. We will call this the `GENOME.fa` and we will use it as the source of our simulations.

2. We will modify our genome file, generate reads from it, and compare it to the reference.

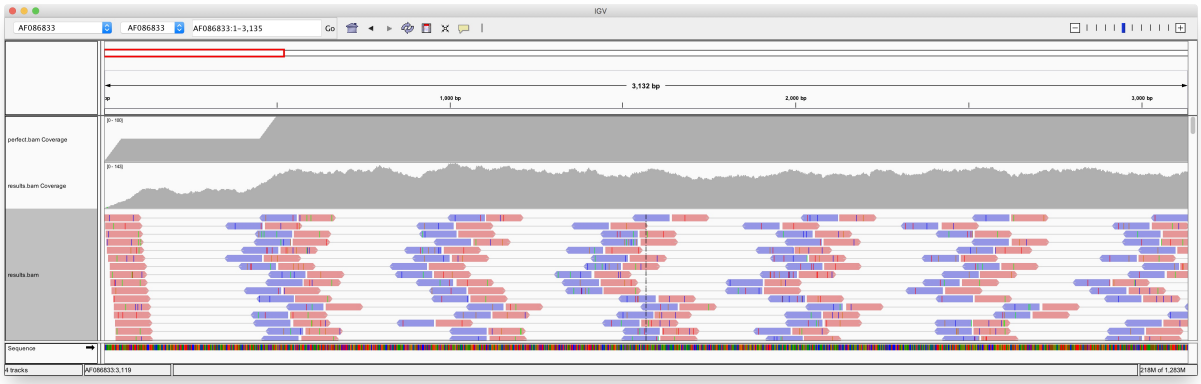
Initially, the `REFERENCE.fa` and `GENOME.fa` files are the same. We will mimic evolution by modifying `GENOME.fa`, then we will align data obtained from it against `REFERENCE.fa`. The process of simulating and aligning can be done step-by-step, but we wrote a script to automate it greatly. Now remember, this script does not mutate the genome, it just simulates reads from it and aligns these reads against the reference.

```
curl -O http://data.biostarhandbook.com/align/simulate-experimental-data.sh
```

This script contains a series of commands that simulate data from the file called `GENOME.fa` and align this data against `REFERENCE.fa`. Run the script:

```
bash simulate-experimental-data.sh
```

It will generate a file called `align.bam` that, when visualized, will look like the following (we kept the track for the idealized data for comparison):



This is more realistic data than our perfect alignment. We can observe the stochasticity of the coverage.

# What would a deletion from the genome look like?

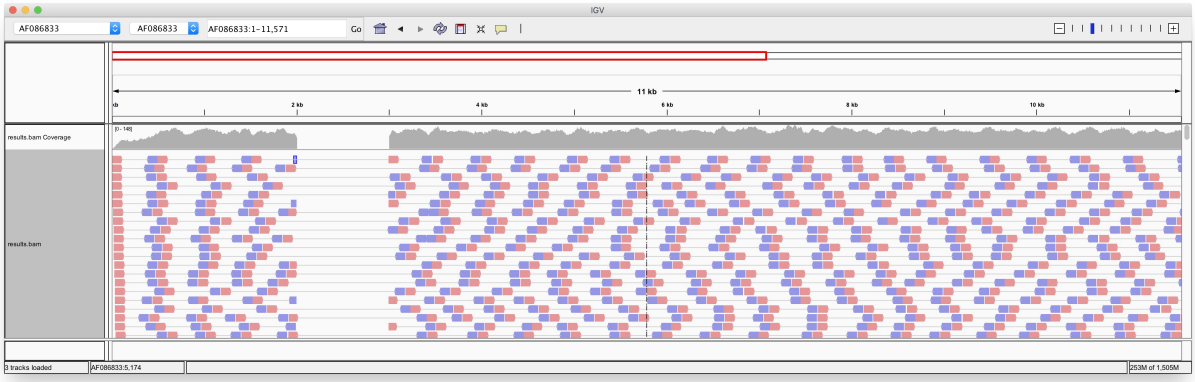
Open the `GENOME.fa` file in a text editor and delete a big chunk of it. You could also perform that from command line like so:

```
cat refs/REFERENCE.fa | seqret --filter -sbegin 1 -send 2000 > part1
cat refs/REFERENCE.fa | seqret --filter -sbegin 3000 > part2
cat part1 part2 | union -filter  > GENOME.fa
```

then rerun the simulator.

```
bash simulate-experimental-data.sh
```

Here is what we get:



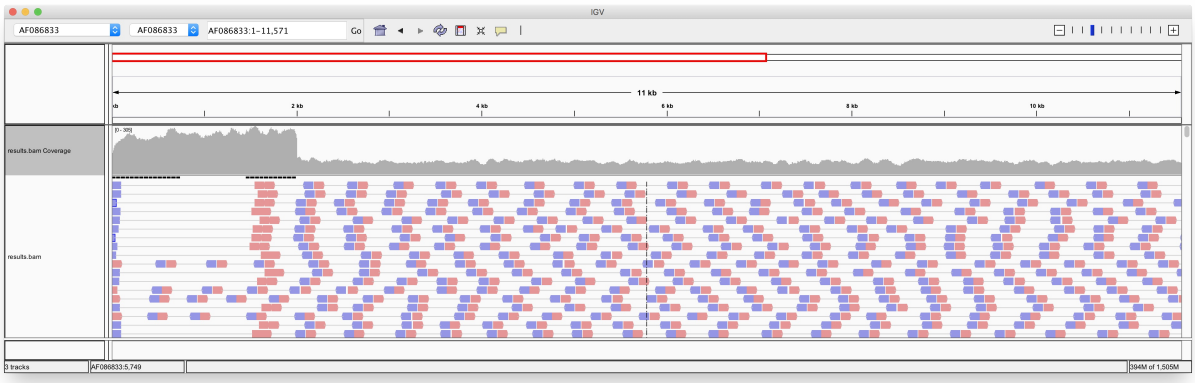

This image makes a lot of sense. The parts that we deleted are missing. But note something else: there are read pairs that start on one end and continue on the others. What this means that the deletion was smaller than the fragment size. A fragment that comes from just the right location can "bridge" over the deleted portion. If the deleted portion is larger than the fragment size then no bridge will ever occur.

# What would copy number variation look like?

For this, add half of the beginning of the genome back to the front of it. Basically imagine the genome is `ABCDEFGH` then make it `ABCDABCDEFGH`. Again you can do this in an editor or from command line with:

```
cat refs/REFERENCE.fa | seqret --filter -sbegin 1 -send 2000 > part1
cat refs/REFERENCE.fa | seqret --filter -sbegin 1 -send 2000 > part2
cat part1 part2 refs/REFERENCE.fa | union -filter  > GENOME.fa
```

These events are called duplications or more generically copy number variation.

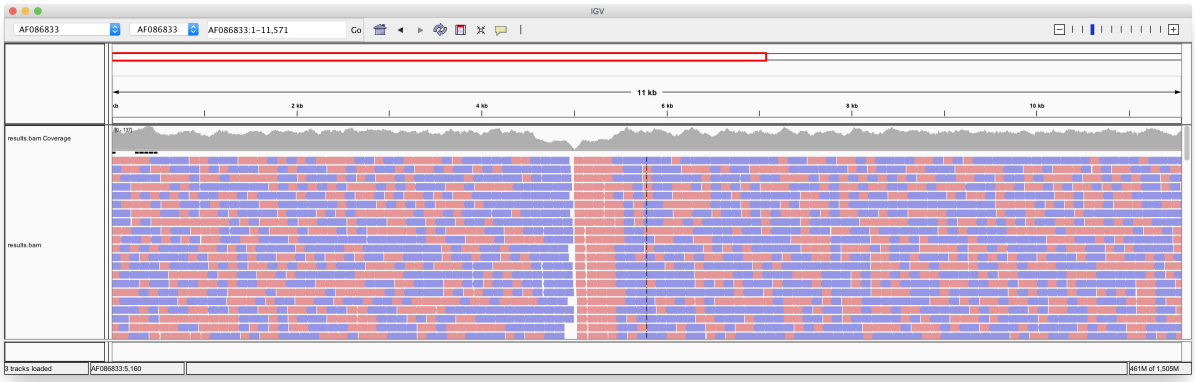



# How can you tell when parts of the genome are swapped?

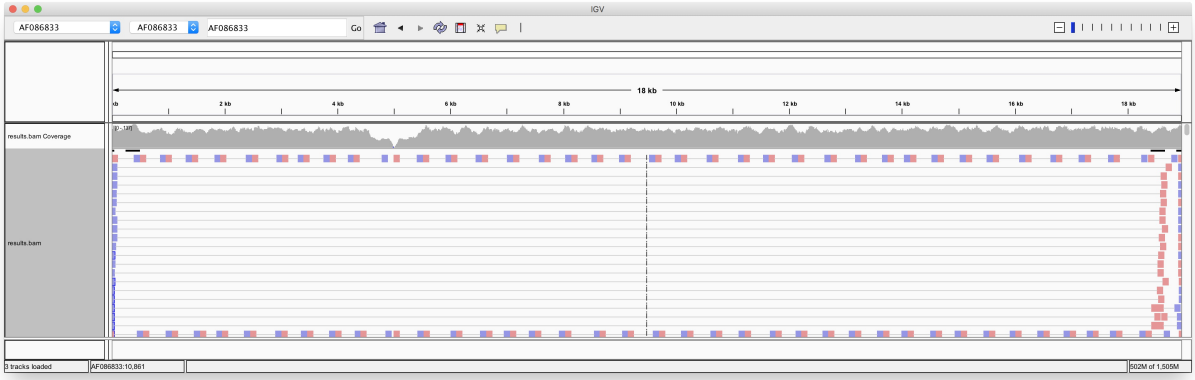
Now we modify our genome to go from `ABCDEFGH` to `EFGHABCD`.

```
cat refs/REFERENCE.fa | seqret --filter -send 5000 > part1
cat refs/REFERENCE.fa | seqret --filter -sbegin 5000 > part2
cat part2 part1 | union -filter  > GENOME.fa
```

How will this data align? The image that we see has more subtle changes:




It is only when we fully expand the pairs that we start to understand the complexity of it all.



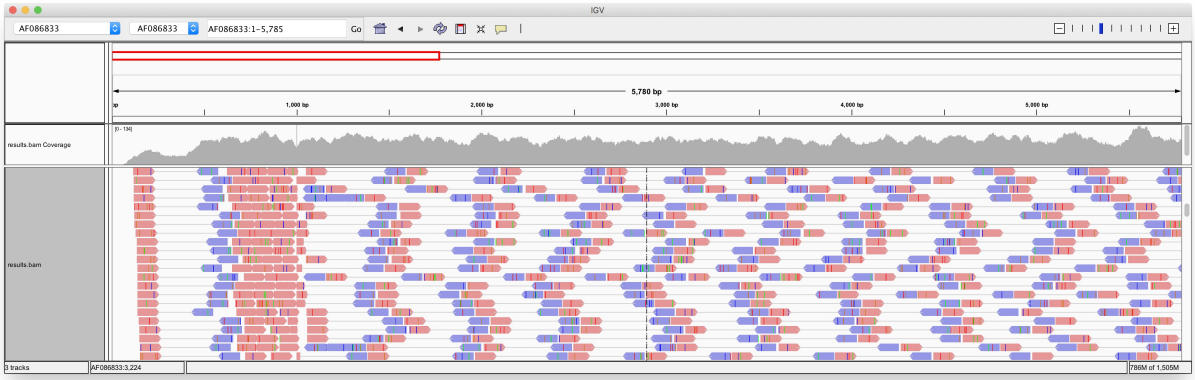

# What if the genome contains novel regions?

This would turn our genome from `ABCDEFGH` to `ABCDWWWWEFGG`. How will this show up in our data? Let's insert a sequence of a different virus in this case HIV-1 into our genome

```
cat refs/REFERENCE.fa | seqret --filter -send 1000 > part1
efetch -db nuccore -id NC_001802.1 -format fasta -seq_stop 1000 > middle
cat refs/REFERENCE.fa | seqret --filter -sbegin 1000 > part2
cat part1 middle part2 | union -filter  > GENOME.fa
```

The read pairs that come from the novel genome will not align anywhere. They will be lost. For read pairs where one of the pairs comes from the known genome and the other from the novel genome the matching one will be aligned and the mate lost. These are called "orphan" alignments.

In addition, reads falling over the junctions may be soft-clipped all at the same coordinate. When we look at the alignment in paired end mode a large number of orphan reads show up:

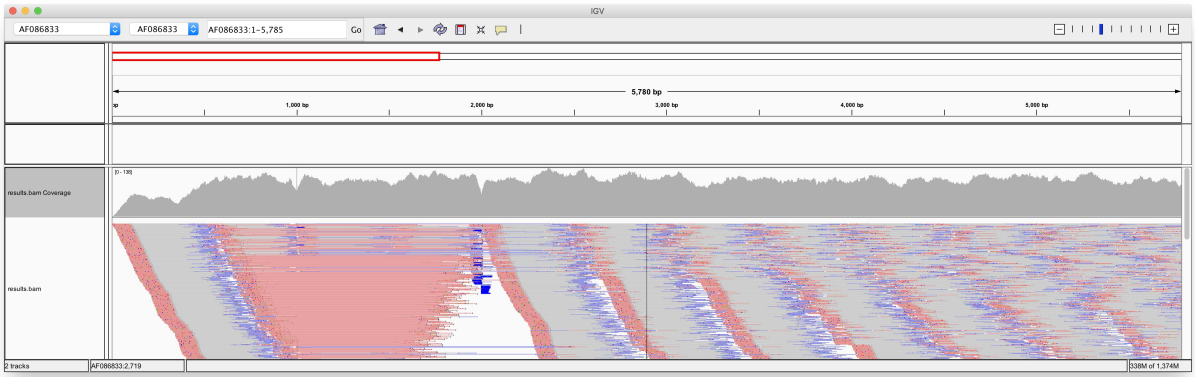



# What if we reverse complement parts of the genome?

In this example, we reverse complement some section of genome, but keep the rest the same:

```
cat refs/REFERENCE.fa | seqret --filter -sbegin 1 -send 1000 > part1
cat refs/REFERENCE.fa | seqret --filter -sbegin 1000 -send 2000 -sreverse1 > part2
cat refs/REFERENCE.fa | seqret --filter -sbegin 2000 > part3
cat part1 part2 part3 | union -filter  > GENOME.fa
```

The resulting image is best described as a "head-scratcher". This is when bioinformaticians look at one another and say: "What the heck is THAT?". Since we know what we put in we can explain what we see -- had this been an unknown reorganization, figuring out what happened would have been a lot more difficult.



# Genomic Variation

## What is a variant?

When it comes to defining genomic variations, the nomenclature and definitions show a surprising level imprecision. We all know what variation is *supposed* to mean of course. It should be a difference from what we expect to see. The problem is we almost never know what our expectations should be. At any given moment we only have imperfect information that changes in time.

As our understanding of genomes evolves concepts such as "reference genome" become harder to apply correctly. What should be considered a "reference genome"? After all the reference genome changes too - albeit usually in a more systematic way. Come to think of it just how different are different reference genomes from one another? Can we visualize that? Will last year's variants still look like variations this year?

These are questions that currently do not have satisfactory answers.

## How do we classify variants?

Genomic variations are typically categorized into different classes and are often denoted with a shortened acronym:

- SNP, a single nucleotide polymorphism - A single base change.
- INDEL, an insertion or a deletion - A single base added or removed.
- SNV, a single nucleotide variant. A SNPs or an INDEL - The change is a single base.
- MNP, a multi-nucleotide polymorphism - Several consecutive SNPs in a block.
- MNV, a multi-nucleotide variant - Multiples SNPs and INDELs as a block.
- Short variations. Variations (MNVs) that are typically less than **50bp** in length.
- Large-scale variations. Variations that are larger **50bp**.
- Structural variants. Variations on the kilobase scale that involve (one or more) chromosomes.

Radically different techniques need to be used to study problems in each class. Individual reads can capture small size changes, large-scale variants are typically detected from the relative positioning of reads about their read pairs as demonstrated in the Visualizing genomic variation chapter. For structural variations entirely new techniques (e.g. nano-channel arrays, Bionanogenomics ) may be needed to detect the large (> 1kb to megabase(s)) variations.

As a rule, SNPs are the most reliably detectable from data. In contrast very short MNVs that include insertions and deletions are the most difficult to resolve correctly. As you shall see in one of the next chapters, for example, a deletion followed by another variation can often be better aligned in alternate positions and that that can "fool" variant callers.

## What are SNPs?

SNPs (Single Nucleotide Polymorphisms) are a particular case of small scale variations. They are defined as:

> A single nucleotide polymorphism, also known as simple nucleotide polymorphism, (SNP, pronounced snip; plural snips) is a DNA sequence variation occurring commonly within a population (e.g. 1%) in which a single nucleotide - A, T, C or G of a shared sequence differs between members of a biological species (or paired chromosome of the same individual).

Note how a SNP is defined as only a polymorphism (a change of a base) and also implies a frequency of observing it in a population.

# Is the SNP term used consistently?

Not really. Previously I have mentioned how "gene" was the most misused word in biology. Well "SNP" is a close second. In our observations most bioinformaticians and most reference and training materials will often use the word SNP to refer to SNVs or even MNVs, sometimes to **all** short variants.

For a start SNPs (snip) is easy to pronounce. It just rolls off your tongue. Then the 1% limitation is, of course, arbitrary. As a matter of fact, the first cutoff for SNPs was at 5% a limit that was only later reduced to 1%.

Another problem with the definition of SNPs is that scientists are unable to randomly sample populations and with that, they introduce the so-called selection bias. It could easily happen that the frequency of a variation that initially appears to be over 1% then later turns out to be less than that value. In our opinion, the 1% cutoff is often used to imply that this variant was observed before.

We believe that even the scientific language is dynamic and we should accept the reality that most scientists use the word SNP to refer to polymorphisms where the frequency in a population is not known. In this book, we will do the same.

Also, we need to remember that what others call SNPs may often include not just polymorphisms but also single nucleotide insertions and deletions.

# Do SNPs cause disease?

SNPs are in a sense the holy grail of genomics. Their "popularity" perhaps is a manifestation of our hopes and wishful thinking. After all wouldn't it be great if every disease or human condition could be distilled down to a few changes and mutations in the genome? We would just need to find that one SNP, and with that we solve the problem, collect our reward and move onto the next problem.

And in fact sometimes (rarely though) things do work out this way. Some individual mutations do indeed cause disease(s). The OMIM database (Online Mendelian Inheritance in Man) is an online catalog of human genes and genetic disorders collect this data.

Visit the chapter called Online Mendelian Inheritance in Man for details on what is currently known about inheritable diseases and how we can interpret the data that describes the inheritance.

# What is a genotype?

A genotype is the genetic constitution of an individual organism.

In the context of genomic variations, a genotype is a defined a little more loosely. It typically refers identifying one, more than one, or all variations of the sequence under investigation.

Specifically, the word "genotyping" seems to be used very liberally in publications and training materials. Most of the time, when reading these it is not clear what the authors mean when they state that they have "genotype" data. What they probably and usually mean is that they have established with some confidence the base composition of their sequences - though it is often not clear how complete their information is.

# What is a haplotype?

A haplotype is a group of genes in an organism that is inherited together from a single parent.

In the context of genomic variations, a haplotype is a group of mutations that are inherited together. But just as with the "genotyping," the word "haplotyping" may be used in broader and narrower sense. Sometimes it just means that a genome can be grouped with other genomes of similar and known characteristics.

# Representing sequence variation in SAM files

The section discusses how variants are represented inside SAM files. It is somewhat technical in nature but don't let that discourage you. Being confident about your files means that you fully understand how that information is represented in them.

Thankfully in the majority of cases you won't need to operate on your files like this. But then, I guarantee you the time will come when you do. When knowing where to look this information up will save you substantial effort.

For example when variation calling goes awry you will need to investigate the alignment file and attempt to understand the information encoded in the file.

Let's prepare some data for this chapter.

```
# Get the reference genome.
mkdir -p refs
REF=refs/AF086833.fa
efetch -db=nuccore -format=fasta -id=AF086833 | seqret -filter -sid AF086833 > $REF

# Index reference with bwa
bwa index $REF
```

# How to mutate a sequence from the command line?

The `msbar` EMBOSS tool was designed to generate mutations in sequences. It can produce point mutations with the flag:

```
-point Types of point mutations to perform
       0 (None);
       1 (Any of the following)
       2 (Insertions)
       3 (Deletions)
       4 (Changes)
       5 (Duplications)
       6 (Moves)
```

Or block mutations with the flags

```
-block Types of block mutations to perform
       0 (None)
       1 (Any of the following)
       2 (Insertions)
       3 (Deletions)
       4 (Changes)
       5 (Duplications)
       6 (Moves)
```

Block mutations are a series of consecutive mutations that span from ranges specified with `-minimum` and `-maximum`.

Get a 50bp query from the reference file:

```
cat $REF | seqret -filter -send 50  > query.fa

# Introduce a point deletion
cat query.fa | msbar -filter -point 3 > mutated.fa

# Visualize the output:
global-align.sh query.fa mutated.fa
```

It will produce:

```
CGGACACACAAAAAGAAAGAAGAATTTTTAGGATCTTTTGTGTGCGAATA
 |||||||||||||||||||||||||||||| ||||||||||||||||||
CGGACACACAAAAAGAAAGAAGAATTTTTA-GATCTTTTGTGTGCGAATA
```

The same with a 2 base block mutation:

```
cat query.fa | msbar -filter -block 3 -minimum 2 -maximum 2 > mutated.fa
```

produces:

```
CGGACACACAAAAAGAAAGAAGAATTTTTAGGATCTTTTGTGTGCGAATA
 |||||||||||||||||||||||||||||||||||||||||||||  |||||
CGGACACACAAAAAGAAAGAAGAATTTTTAGGATCTTTTGTGT--GAATA
```

We can introduce more than one mutation with the `-count` parameter:

```
cat query.fa | msbar -filter -count 2 -block 3 -minimum 2 -maximum 2 > mutated.fa
```

that will produce:

```
CGGACACACAAAAAGAAAGAAGAATTTTTAGGATCTTTTGTGTGCGAATA
|||||||||||||  ||||||||||||||||||  |||||||||||||||
CGGACACACAAAA--AAAGAAGAATTTTTAGGA--TTTTGTGTGCGAATA
```

Since the tool selects mutations randomly, when choosing mutations like substitutions it is possible to mutate the sequence back to itself i.e. replacing an `A` with another `A`

One flaw of the `msbar` tool is that it does not tell us what changes it has made. Hence it is hard to use it in a context of validating a variant caller.

# How do I reconstruct alignment from CIGAR and MD strings?

The encoding of the SAM file (add cross-reference to SAM format information) makes reconstructing the actual alignment a little more laborious than one might imagine. Let us walk through a simple example of mutating a 50bp sequence from the ebola genome, then reconstructing that information from its alignment back to reference.

In a nutshell we will create the mutation:

```
CGGACACACAAAAAGAAAGAAGAATTTTTAGGATCTTTTGTGTGCGAATA
||||||||.||||||||||||||||||||||||||||||||||||||||
CGGACACATAAAAAGAAAGAAGAATTTTTAGGATCTTTTGTGTGCGAATA
```

and this above will be represented in the CIGAR `50M` and MD tag of `8C41` You can see advantages of the shortened representations the entire long strech above is a short word `8C41`

The `-point 4` flag creates a substitution in a random location of the sequence (in your case the mutation will be somewhere else and it is also possible that the mutation generates the existing base. In that case rerun it):

```
cat query.fa | msbar --filter --point 4 > mutated.fa
```

To see what the change actually is we can run our global aligner:

```
global-align.sh query.fa mutated.fa
```

it will print:

```
CGGACACACAAAAAGAAAGAAGAATTTTTAGGATCTTTTGTGTGCGAATA
||||||||.||||||||||||||||||||||||||||||||||||||||
CGGACACATAAAAAGAAAGAAGAATTTTTAGGATCTTTTGTGTGCGAATA
```

We can see how `msbar` above replaced a `C` with a `T`. So what will this look like in a SAM record?

Let us generate the alignment.

```
bwa mem $REF mutated.fa > align.sam
```

Then view the CIGAR, SEQUENCE and MD tags (columns 6,10 and 13):

```
samtools view align.sam | cut -f 6,10,13
```

This prints:

```
50M	CGGACACATAAAAAGAAAGAAGAATTTTTAGGATCTTTTGTGTGCGAATA	MD:Z:8C41
```

The SAM specification states that the MD field aims to achieve SNP/INDEL calling without looking at the reference. What this means is that from the CIGAR, SEQUENCE and MD columns we can reconstruct the original data and we don't need to know the reference.

The MD tag is value is `8C41`. It says that 8 bases match then the 9th is a mismatch and the reference contains a `C` at that location. The 9th base in the SEQUENCE is a `T` hence this we now know this is a `C` to `T` variation.

Usually you don't actually have to do this by hand as above. There are tools that do this for you. It is still important to know how to look this up as there will be times when tools let you down and it is up to you to figure out what is going on.

# What will deletions look like in a SAM format?

Let's look at a block deletion:

```
cat query.fa | msbar -filter -count 2 -block 3 -minimum 2 -maximum 2 > mutated.fa
```

That produces:

```
CGGACACACAAAAAGAAAGAAGAATTTTTAGGATCTTTTGTGTGCGAATA
||||||||||||||  ||||||||||||||||||  ||||||||||||||||
CGGACACACAAAA--AAAGAAGAATTTTTAGGA--TTTTGTGTGCGAATA
```

When aligned with `bwa` the resulting CIGAR and MD tags are `13M2D18M2D15M` and `13^AG18^TC15`:

```
13M2D18M2D15M    CGGACACACAAAAAAAGAAGAATTTTTAGGATTTTGTGTGCGAATA    MD:Z:13^AG18^TC15
```

Note how the CIGAR string tells us that we have `2D` but it is the MD tag that tells us what has been deleted: `^AG`

# What will insertions look like in a SAM format?

```
cat query.fa | msbar -filter -count 2 -block 2 -minimum 2 -maximum 2 > mutated.fa
```

produces:

```
CGGACACACAAAAAGAAAG--AAGAATTTTTAGGATCTTTTGTG--TGCGAATA
||||||||||||||||||  ||||||||||||||||||||||  ||||||||
CGGACACACAAAAAGAAAGATAAGAATTTTTAGGATCTTTTGTGCTTGCGAATA
```

the corresponding SAM CIGAR and MD fields are `19M2I23M2I8M` and `50`:

```
19M2I23M2I8M    CGGACACACAAAAAGAAAGATAAGAATTTTTAGGATCTTTTGTGCTTGCGAATA    MD:Z:50
```

# How long is an alignment?

This question can have different interpretations. What it typically means is the distance between the start and end coordinates on the reference.

For either case we can compute the alignment lengths from the CIGAR string alone following these rules:

Dealing with matches and mismatches cigar strings such as `50M` or in extended format `24=1X25=`

These will produce an alignment that is the sum of the integers in the formula: 50. We now see some of the logic behind labeling both matches and mismatches with `M`. They produce the same alignment lengths. Though this labeling is hopefully on the way out - as being a grossly misleading and frankly dumb way of describing alignments. When using the extended cigar string the `=` will be used for matches and the `X` sign for mismatches and that too would also produce a 24

- 1 + 25 = 50bp long alignments.

Dealing with deletions say we have `3S28M2D15M` the MD tag is `28^TC15`. Softcliping means that this region is not aligned. Then then the alignment length is 28 + 2 + 15 = 45 so the alignment will start at column POS and will go to POS + 45.

Dealing with insertions for example: `14M2I19M2I17M` the MD tag is `50`. The insertions are relative the query do not alter the alignment relative to the reference that will be 14 + 19 + 17 = 50.

# How to get the alignment lengths with a tool?

The `bedtools` suite can do this:

```
bedtools bamtobed -i align.bam
```

though be warned that by default this suite uses the BED format - a zero based data representation hence coordinates will go from zero.

# How to visualize a SAM file as pairwise alignment?

Matt LaFave's tool `sam2pairwise` does this:

- https://github.com/mlafave/sam2pairwise
- https://www.biostars.org/p/110498/

usage:

```
samtools view align.bam | sam2pairwise
```

will produce:

```
AF086833    0    AF086833    6    60    3S28M2D15M    *    0    0

CGGACACAAAAAGAAAGAAGAATTTTTAGGA--TTTTGTGTGCGAATA
   ||||||||||||||||||||||||||||  ||||||||||||||||
NNNACACAAAAAGAAAGAAGAATTTTTAGGATCTTTTGTGTGCGAATA
```

# Online Mendelian Inheritance of Man

Online Mendelian Inheritance in Man (OMIM) is a continuously updated catalog of human genes and genetic disorders and traits, with particular focus on the molecular relationship between genetic variation and phenotypic expression. It contains information about Mendelian disorders.

## How does OMIM work ?

Whereas the http://www.omim.org/ website offers useful search and visualization functions, a different type of data mining can be performed from the command line. Do note that OMIM website is primarily intended to assist physicians.

Here I want to make a point. Providing a service to clinicians is a high-risk task as the responsibilities associated are much higher than that of other services. OMIM tries to manage this risk by producing humongous amounts of information that may even just be distantly related to a search or term of interest. In essence it organizes information to some extent but it also tries very hard to transfer all responsibility to the end user. It makes it their job to figure out what all that means. Just about any searchable term will produce pages and pages of results. It is by design so - but remeber the actual useful information is probably a tiny fraction (if any) of all those results. When we investigate the data that underlies OMIM do we fully realize how far from completion this effort truly is.

## Can the OMIM data be obtained?

Whereas the main web site is the entry point to navigate for most people - the command line offers allows us to see and touch the data that connects it all together. Hence it may give different insights and deeper understanding and appreciation. The data itself has a fairly simple and flat structure and is amenable to be processed via the Unix command line.

Do note however that only the gene-to-phenotype data is available for download. The content of the pages that provide clinical advice is not included.

Registering at http://www.omim.org/ offers download access to their data in files named such as

- `mim2gene.txt` ,
- `mimTitles.txt` ,
- `genemap.txt` ,
- `morbitmap.txt` ,
- `genemap2.txt` ,

files that we can investigate from the Unix command line as well.

## What format is the OMIM data in?

The OMIM file formats (as typical in bioinformatics) are somewhat obscure and counter intuitive, going back many decades well before the internet era. Identifiers carry markers used to inform a viewers of the meaning of each term:

- Asterisk (*) Gene
- Plus (+) Gene and phenotype, combined
- Number Sign (#) Phenotype, molecular basis known
- Percent (%) Phenotype or locus, molecular basis unknown
- NULL () Other, mainly phenotypes with suspected mendelian basis
- Caret (^) Entry has been removed from the database or moved to another entry

As seen above for example the "percent" symbol was originally used as an marker to indicate a confirmed mendelian phenotype or phenotypic locus for which the underlying molecular basis is not known. In the data files the word `Percent` is written out explicitly.

# What terms and concepts does OMIM know about?

The `mimTitles.txt` file list the terms in the OMIM database. Think of it as table of contents. It connects a so called MIM number to a concept.

```
# Copyright (c) 1966-2016 Johns Hopkins University. Use of this file adheres to the terms spec
ified at https://omim.org/help/agreement.
# Generated: 2016-11-05
# Prefix        Mim Number      Preferred Title; symbol Alternative Title(s); symbol(s) Includ
ed Title(s); symbols
NULL    100050  AARSKOG SYNDROME, AUTOSOMAL DOMINANT
Percent 100070  AORTIC ANEURYSM, FAMILIAL ABDOMINAL, 1; AAA1    ANEURYSM, ABDOMINAL AORTIC; AAA
;; ABDOMINAL AORTIC ANEURYSM
Number Sign     100100  PRUNE BELLY SYNDROME; PBS       ABDOMINAL MUSCLES, ABSENCE OF, WITH UR
INARY TRACT ABNORMALITY AND CRYPTORCHIDISM;; EAGLE-BARRETT SYNDROME; EGBRS
NULL    100200  ABDUCENS PALSY
Number Sign     100300  ADAMS-OLIVER SYNDROME 1; AOS1   AOS;; ABSENCE DEFECT OF LIMBS, SCALP,
AND SKULL;; CONGENITAL SCALP DEFECTS WITH DISTAL LIMB REDUCTION ANOMALIES;; APLASIA CUTIS CONG
ENITA WITH TERMINAL TRANSVERSE LIMB DEFECTS    APLASIA CUTIS CONGENITA, CONGENITAL HEART DEFEC
T, AND FRONTONASAL CYSTS, INCLUDED
Caret   100500  MOVED TO 200150
Percent 100600  ACANTHOSIS NIGRICANS
```

The number of total terms in OMIM:

```
cat mimTitles.txt | grep -v '#' | wc -l
# 24970
```

# How many phenotypes of mendelian inheritance?

Phenotypes where the molecular origin is known:

```
cat mimTitles.txt | grep 'Number' | wc -l
# 4869
```

Phenotypes where the molecular origin is unknown:

```
cat mimTitles.txt | grep 'Percent' | wc -l
# 1615
```

What are the phenotypes of unknown molecular origin:

```
cat mimTitles.txt | grep 'Percent' | head
```

Producing the file:

```
Percent 100070  AORTIC ANEURYSM, FAMILIAL ABDOMINAL, 1; AAA1    ANEURYSM, ABDOMINAL AORTIC; AAA
;; ABDOMINAL AORTIC ANEURYSM
Percent 100600  ACANTHOSIS NIGRICANS
Percent 100820  ACHOO SYNDROME  AUTOSOMAL DOMINANT COMPELLING HELIOOPHTHALMIC OUTBURST SYNDROME
;; PHOTIC SNEEZE REFLEX;; SNEEZING FROM LIGHT EXPOSURE;; PEROUTKA SNEEZE
Percent 101850  PALMOPLANTAR KERATODERMA, PUNCTATE TYPE III; PPKP3      ACROKERATOELASTOIDOSIS;
 AKE;; COLLAGENOUS PLAQUES OF HANDS AND FEET
Percent 102100  ACROMEGALOID CHANGES, CUTIS VERTICIS GYRATA, AND CORNEAL LEUKOMA        ROSENT
HAL-KLOEPFER SYNDROME
Percent 102150  ACROMEGALOID FACIAL APPEARANCE SYNDROME AFA SYNDROME;; THICK LIPS AND ORAL MUC
OSA
Percent 102300  RESTLESS LEGS SYNDROME, SUSCEPTIBILITY TO, 1; RLS1      ACROMELALGIA, HEREDITA
RY;; EKBOM SYNDROME
Percent 102350  ACROMIAL DIMPLES        SUPRASPINOUS FOSSAE, CONGENITAL
Percent 102510  ACROPECTOROVERTEBRAL DYSPLASIA; ACRPV   F SYNDROME
Percent 102800  ADENOSINE TRIPHOSPHATASE DEFICIENCY, ANEMIA DUE TO
```

The file can be searched (case insensitive) for diseases:

```
cat mimTitles.txt | egrep -i sickle
```

to produce:

```
NULL    143020  HPA I RECOGNITION POLYMORPHISM, BETA-GLOBIN-RELATED; HPA1       RESTRICTION FR
AGMENT LENGTH POLYMORPHISM, SICKLE CELL ANEMIA-RELATED
Number Sign     603903  SICKLE CELL ANEMIA
```

The number `603903` is the so called MIM number, a unique identifier within the framework. To find the gene connected to sickle cell Anemia

```
cat genemap2.txt | grep 603903
```

produces a fairly lengthy (but still single line entry with):

```
chr11   5225465 5227070 11p15.4         141900  HBB     Hemoglobin beta HBB     3043    ENSG00
000244734 pseudogene
HBBP1 between HBG and HBD loci       Delta-beta thalassemia, 141749 (3), Autosomal dominant; E
rythremias, beta- (3);
Heinz body anemias, beta-, 140700 (3), Autosomal dominant; Hereditary persistence of fetal hem
oglobin, 141749 (3),
Autosomal dominant; {Malaria, resistance to}, 611162 (3); Methemoglobinemias, beta- (3); Sickl
e cell anemia, 603903
(3), Autosomal recessive; Thalassemia-beta, dominant inclusion-body, 603902 (3); Thalassemias,
 beta-, 613985 (3)
   Hbb-bt,Hbb-bs (MGI:5474850,MGI:5474852)
```

To see just a first few columns:

```
cat genemap2.txt | grep 603903 | cut -f 1-8
```

produces:

```
chr11   5225465 5227070 11p15.4         141900  HBB     Hemoglobin beta
```

# How many phenotypes are connected to genes?

The 13th column of the genemap2.txt files connects a gene to a phenotype. But not all those columns ar filled in the file.

```
cat genemap2.txt | cut -f 13 |  wc -l
# 16086

cat genemap2.txt | cut -f 13 | egrep '.+' | wc -l
# 4996
```

Out of `16086` genes in this file just `4996` have non-zero field for phenotypes.

# How many unique pairs of gene and phenotypes are there?

As it turns out this is unexpectedly difficult to extract. Columns 6 and 13 contain what we need, but the latter is a free text format from which each 6 digit MIM number needs to be parsed out.

```
cat genemap2.txt | grep 603903 | cut -f 13
```

The output of which is:

```
Delta-beta thalassemia, 141749 (3), Autosomal dominant; Erythremias, beta- (3);
Heinz body anemias, beta-, 140700 (3), Autosomal dominant; Hereditary persistence of
fetal hemoglobin, 141749 (3), Autosomal dominant; {Malaria, resistance to}, 611162 (3);
Methemoglobinemias, beta- (3); Sickle cell anemia, 603903 (3), Autosomal recessive;
Thalassemia-beta, dominant inclusion-body, 603902 (3); Thalassemias, beta-, 613985 (3)
```

What we need from this are the numbers `141749`, `140700` and preferable a file that lists them like so:

```
603903   141749
603903   140700
603903   141749
...
```

This is one of those typical moments, and you will yourself run into this a lot, when a bioinformatician needs to write some simple code to get them past an obstacle. There are many ways to solve this, for an example `awk` program would look like this:

```
# Split input by tabs.
# Format output by tabs.
BEGIN { FS = OFS = "\t" }

{
    # Gene MIM number is column 6.
    genemim = $6
    geneid = $8

    # The phenotypes are in column 13.
    split($13, elems, " ")

    # Go over each element and print the matches to numbers.
    for (i in elems){
        word = elems[i]

        # MIM numbers have exactly 6 digits.
        if (word ~ /^[0-9]{6}/ ){
            print genemim, geneid, word
        }
    }
}
```

Running this code on the entire data and tabulating at the end. How many disease/phenotype entries are there:

```
cat genemap2.txt | awk -f prog.awk | wc -l
# 6964
```

So the `4996` annotated genes produce a total of `6964` phenotypes.

# What genes have the most annotated phenotypes?

These are the genes where mutations appear to cause the most phenotypes:

```
cat genemap2.txt | awk -f prog.awk | cut -f 1,2 | sort | uniq -c | sort -rn | head
```

produces:

```
 15 120140  Collagen II, alpha-1 polypeptide
 14 134934  Fibroblast growth factor receptor-3
 12 176943  Fibroblast growth factor receptor-2 (bacteria-expressed kinase)
 12 150330  Lamin A/C
 11 601728  Phosphatase and tensin homolog (mutated in multiple advanced cancers 1)
 11 191170  Tumor protein p53
 11 171834  Phosphatidylinositol 3-kinase, catalytic, alpha polypeptide
 11 109270  Solute carrier family 4, anion exchanger, member 1 (erythrocyte membrane protein b
and 3, Diego blood group)
 10 605427  Transient receptor potential cation channel, subfamily V, member 4 (vanilloid rece
ptor-related osmotically activated channel)
 10 300017  Filamin A, alpha (actin-binding protein-280)
```

So there are `15` entries in this file for gene with MIM number `120140` that corresponds to gene *Collagen II, alpha-1 polypeptide*. What are these phenotypes?

```
cat genemap2.txt | awk -F '\t' ' $6 ~ /120140/ { print $13 } '
```

That produces

```
Achondrogenesis, type II or hypochondrogenesis, 200610 (3), Autosomal dominant; Avascular necr
osis of the femoral head,
608805 (3), Autosomal dominant; Czech dysplasia, 609162 (3), Autosomal dominant; Epiphyseal dy
splasia, multiple, with
myopia and deafness, 132450 (3), Autosomal dominant; Kniest dysplasia, 156550 (3), Autosomal d
ominant;
Legg-Calve-Perthes disease, 150600 (3), Autosomal dominant; Osteoarthritis with mild chondrody
splasia, 604864 (3),
Autosomal dominant; Otospondylomegaepiphyseal dysplasia, 215150 (3), Autosomal recessive; Plat
yspondylic skeletal
dysplasia, Torrance type, 151210 (3), Autosomal dominant; SED congenita, 183900 (3), Autosomal
 dominant; SMED Strudwick
type, 184250 (3), Autosomal dominant; Spondyloepiphyseal dysplasia, Stanescu type, 616583 (3),
 Autosomal dominant;
Spondyloperipheral dysplasia, 271700 (3), Autosomal dominant; Stickler sydrome, type I, nonsyn
dromic ocular, 609508 (3),
Autosomal dominant; Stickler syndrome, type I, 108300 (3), Autosomal dominant; Vitreoretinopat
hy with phalangeal
epiphyseal dysplasia (3)
```

# Alignment Pileups

Before we delve deeper into the variant calling it is important to understand that all variant detection operates on alignment files. While the so called "variant callers" can refine alignments they can only improve on the existing alignments. Hence producing the right alignments to start with is essential.

Let's prepare some data for this chapter.

```
# Get the reference genome.
mkdir -p refs
REF=refs/1976.fa
efetch -db=nuccore -format=fasta -id=AF086833 | seqret -filter -sid AF086833 > $REF

# Index reference with bwa
bwa index $REF
```

# What are pileups?

A pileup is a data representation that shows the base composition at a given coordinate. A SNP caller operates on this type of data, a consensus over a coordinate or a region in the genome.

```
# Get some data and align it.
R1=SRR1972739_1.fastq
R2=SRR1972739_2.fastq

fastq-dump -X 10000 --split-files SRR1972739
bwa mem $REF $R1 $R2 | samtools sort > align.bam
samtools index align.bam
```

First you need to index the reference genome for samtools.

```
samtools faidx $REF
```

"Another indexing?" you may ask. Yes. It is an annoying and error prone process. Every tools asks us to prepare the reference for their use.

```
# Pileup with a reference file.
samtools mpileup -f $REF align.bam | head -5
```

This produces the output by the following columns: coordinate, genomic index, base in the reference, coverage, match type and, base quality (the columns are not actually printed):

```
CHROM       POS    REF   DEPTH   BASES  QUALITY

AF086833     46     G      1      ^].     C
AF086833     47     A      1      .       @
AF086833     48     A      1      .       C
AF086833     49     T      1      .       F
```

The based symbols such as `^].` can be decoded in the following way:

- `.` (dot) means a base that matched the reference on the forward strand
- `,` (comma) means a base that matched the reference on the reverse strand
- `AGTCN` denotes a base that did not match the reference on the forward strand
- `agtcn` denotes a base that did not match the reference on the reverse strand
- A sequence matching the regular expression `\+[0-9]+[ACGTNacgtn]+` denotes an insertion of one or more bases starting from the next position
- A sequence matching the regular expression `-[0-9]+[ACGTNacgtn]+` denotes a deletion of one or more bases starting from the next position
- `^` (caret) marks the start of a read segment and the ASCII of the character following `^' minus 33 gives the mapping quality
- `$` (dollar) marks the end of a read segment
- `*` (asterisk) is a placeholder for a deleted base in a multiple basepair deletion that was mentioned in a previous line by the `-[0-9]+[ACGTNacgtn]+` notation

So how do you interpret this line:

```
AF086833    46    G    1    ^].   C
```

It goes like this: "On sequence id `AF086833` the position `46` contains the base `G`. In the alignment file this coordinate and base is covered by `1` read that aligns such that this base is at the beginning of the read `^` and has a mapping quality of `]` on the forward strand `.` This base was sequenced with a FASTQ quality of `C`." That was a fun read, no? Not really. The mapping quality for `]` can be found as `python -c 'print ord("]")-33 = 60` and corresponds to 1 in a million (1/10^6)

Pileups can take location coordinates:

```
samtools mpileup -f $REF -r AF086833:470-470 align.bam | head
```

A pileup over a certain region:

```
AF086833    470    A    73    .,........$,...$..$...,.....,,,..,.,.,,,,,,,,,,.$.,,.,,,,,,,,..
,,....,,,,,.,^].    BIiiD<?ADBHEi/k0FBDJIHmCIJJJIGJHJHJJHCJJJHHFEHFFHEEEEDDCDHHCCIAHHDDDDD@DB
```

Pileups can take SAM flags in the `--rf` (required flag) or `--ff` (filtering flag)

```
# Pileup from reads on reverse strand.
samtools mpileup --rf 16 -f $REF align.bam -r AF086833:470-470 | head

AF086833    470    A    40    ,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,    IHFJJJJJJJJHCJJJHHFF
FFFEEEEDDCDCCDDDDDDD    40    ,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,    IHFJJJJJJJJHCJJJHHFF
FFFEEEEDDCDCCDDDDDDD
```

More information on pileups:

- Pileup format specification on Wikipedia.
- 5 Things to Know About SAMtools Mpileup on the MassGenomics blog.

# How to determine variants from pileups?

Recall the Densiovan alignment? The genome sequence of a Denisovan individual that lived 45,000 years ago was generated from a small fragment of a finger bone discovered in the Denisova Cave in southern Siberia in 2008.

- Website: http://www.eva.mpg.de/denisova
- Data: http://cdna.eva.mpg.de/denisova/

Let's access the Denisovan data at:

```
BAM=http://cdna.eva.mpg.de/denisova/alignments/T_hg19_1000g.bam
```

It is very important to understand what genome was this data aligned against. Note how very little information is present in structured form - we assume that the genome build was `hg19` from the word being present in the file name. Clearly that is not how this information should be stored, it should be encoded inside the file rather than outside of it (filename):

While as mentioned before the MD tag allows calling variations withouth a reference if we had that a lot simpler to visualize the differences if these exist.

```
# UCSC data download has data split by chromosomes for 7
CHR7=http://hgdownload.soe.ucsc.edu/goldenPath/hg19/chromosomes/chr7.fa.gz

# We need to unzip but also rename the chromosome from chr7 to just 7
curl -O $CHR7
cat chr7.fa.gz | gunzip | seqret -filter -sid 7 > chr7.fa
samtools faidx chr7.fa
```

View a location of this alignment:

```
samtools mpileup -f chr7.fa $BAM -r 7:17,375,390-17,375,394
```

This is what we get:

```
7  17375390   T   34   ,.,....,,,...c,...,.,,.,.,,,.....    ]G]]]]]Z[V]]S]]V]]]]]]]]]]]]]B]B
7  17375391   G   34   ,.,....,,,...,,...,.,,.,.,,,.....    ]@<]]]]XZYU]]W]]N]]]]]]]]]]W]]]?]<
7  17375392   T   34   cCcCCCCcccCCCccCCCcCccCcCcccCCCCCC    ]?B]Z]]]ZZQ]]U]]Z]]]]]]]]]]\]]]B]<
7  17375393   A   35   ,.,....,,,...,,...,.,,.,.,,,.....,    ]><]]]]]Z]S]][]][]]]]]]]]]]]]]]B]>
1
7  17375394   A   35   ,.,....,,,...,,...,.,,.,.,,,.....,    ]<9]Z]]]Z]V]]\]]]]]]]]]]]]]]]]A]B
=
```

# The Variant Call Format (VCF)

## What is the VCF format?

The variant call format (VCF) is a data representation format used to describe variations in the genome. A VCF file may contain information on any number of samples, thousands even and can be thought of as a single database that summarizes the final results of multiple experiments in a single file.

As a format VCF is composed of two sections: a **header** section and a **record** section.

While it is a plain text format, its contents can go from being easy to read and understand to seemingly impossible to visually decipher.

The challenge of reading out variants by eye is (probably) is not the format's "fault" - the variations in the genome can have relatively complex representations.

## How important is it to understand the VCF format?

In many cases, it is possible to perform sophisticated analyses without ever directly looking inside a VCF file. Many tools take VCF files as inputs and produce other reports out of them. At other times a cursory understanding of what a VCF file contains and how the information is laid inside of them out is sufficient to complete an entire analysis.

But just as with the SAM format - understanding the VCF format is essential for anyone that needs to venture off the beaten path. There will be times when crucial information is buried deep inside a VCF file with hundreds of samples. You may have to use toolsets like `bcftools` to filter, combine and subselect VCF files in every which way imaginable. At those times understanding the small details of the VCF format become indispensable.

## What is a VCF header?

VCF header sections are located at the beginning of the VCF files and consists of lines that start with the `##` symbols like so:

```
##fileformat=VCFv4.1
##contig=<ID=AF086833,length=18959>
##INFO=<ID=AF,Number=A,Type=Float,Description="Allele Frequency">
##FORMAT=<ID=GT,Number=1,Type=String,Description="Genotype">
```

As variant calling is a more complex operation than alignment, additional information needs to be recorded about the process.

The VCF headers contain the specifications on various terms that used throughout the file. For example, from the above header, we learn that `AF` in the VCF file will be used to describe a variant and it means "Allele Frequency" and seeing `AF=0.5` would mean that the allele frequency for a variant was measured to be `0.5` .

The starting word `INFO` in the definition line above means that the ID will describe a variant across all samples. This means that the same value applies to (or has been derived from) all samples together and those characterize the variant.

If the word `FORMAT` is used instead it would mean that, for this ID you will have a different value for each sample.

You might ask yourself: but what does "Allele Frequency" itself mean? Where is the actual definition of how that term was computed? Some terms are well-defined concepts from other scientific fields like population genetics and will be relatively easy to find a meaning for. For example, if you did not know beforehand a Google search could tell you that "allele frequency" is the frequency of a variant in a population. It can be defined as the fraction of all chromosomes that carry the allele.

But there will be terms for which it will be much harder to find a proper explanation. For example:

```
## INFO=<ID=VDB,Number=1,Type=Float,Description="Variant Distance Bias for
## filtering splice-site artefacts in RNA-seq data (bigger is better)",Version="3">
```

So what is `VDB` then? You'd have to do a lot of legwork to figure it out, perhaps read the manual or publication of the variant caller that produced this value. And occasionally, for some terms, you may not ever find (or understand) the explanation on your own.

Even the creator of this `VDB` ID feels the need to mention that for this value "bigger is better." That is not all that encouraging from the perspective of a scientist. You might ask yourself the question, do I even have to know about `VDB` in the first place? To which the answer usually is, no, you don't.

Some (many or even most) IDs might be immaterial to the study that you are conducting. Variant callers are very prolific and liberally sprinkle values of all kinds of quantities and measures that the creators of these tools felt that should always be mentioned. Occasionally these are relevant but (or even most) of the time you neither need to nor would want to know them.

# What are VCF records?

The record section of a VCF file consists of at least eight tab delimited columns where the first eight columns describe a variant and the rest of the columns describe the properties of each sample. The 9th column is the `FORMAT` and each column past the 9th will represent a sample.

As mentioned before, but it is worth reiterating, a VCF file may contain any number of sample columns, thousands even and can be thought of as a single database that represents all variations in all samples.

The first nine columns of a VCF file are:

1. `CHROM` : The chromosome (contig) on which the variant occurs
2. `POS` : The genomic coordinates on which the variant occurs. For deletions, the position given here

are on of the bases preceding the event.

3. `ID` : An identifier for the variant (if it exists). Typically a dbSNP database if that is known.
4. `REF` : The reference allele on the forward strand.
5. `ALT` : The alternate allele(s) on the forward strand. More than one may be present.
6. `QUAL` : A probability that the `REF/ALT` variant exists at this site. It is in Phred scale, just as the FASTQ quality and the MAPQ field in the SAM file are.
7. `FILTER` : The name of filters that the variant fails to pass, or the value `PASS` if the variant passed all filters. If the `FILTER` value is `.` , then no filtering has been applied to the record.
8. `INFO` : Contains the site-specific annotations represented as `ID=VALUE` format.
9. `FORMAT` : Sample-level annotations as colon separated TAGS.

For example:

```
#CHROM    POS   ID  REF    ALT   QUAL   FILTER  INFO    FORMAT     align.bam
AF086833   60   .    A      T     54       .    DP=43    GT:PL    0/1:51,0,48
```

# How to interpret the FORMAT field?

This field specifies the meaning of the numbers in the each column of the sample. The fields are colon `:` separated and map each field of the `FORMAT` to each value in the sample column. Suppose that the following were columns 9,10 and 11 of a VCF file:

```
FORMAT         sample1         sample2
GT:PL         0/1:51,0,48    1/1:34,32,0
```

We interpret it in the following way.

- The variant observed for `sample1` has the values `GT=0/1` and `PL=51,0,48`
- This same variant when observed in `sample2` has the values `GT=1/1` and `PL=34,32,0`

What if you wanted to know what do `GT` and `PL` mean? You'd have to go to the header for their definition, and there you would find

```
##FORMAT=<ID=GT,Number=1,Type=String,Description="Genotype">
##FORMAT=<ID=PL,Number=G,Type=Integer,Description="List of Phred-scaled genotype likelihoods">
```

# What is represented in the REF/ALT columns.

The `REF` column represents the reference allele at position `POS`

The `ALT` column represent **all** variants observed at that site. When there is only a single simple variant, say a SNP at position 60 where an `A` was replaced by a `T` the VCF will look eminently readable. The variant record would state:

```
#CHROM    POS   ID  REF    ALT   QUAL   FILTER  INFO    FORMAT     align.bam
AF086833   60   .    A      T     54       .    DP=43    GT:PL    0/1:51,0,48
```

It also tells us that the genotype is `GT=0/1` and that Phred-scaled genotype likelihoods are `PL=51,0,48` (we will explain both concepts in more detail a bit later).

Now when there are two possible variants in the same location, it starts to be a little more complicated. This site now has two alternates at the same site:

```
#CHROM    POS   ID  REF    ALT   QUAL  FILTER  INFO     FORMAT         align.bam
AF086833   60   .    A     T,C   43.2     .    DP=95    GT:PL   1/2:102,124,42,108,0,48
```

There are two alternates, and we see from the `GT` field that our sample was heterozygous for the alternates. Neither of the alleles matched the reference.

Alas, the complexity does not stop there. This is where the VCF format becomes a lot harder to interpret. Previously we only mutated base `60`. But what if beyond just the mutation, some members of this population had a three base deletion at coordinates `58`, `59` and, `60`. Now the our VCF record will show:

```
#CHROM    POS   ID  REF    ALT   QUAL  FILTER   INFO         FORMAT        align.bam
AF086833   55   .   TGAGGA  TGA   182     .     INDEL;IDV=42   GT:PL    0/1:215,0,255
AF086833   60   .   A       C,T   39.8    .     DP=126;        GT:PL    1/2:99,87,47,86,0,49
```

What is important to note here that even though that the deletion started at coordinate `58` is shown at coordinate `55` in the file, but since the both the `ALT` and `REF` start with identical bases the `TGA` difference between them begin only after three bases. So when `TGAGGA` is replaced by `TGA` the actual change is the three base deletion starting from `58` and will overlap with coordinate `60`. But it is clear that you can't easily see that from the VCF file.

In fact, you may even ask yourself why is the variant reported as `TGAGGA --> TGA`? Why not `GAGGA --> GAGGA` or even `GGA ->`? Clearly we could report these difference in many different ways. You'll be surprised to know that this is still a not fully resolved problem in variant reporting.

There is a method called "variant normalization" that attempts to ensure that variants are reported identically. To explore that and understand how variants are determined and represented see the chapter [NEXT WEEK]

# How are genotypes described?

Whereas the `REF` and `ALT` columns show the change, we need to know how many of the copies of the DNA carry the variant or variants. The variant is encoded in the field called `GT` that indicates the genotype of this sample at this site.

It is constructed out of slash-separated numbers where:

- `0` indicates the `REF` field,
- `1` indicates the first entry in the the `ALT` field,
- `2` indicates the second entry in the the `ALT` field and so on.

For example for diploid organism the GT field indicates the two alleles carried by the sample:

- `0/0` - the sample is homozygous reference
- `0/1` - the sample is heterozygous, carrying one of each the REF and ALT alleles
- `1/2` - would indicate a heterzygous carrying one copy of each of the ALT alleles.
- `1/1` - the sample is homozygous for the first alternate
- `2/2` - the sample is heterozygous for the second alternate

and so on. For samples of higher ploidy (multiple copies of chromosomes) there would be as many `/` characters as there are copies of each chromosome. For example, a tetraploid genotype could be `0/1/0/2`.

# What are genotype likelihoods?

The genotype likelihoods are normalized probabilities that describe the probability that each of the possible genotypes occurs. The value is normalized to 0. So the most likely genotype is 0, and the rest are computed relative to that. Since it is a statistical measure (a p-value), it is the chance of not observing the genotype hence the "best" value is 0.

So let's take the:

```
GT:PL    0/1:51,0,48
```

`PL` field will contain three numbers, corresponding to the three possible genotypes ( `0/0` , `0/1` , and `1/1` ). What is says the following:

- The support is highest for genotype `0/1` and that is shown as zero.
- The chance that this genotype is `0/0` instead of `0/1` is Phred score 51 meaning `0.000001`
- The chance that this genotype is `1/1` instead of `0/1` is Phred score 48 meaning `0.00001`

We got the `0.000001` from the formula `10^(-51/10)` it is the same Phred score that we grew to love (not really) where you drop a digit from it and that's how many numbers you go to the left 40 -> 0.0001

# What is a BCF file?

The BCF format is a binary representation of the VCF format. It has the a similar relationship as the SAM/BAM but with the exception that the BCF files are not sorted and indexed and cannot be queried the same way as typical BAM files do.

# How do we generate VCF files?

Typically a VCF file is a result of running a "variant caller" (or "SNP caller" as some people call it) on one or more BAM alignment files. The result of running the variant caller will be a VCF file that contains a column for each sample. A single BAM file may also contain multiple samples tagged via readgroups in which case the variant caller may report these as different columns as well.

The `dwgsim` simulator that we already used also produces a VCF file with the mutations it has simulated. These can be viewed in IGV just by dragging the file into the panel.

```
REF=refs/AF086833.fa
efetch -db=nuccore -format=fasta -id=AF086833 | seqret -filter -sid AF086833 > $REF
dwgsim $REF data
```

will produce a file called `data.mutations.vcf`. Take a moment to investigate this file.

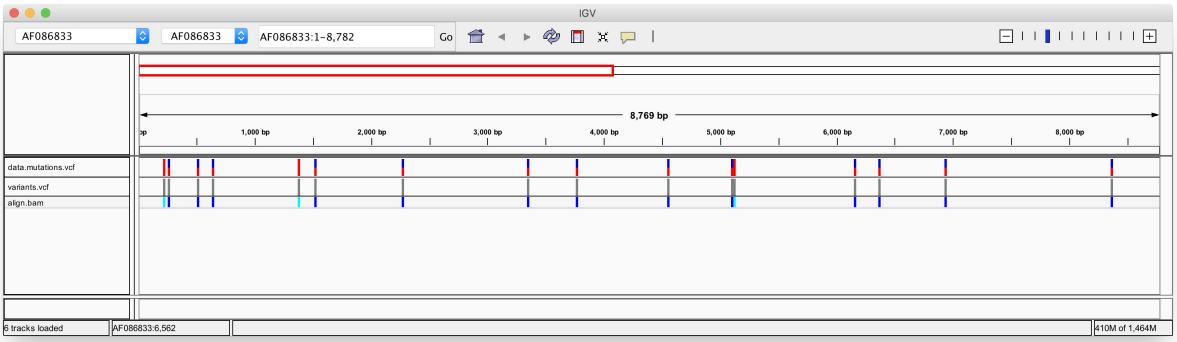
```
##fileformat=VCFv4.1
##contig=<ID=AF086833,length=18959>
##INFO=<ID=AF,Number=A,Type=Float,Description="Allele Frequency">
##INFO=<ID=pl,Number=1,Type=Integer,Description="Phasing: 1 - HET contig 1, #2 - HET contig #2
, 3 - HOM both contigs">
##INFO=<ID=mt,Number=1,Type=String,Description="Variant Type: SUBSTITUTE/INSERT/DELETE">
#CHROM      POS    ID    REF ALT   QUAL    FILTER        INFO
AF086833    1093    .    T    C    100    PASS    AF=0.5;pl=2;mt=SUBSTITUTE
AF086833    1195    .    A    T    100    PASS    AF=0.5;pl=1;mt=SUBSTITUTE
AF086833    2566    .    G    T    100    PASS    AF=0.5;pl=2;mt=SUBSTITUTE
AF086833    3345    .    A    AC   100    PASS    AF=0.5;pl=2;mt=INSERT
```

Let's now generate read data from the mutated genome and compare our variants to those that are represented in this file above.

Originally `samtools` used to include a variant caller that later was moved to a different package called `bcftools`. Beyond variant calling, this suite has a very broad number of use cases that we will explore in other chapters.

```
# This is the data naming generated by dwgsim.
R1=data.bwa.read1.fastq
R2=data.bwa.read2.fastq

#
# Generate the alignment from the data simulated above.
#
bwa mem $REF $R1 $R2 | samtools sort > align.bam
samtools index align.bam
samtools faidx $REF

# Compute the genotypes from the alignment file with pileup.
samtools mpileup -uvf $REF align.bam > genotypes.vcf

# Call the variants from the genotypes.
bcftools call -vm -Ov genotypes.vcf > variants.vcf
```

The resulting file called `variants.vcf` may be visualized in IGV, here we are comparing it against the expected variations generated by `dwgsim`:




# What are additional resources?

- VCF Poster: http://vcftools.sourceforge.net/VCF-poster.pdf
- VCF short summary: http://www.htslib.org/doc/vcf.html
- VCF Specification: http://samtools.github.io/hts-specs/
- A detailed description of the VCF: What is a VCF file on the GATK forums.

# Filtering information in a VCF files

Tools that can be used for filtering and manipulating VCF files include

- Bcftools
- SnpSift

Most bcftools commands accept VCF, bgzipped VCF and BCF. However some commands (eg:bcftools query) require a tabix indexed and hence bgzip compressed file.

SnpSift filtering tool is especially useful if you have a 'SnpEff' annotated VCF file.

The VCF file and its index used in the examples here can be downloaded as

```
curl -O http://data.biostarhandbook.com/variant/subset_hg19.vcf.gz
curl -O http://data.biostarhandbook.com/variant/subset_hg19.vcf.gz.tbi
```

This file contains variants in chromosome 19:400kb-500kb.

# How do I extract information from VCF in custom format?

Bcftools `query` command wiht the `-f` flag can be used to extract fields from VCF or BCF files. To extract just the CHROM, POS, REF and ALT fields:

```
bcftools query -f '%CHROM %POS %REF %ALT\n' subset_hg19.vcf.gz | head -3
```

to produce:

```
19 400410 CA C
19 400666 G C
19 400742 C T
```

# How do I list all samples stored in a VCF file?

```
bcftools query -l subset_hg19.vcf.gz
```

produces:

```
HG00115
HG00116
HG00117
HG00118
HG00119
HG00120
```

# How do I extract all variants from a particular region ?

To specify the region, use `-r` option with region specified in the format `chr|chr:pos|chr:from-to|chr:from-[,…]`

```
bcftools query -r '19:400300-400800' -f '%CHROM\t%POS\t%REF\t%ALT\n' subset_hg19.vcf.gz | head
 -3
```

This will produce:

```
19    400410    CA    C
19    400666    G    C
19    400742    C    T
```

If you want to output the results as a VCF file, use `filter` or `view` command instead of `query` . Multiple regions can be listed together in a bed file and can be loaded with `-R` flag to extract variants in those intervals.

# How do I extract variants excluding a specific region?

To exclude a certain region, it can be prefixed with `^` . The `-r` option cannot be used in this case instead use `-t` .

```
bcftools query -t ^'19:400300-400800' -f '%CHROM\t%POS\t%REF\t%ALT\n' subset_hg19.vcf.gz | head
 -3
```

To produce:

```
19    400819    C    G
19    400908    G    T
19    400926    C    T
```

To exclude multiple regions, compile them into a bed file and use with `-T` and `^` .

```
bcftools query -T ^exclude.bed -f '%CHROM\t%POS\t%REF\t%ALT\n' subset_hg19.vcf.gz | head -3

19    400742    C    T
19    400819    C    G
19    401013    G    A
```

where exclude.bed looks like this

```
#cat exclude.bed
19    400300    400700
19    400900    401000
```

# How do I get variants present in all samples?

We can exclude sites with one or more missing genotype and one or more homozygous reference call to extract variant sites that are present in all samples. Bcftools `view` command with `-g,--genotype` option helps to achieve this.

```
bcftools view -e 'GT= "." | GT="0|0"' subset_hg19.vcf.gz |bcftools query -f '%POS[\t%GT\t]\n'
| head -3

402556    0|1        0|1        1|1        1|0        0|1        1|1
402707    0|1        0|1        1|1        1|0        0|1        1|1
402723    0|1        0|1        1|1        1|0        0|1        1|1
```

The output of `bcftools view` command is piped into `query` command to print it in a user friendly format.

# How do I select INDELs only?

`query` command with `-v, --types` option can be used to select a variant type of interest. Here a site is selected if any of the ALT alleles is of the type requested. In this case,types are determined by comparing the REF and ALT alleles in the VCF record not INFO tags like INFO/INDEL or INFO/SNP.

Another way to extract a specific variant is to use the information given in INFO tag. For this we can use `-i, --include` option along with an expression.

In our example we do not have any INDEL sites.

```
bcftools view -v indels subset_hg19.vcf.gz | bcftools query -f '%POS\t%TYPE\n'  |wc -l
141

bcftools view -i 'TYPE="indel"' subset_hg19.vcf.gz | bcftools query -f '%POS\t%TYPE\n'  |wc -l
140
```

The difference in the two commands is because of the cariant at position 485135. It is marked in the INFO/TYPE field as

```
485135    SNP,INDEL
```

Hence when searched with an expression for exact match in the INFO field, this variant won't be included.

Multiple variant types can be selected with `-v` by listing them separated by comma. To exclude a variant type use `-V,--exclude-types`.

# How do I extract per-sample information from a multi-sample VCF file?

`FORMAT` which is the 9th column in VCF file specifies how to interepret the colon separated values pertaining to each sample given in 10th column onwards. These `FORMAT` tags can be extracted using the square brackets [] operator, which loops over all samples.

Print genotype (GT) of all samples at each position.

```
bcftools query -f '%CHROM\t%POS[\t%GT\t]\n' subset_hg19.vcf.gz | head -3
```

Produces:

```
19    400410    0|0        0|0        0|0        0|0        0|0        0|0
19    400666    1|0        0|1        0|1        0|0        0|0        1|1
19    400742    0|0        0|0        0|0        0|0        0|0        0|0
```

If you want to print header too, use `-H`

```
bcftools query -H -f '%CHROM\t%POS[\t%GT\t]\n' subset_hg19.vcf.gz | head -3
```

# How do I select specific samples?

`view` command with `-s, --samples` option can be used to subset a VCF file based on samples.

To select samples `HG00115` and `HG00118` from the sample vcf file.

```
bcftools view -s HG00115,HG00118 subset_hg19.vcf.gz | bcftools query -H -f '%POS[\t%GT]\n' | h
ead -4
```

produces:

```
# [1]POS    [2]HG00115:GT    [3]HG00118:GT
400410    0|0    0|0
400666    1|0    0|0
400742    0|0    0|0
```

# How do I exclude specific samples?

To exclude samples `HG00115` and `HG00118` from VCF file.

```
bcftools view -s ^HG00115,HG00118 subset_hg19.vcf.gz  | bcftools query -H -f '%POS[\t%GT]\n' |
head -4
```

produces:

```
# [1]POS    [2]HG00116:GT    [3]HG00117:GT    [4]HG00119:GT    [5]HG00120:GT
400410    0|0    0|0    0|0    0|0
400666    0|1    0|1    0|0    1|1
400742    0|0    0|0    0|0    0|0
```

# How do I get variants for which allele count is above a certain value?

`view` command with `-c, --min-ac` helps to set the minimun allele count of sites to be printed.

Command below remove sites where allele count (AC tag) < 5.

```
bcftools view -c 5 subset_hg19.vcf.gz | grep -v "#" | wc -l
# 185
```

# How do I select sites where all samples have homozygous genotype ?

`view` command with `-g,--genotype` helps to include or exclude one or more homozygous (hom), heterozygous (het) or missing (miss) genotypes.

```
 bcftools view -g ^het subset_hg19.vcf.gz | bcftools view -g ^miss | bcftools query -f '%POS[\
t%GT\t]\n'| head
 # 3891
```

In the command above `-g ^het` and `-g ^miss` exclude sites with one or more hetrozygous variant or a missing genotype.

# How do I select sites where 'n' samples have heterozygous variants?

SnpSift `filter` command below selects sites where 3 or more samples have heterozygous genotypes.

In the command below snpsift is an alias to "java -jar ~/bin/SnpSift.jar"

```
alias snpsift="java -jar ~/bin/SnpSift.jar"
snpsift filter  "countHet() >=3" subset_hg19.vcf.gz |snpsift extractFields - "POS" "GEN[*].GT"
|  head -4
```

will produce

```
#POS    GEN[*].GT
400666    1|0    0|1    0|1    0|0    0|0    1|1
400941    0|1    1|0    0|1    1|0    0|0    0|0
401235    0|0    0|0    0|1    0|0    0|1    1|0
```

# How do I select variant sites based on quality and read depth?

If we want to extract sites with QUAL >50 and read depth >5000 , then these conditions can be included in an expresion like this.

`QUAL>50 && DP>5000`

This can be used with `-i,--include` to extract sites satisfying the expression. The command would then be

```
bcftools query -i 'QUAL>50 && DP>5000' -f '%POS\t%QUAL\t%DP\n' subset_hg19.vcf.gz |  head -3

400410    100    7773
400666    100    8445
400742    100    15699
```

# How do I merge multiple VCF/BCF files?

If we had multiple VCF files, then we can merge them into a single multi-sample VCF with bcftools `merge` command.

```
bcftools merge -l samplelist > multi-sample.vcf
```

where samplelist is

```
#cat samplelist
sample1.vcf.gz
sample2.vcf.gz
sample3.vcf.gz
....
```

# How do I find variant sites present in all vcf files?

If we have 3 samples and a vcf file for each of them, then to find sites that are common to all 3

```
bcftools isec -p outdir -n=3 sample1.vcf.gz sample2.vcf.gz sample3.vcf.gz
```

`n=3` in the command specifies to output sites present in 3 files. The output files will be in a directory named 'outdir'.

# How do I extract variant sites present in at lest 2 files?

```
bcftools isec -p outdir -n+2 sample1.vcf.gz sample2.vcf.gz sample3.vcf.gz
```

`n+2` in the command specifies to output sites present in 2 or more files. The output files will be in a directory named 'outdir'.

# How do I find variants unique to one file but absent in all other files?

Bcftools `isec` command with `-C,--complent` option output positions present only in the first file but missing in others.

```
bcftools isec -p outdir -C sample1.vcf.gz sample2.vcf.gz sample3.vcf.gz
```

The above command will produce sites unique to sample1.

# Variant calling

Variant calling is the process of identifying and cataloging the differences between the measurements (sequencing reads) and a reference genome.

## How do I call variants?

The process is fairly straightforward as long as you view the process as a series of interconnected yet still separate processes, rather than a single one step analysis.

At the simplest level variants are usually determined ("called") from alignments (BAM) files. A typical process involves the following:

1. Align reads to reference.
2. Correct and refine alignments (this may be optional).
3. Determine variants from the alignments.
4. Filter the resulting variants for the desired characteristics.
5. Annotate filtered variants.

Let us mention that each of the steps above allow for substantial customizations that in turn may fundamentally alter the outcomes in the last stage.

## What is the best method to call variants?

There is no universal rule, method or protocol that would always produce correct answers or guarantee some range of optimality. Moreover any method, even a simplistic or possibly wrong one may be able to produce seemingly correct calls when the the data is unambiguous.

Much ink and effort is spent generating and comparing results across massive datasets, you can find these as being labeled as "concordance" of SNP callers. Alas, in our finding none of these reports ever provide a universally clear picture or even advice on what works well and how to do this process correctly.

This is not to say that we don't have access to resonably reliable variant callers. Scientist have been working feverishly and radical improvements have been and are being made.

## What are the most commonly used variant callers?

There are "generic" variant callers:

- bcftools
- FreeBayes
- GATK
- VarScan2

And then there are specialized software that were designed to account for specific properties of the data, such as somatic mutations, germline mutations, family trios and many others. Specialized tools typically are able to make use of other properties of the data and produce more accurate variant calls.

# Variant calling: Ebola dataset

We will walk through an example of variant calling for the data from Zaire ebolavirus sample sequencing from the 2014 outbreak in Sierra Leone, West Africa then generalize and automate it.

Let's assume that we alredy knew the accession number for the genome studied in the `1976` outbreak called 'Mayinga'. This number is `AF086833`. We'll now make use of all the techniques that we've have learned so far to find out what is different in the genomes of the 2014 strains.

## 1. Obtain and index the reference genome.

```
# Reference genome accession number.
ACC=AF086833

# Get the reference genome.
mkdir -p refs
REF=refs/$ACC.fa

# Get the reference sequence.
efetch -db=nuccore -format=fasta -id=$ACC | seqret -filter -sid $ACC > $REF

# Index reference for the aligner.
bwa index $REF
```

## 2. Obtain the sequencing data that is to be aligned

```
# SRR run number.
SRR=SRR1553500

# Get data from an ebola sequencing run.
fastq-dump -X 100000 --split-files $SRR
```

## 3. Align the data with the aligner of you choice

```
# This will be the resulting alignment file.
BAM=$SRR.bam

# Shortcuts to read names.
R1=${SRR}_1.fastq
R2=${SRR}_2.fastq
TAG="@RG\tID:$SRR\tSM:$SRR\tLB:$SRR"

# Align and generate a BAM file.
bwa mem -R $TAG $REF $R1 $R2 | samtools sort > $BAM
samtools index $BAM
```

# How do I use `bcftools` to call variants?

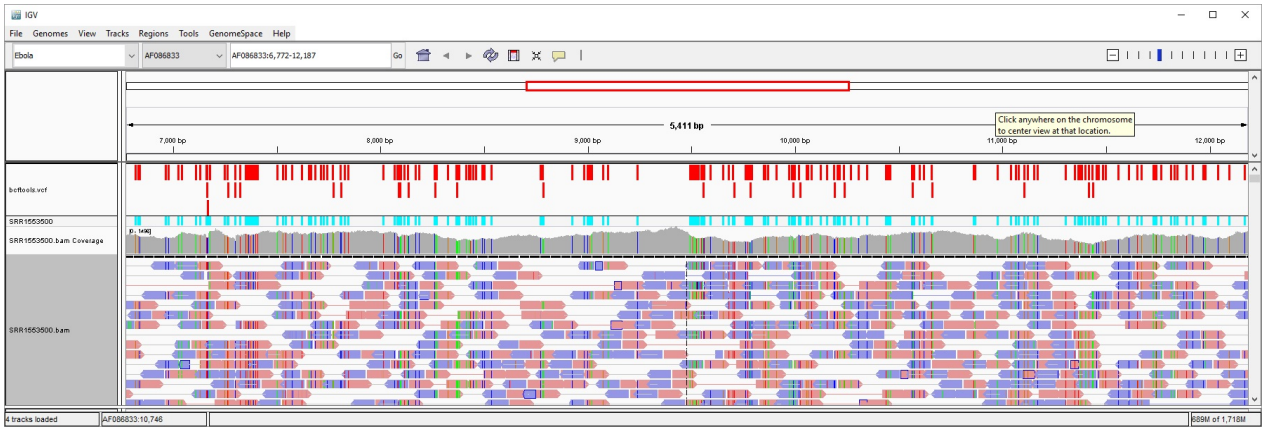
Samtools/BCFTools can be used to call SNPs in the following way:

```
# Determine the genotype likelihoods for each base.
samtools mpileup -uvf $REF SRR1553500.bam > genotypes.vcf

# Call the variants with bcftools.
bcftools call --ploidy 1 -vm -Ov genotypes.vcf > bcftools.vcf
```

You may also run both commands on one line to avoid having to store the intermediate, fairly large genotype file:

```
samtools mpileup -uvf $REF SRR1553500.bam | bcftools call -vm -Ov >  bcftools.vcf
```

Clearly the genomes have changed a lot (using Integrative Genomics Viewer to look at the BAM file):




# How do I use the FreeBayes variant caller?

On this same data the FreeBayes variant caller may be invoked as:

```
freebayes -f $REF SRR1553500.bam > freebayes.vcf
```

# What is GATK (The Genome Analysis Tookit)?

GATK is not just a variant caller - it is a toolset of epic complexity that offers features and functionality that most of us won't ever understand (or need for that matter).

The Genome Analysis Tookit is in our opinion the best variant caller, and would be our first choice in most cases had it not been for unnecessarily awkward usage and incredible mental overhead that comes with it. Every step seems to be just a tad bit more complicated than with other tools, and all the small inconsistencies add up.

To their credit, the GATK team maintains a forum where they do their best to address the myriad of (legitimate) questions that their user-unfriendly tool generates:

- Main website: https://software.broadinstitute.org/gatk/

GATK also has a "best practices" site that presents recommended workflows for variant calling using GATK.

- GATK best practices: https://software.broadinstitute.org/gatk/best-practices/

# How do I use GATK?

GATK operate via "tools". Tools are independent programs that provide a certain functionality. These are typically run as

```
java  -jar  GenomeAnalysisTK.jar  -T  ToolName  -R reference.fasta
-I inputBAM.bam -V inputVCF.vcf -o outputfile  -L  20:1000000-2000000
```

In addition a separate program named `picard` (a homage to Capt. Picard of Star Trek Nex-Generation) is often required to prepare and maniulate data.

For our example the Genome Analysis Tookit could be used with the `HaplotypeCaller` tool as:

```
# Prepare the reference with 2 tools.
samtools faidx $REF
picard CreateSequenceDictionary REFERENCE=$REF  OUTPUT=refs/AF086833.dict

# Generate the variants.
gatk -T HaplotypeCaller -R $REF -I SRR1553500.bam -o gatk.vcf
```

# How can I improve the quality of variant calls?

Most pipelines and protocols recommend a series of quality filtering steps and processes that will improve the quality of the variant calls. These are too numerous to enumerate here but can be found under titles such as:

- GATK best practices: https://software.broadinstitute.org/gatk/best-practices/

You would also need to read a few documents, papers and manuals that summarize other researcher's experiences with the exact domain of application that you interested in. For example the VarScan2 manual states that when investigating trios (2 parent + child):

> the ideal de novo mutation call will have high depth (>20x) in all three samples, with good support in the child (40-50% of reads for autosomal calls) and no variant-supporting reads in either parent."

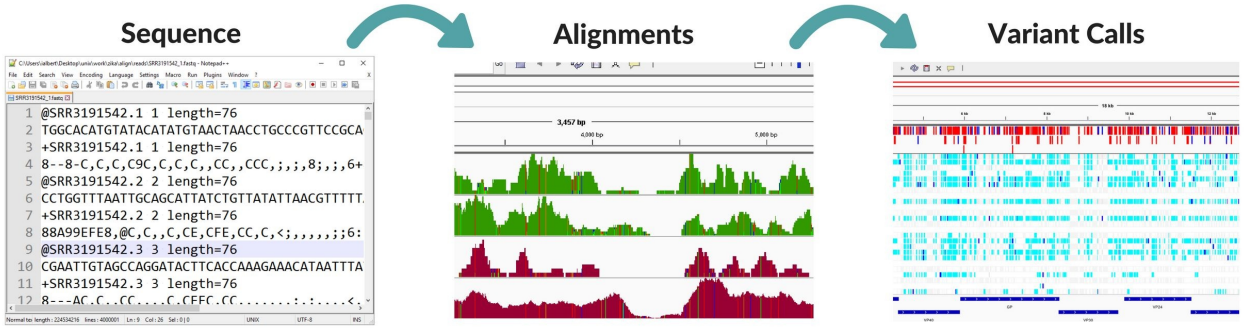
Do understand that best-practices simply improve on a process with diminishing returns but cannot, on their own, solve some of the biggest problems of variant calling.

# Can I customize the variant calling process?

Just as with aligners the process of calling variants can be greatly customized. Some parameters (e.g. ploidy), reflect certain expectations of the genome and sample collection under study and will need to be set to match those expectations.

# Multi-sample variant calling

A typical variant calling protocol follows this path:




Note: this section is a continuation of Variant calling: Ebola dataset

# How can I automate multi-sample variant calling process?

Recall that we found the SRR `runids` for the ebola project by searching the `sra` database with `PRJNA257197` and formatting the output as `runinfo`. But what if we wanted to analyze one or more of these runs? Here is where Unix scripting is the most helpful. It allows full automation of the entire process.

Get the `runinfo` and select a few files from it

```
esearch -db sra -query PRJNA257197 | efetch -format runinfo > runinfo.csv
```

Take the first five values of the first column:

```
cat runinfo.csv  | cut -f 1 -d , | head -5 | grep -v Run  > samples.txt
```

The `samples.txt` file now contains:

```
SRR1972917
SRR1972918
SRR1972919
SRR1972920
```

We have created a script that contains the instructions necessary to analyze multiple runs that are identified by just their run id. See: http://data.biostarhandbook.com/variant/find-variants.sh

Get the script

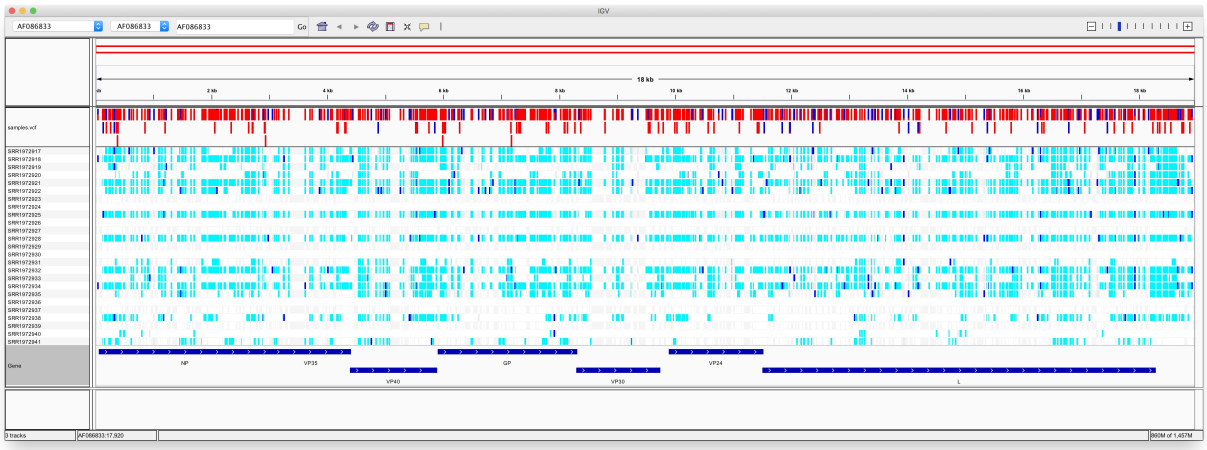
```
curl -O http://data.biostarhandbook.com/variant/find-variants.sh
```

Where the accession number and samples.txt file may contain any other values. The only limitation built into the script is the `READNUM` variable inside the script that is set allow to allow the scripts to run in seconds. In a realistic scenario one would raise that number (or remove the limitation altogether)

Run it as:

```
bash find-variants.sh AF086833 samples.txt
```

The resulting `combined.vcf` file allows us to compare all strains to the reference but relative to one another as well.



We could could also run it against a different reference genome:

```
bash find-variants.sh KJ660346 samples.txt
```

This script will download, align and call variants relative to a reference genome passed as a first parameter and using any number of SRA samples listed in the file. The script will do all the necessary steps in a repeatable and reusable manner.

1. Downloads the reference genome.
2. Prepares and indexes the reference genome for subsequent tools.
3. Downloads each dataset from SRA.
4. Aligns each downloaded dataset against the reference genome.
5. Calls variants on each alignment file.
6. Calls variants on all alignments as a group ( `combined.vcf` ).

In addition it generate the file called `runlog.txt` that contains the full log of the run.

If you are on Bash for Windows make sure to run the `wonderdump` first like so:

```
cat samples.txt | xargs -n 1 echo wonderdump | bash
```

Note how the file is generic, it allows changing the reference genome, and can run on any number of samples.

Investigate the script we used and think about what it does and what makes sense and what you'd do differently. Each script is an insight into the author's way of thinking. Your scripts will have to reflect yours.

# What is the result of running this script?

The result of running this script is a file called `samples.vcf` that contains the variants across all the selected samples.

We have uploaded the results of a run with 25 samples to the data site, that you can download:

```
curl -O http://data.biostarhandbook.com/variant/ebola-samples.vcf
```

# Why is variant calling challenging?

As you will find there is no shortage of scientific papers comparing different variant callers. What is however never adequately explained is why is this task challenging and how are various tools trying to handle the problems.

# Why is it so difficult to call variants?

Most genomics-related DNA sequences have several properties that confound intuition as they require thinking about context that human mind does not typically need to solve:

1. The DNA sequences can be exceedingly long (millions of bases)
2. Just four building blocks A/T/G/C make up any DNA sequence. These, even when distributed randomly could produce a variety of patterns just by sheer chance.
3. The DNA is only partially affected by randomness - usually, most of it has a function.
4. Depending on the organisms various segments of the DNA are affected by different rates of randomness.
5. Experimental protocols break long pieces of DNA into tiny ones that we then attempt to match back to a reference. Ideally, we hope that it still matches the most closely to the original location - because only then will our match be a variant of the site of interest.

If our fragment were to match better somewhere else than the original location our sequence would align there instead of its "correct" position.

We call this a "misalignment," though as we mentioned before the word is usually a misrepresentation of what happens. It is not actually "misaligning." From algorithmic perspective the alignment is correct, and it does what we instructed it to do.

As you can see the deck is stacked quite a bit against us - it is surprising that variant calling works as well as it does!

# How do variant calls overcome these challenges?

Nothing gets scientist's minds going better than the need to tackle a seemingly impossible problem. Unexpectedly, the very fact that we chop sequences into little pieces has an unexpected benefit. If we only had a single sequence at hand then, once a "misalignment" took place there would be little extra information to recognize and correct it.

When the sequences are short, then each is affected slightly differently - and as it turns out - it is possible to attempt to recover the "correct" alignment from all the different types of errors that affect each shorter read differently. In essence, it is feasible to find another variation that would better explain and reconcile the differences present in each read.

This process has different implementations: it may be called *realignment* or *probabilistic alignment*.

# How to explore the effect of mutations on variant callers?

We have written a script called compare-variant-callers.sh that allows us to generate variants with multiple methods. The script assumes that you have installed `bcftools`, `freebayes` and `GATK` - if you only have a subset of these installed, you will need to edit the script and comment out the sections that run programs that your computer does not have installed.

Obtain the code:

```
curl -O http://data.biostarhandbook.com/variant/compare-variant-callers.sh
```

Now run it:

```
bash compare-variant-callers.sh
```

It should print:

```
*** Obtain data for AF086833 and save it as refs/REFERENCE.fa.
*** Index refs/REFERENCE.fa with several tools.
*** Generating simulated reads from: GENOME.fa
*** Aligning the reads into: align.bam
*** Genome alignment into: global.bam
*** Calling variants with samtools.
*** Calling variants with freebayes.
*** Calling variants with GATK.
*** Run completed successfully.
*** VCF files: samtools.vcf freebayes.vcf gatk.vcf
```

It ran three variant callers and created quite a few files, but the ones we are interested in are:

1. `samtools.vcf`
2. `freebayes.vcf`
3. `gatk.vcf`

Since we have not yet introduced any mutations, none of these files should contain any mutations.

Let's change that.

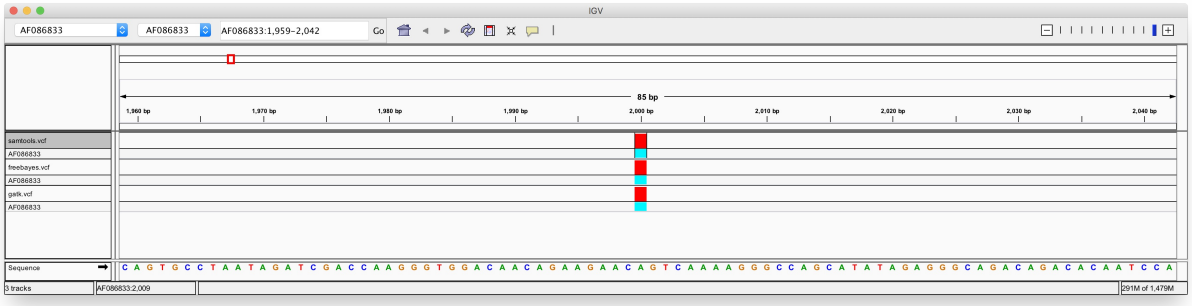
The file called `GENOME.fa` contains the genome from which the sequencing samples are generated. With a method of your choice modify and save this file. We will use the command line tool `biosed` to create a SNP replacing an `A` with a `T`. We have also selected this variation to be at position 2000 to help us more quickly identify whether the SNP caller indicates it correctly.

```
# A shortcut to the reference.
REF=refs/REFERENCE.fa

# This is the genome file that gets mutated from the reference.
GENOME=GENOME.fa

# Replace one sequence with another (the only difference is the A -> T at the end
cat $REF | biosed -filter -target AGGGTGGACAACAGAAGAACA -replace AGGGTGGACAACAGAAGAACT > $GENO
ME
```

now rerun the comparison script:

```
bash compare-variant-callers.sh
```

Here is what we get. All variant callers can reproduce and find this SNP:




# What happens when the variation is more complicated?

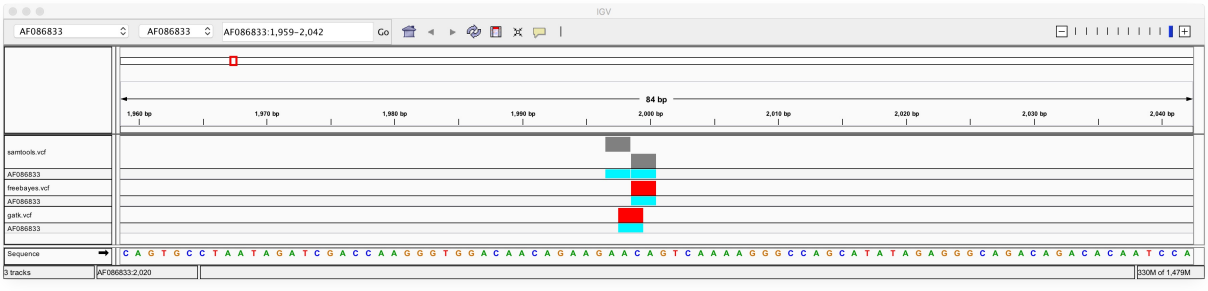
Let us cut two bases from the end of the sequence; this is now a two base INDEL

```
cat $REF | biosed -filter -target AGGGTGGACAACAGAAGAACA -replace AGGGTGGACAACAGAAGAA > $GENOME
```

now rerun the comparison script:

```
bash compare-variant-callers.sh
```

What do we get now? Each of the variant callers produces a DIFFERENT variant call.



# Variant Normalization

## What is variant normalization?

In Unified representation of genetic variants, Bioinformatics 2015 the author state:

> A genetic variant can be represented in the Variant Call Format (VCF) in multiple different ways. Inconsistent representation of variants between variant callers and analyses will magnify discrepancies between them and complicate variant filtering and duplicate removal.

For example, the same variant could be represented as

```
 GAC --> AC
TGAC --> AC
GACC --> ACC
```

Variant normalization is the process of simplifying the representation of a variant to obey the following rules:

- Represent the variant with as few letters as possible.
- No allele may be zero length.
- Variant must be "left aligned" (this also applies to repetitive regions).

A variant is "left aligned" if it is no longer possible to shift its position to the left while keeping the length of all its alleles constant. For example an insertion of C into

```
CCCC
```

would need to be reported at the leftmost position.

## Should variant callers report normalized variants by default?

Yes and eventually they will. GATK for example already produces normalized variants. The output of other tools needs to be normalized.

## How do I normalize a variant?

The `bcftools` and the `vt` tools have actions that perform variant normalization.

```
bcftools norm -f $REF samples.vcf
```

and

```
vt normalize -r $REF samples.vcf
```

# Variant annotation and effect prediction

## What is variant annotation?

Variant annotation means predicting the effects of genetic variants (SNPs, insertions, deletions, copy number variations (CNV) or structural variations (SV)) on the function of genes, transcripts, and protein sequence, as well as regulatory regions.

Variant effect annotators typically require additional information from other data sources. The predicted results will depend on the type of other data that is known about the organism or genome under study. The starting points for a variant annotator are the variants (SNPs, insertions, deletions and MNPs) usually in VCF format.

The results of variant annotation may be reported in text, HTML or pdf format along with an annotated VCF file that now contains additional information on each variant. The text/HTML reports contain substantially more information than the VCF files as this latter format limits the type of content that may be added.

## Are are consequences of "variant effects"?

Whereas the way the word is used seems to imply a "universal" effect, keep in mind that all effects are "relative" to the information that is currently present in a database.

The same way as in an alignment an insertion in one of the sequences could be reported as being a deletion in the other - it all depends on what we choose to be the reference, any effect predicted by an annotator refers to the information already known and stored in the database.

## What kind effects can variant effect predictors find?

The variant effects (sometimes called consequences) fall into multiple categories:

- Correlate the location of the variant with other genomic annotations (e.g. upstream of a transcript, in coding sequence, in non-coding RNA, in regulatory regions)
- List which genes and transcripts are affected by the variant.
- Determine consequence of the variant on protein sequence (e.g. stop_gained, missense, stop_lost, frameshift, other property changes)
- Match known variants that may have been found in other projects such as the 1000 Genomes Project.

Several terms are used to express the effect (also called consequence) of variants. The actual definition of each word is part of the Sequence Ontology that we covered in the first chapters. Also, each effect is associated with a somewhat subjective qualifier (LOW, MODERATE, HIGH) that attempts to characterize the severity and impact of the effect on a biological process.

Here are a few potential "effects":

- `stop_gained` : A sequence variant whereby at least one base of a codon is changed, resulting in a

premature stop codon, leading to a shortened transcript, SO:0001587, HIGH
- `inframe_insertion` : An inframe nonsynonymous variant that inserts bases into in the coding sequence, SO:0001821, MODERATE
- `missense_variant` : A sequence variant, that changes one or more nucleotides, resulting in a different amino acid sequence but where the length is preserved, SO:0001583, MODERATE
- `protein_altering_variant` : A sequence_variant which is predicted to change the protein encoded in the coding sequence, SO:0001818, MODERATE

Variant Effect Predictor (VEP) from Ensembl maintains a list of consequence terms and their meaning when used in VEP.

# What kind of variant annotators can I use?

This is a burgeoning field with several tools and techniques:

- VEP: Variant Effect Predictor A web-based tool from Ensembl that integrates with the Ensembl data.
- snpEff: Command line genetic variant annotation and effect prediction toolbox.
- AnnoVar: Software tool to functionally annotate genetic variants detected from diverse genomes.
- VAAST 2: Variant annotation, analysis, and search tool.

# How do I use snpEff?

We will compare two strains of the Ebola virus, and we will attempt to determine which changes are likely to impact the function of the genome.

For all annotation tools, we need to know which genome build the annotation database is using. We will need to align our data against the same genome build.

Find what databases does snpEff know about the present.

```
snpEff databases > listing.txt
```

Find the information on Ebola

```
cat listing.txt | grep ebola
```

Produces the output:

```
ebola_zaire    Ebola Zaire Virus    KJ660346.1    OK
http://downloads.sourceforge.net/project/snpeff/databases/v4_1/snpEff_v4_1_ebola_zaire.zip
```

We need to download that database to use it

```
snpEff download ebola_zaire
```

We can visualize the content of a database for example to check the chromosomal naming and have an understanding of what the database contains.

```
snpEff dump ebola_zaire | more
```

Note the version number of the accession number `KJ660346`. As a rule, we have to ensure that our alignments are performed using the same build. Using this accession number means that our effects will be in reference to this genomic build.

Get the script that we used before:

```
curl -O http://data.biostarhandbook.com/variant/find-variants.sh
```

The quality of a script becomes evident when we need to modify it to run on a different data set. Fewer the modifications we need to make, the less likely it is that we will introduce an error. Ideally we change just a tiny section or a parameter and the whole script should run with no other changes. Think about how much time and effort this saves you. Within seconds you can re-run the entire analysis on a different genome (or with different SRR data).

We also need to select SRR runs that we wish to study with the same commands that we demonstrated in the chapter Variant calling Ebola data

```
curl http://data.biostarhandbook.com/sra/ebola-runinfo.csv > runinfo.txt
cat runinfo.txt | grep "04-14" | cut -f 1 -d ',' | grep SRR | head -5 > samples.txt
```

The file `samples.txt` contains five sample run ids:

```
SRR1972917
SRR1972918
SRR1972919
SRR1972920
SRR1972921
```

Run the script:

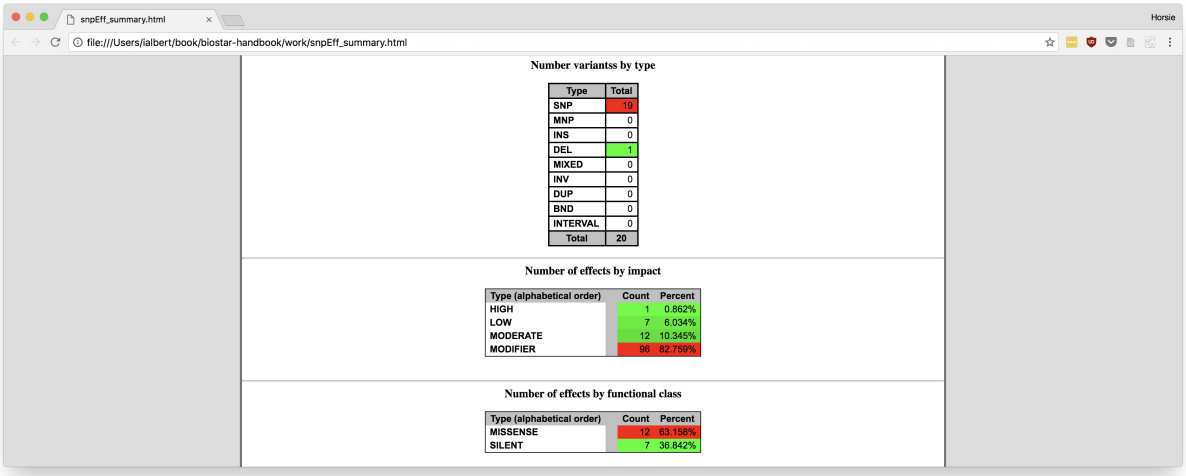
```
bash find-variants.sh KJ660346 samples.txt
```

And finally to produce the annotated results:

```
snpEff ebola_zaire combined.vcf > annotated.vcf
```

Now to show these annotations we also need to build the customn genome in IGV.

`snpEff` also generates report files `snpEff_summary.html` this is an HTML report with several entries. For example "Codon changes" shows:




# Can I build custom annotations for snpEff?

Custom annotations can be built for snpEff from GeneBank and other files. See the Install snpEff page for details.

# How do I use the Ensemble Variant Effect Predictor (VEP)

Get our subset of variants obtained from 1000 genomes project called `subset_hg19.vcf.gz` from http://data.biostarhandbook.com/variant/ or from the command line as:

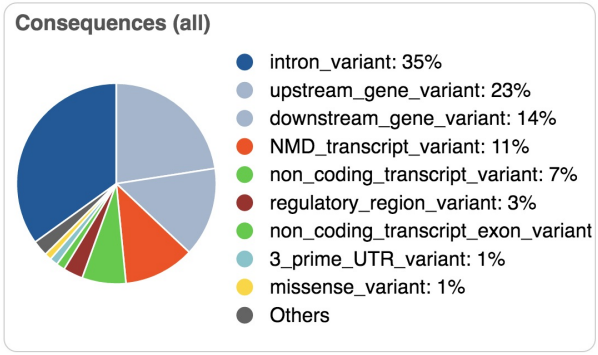
```
curl -O http://data.biostarhandbook.com/variant/subset_hg19.vcf.gz
```

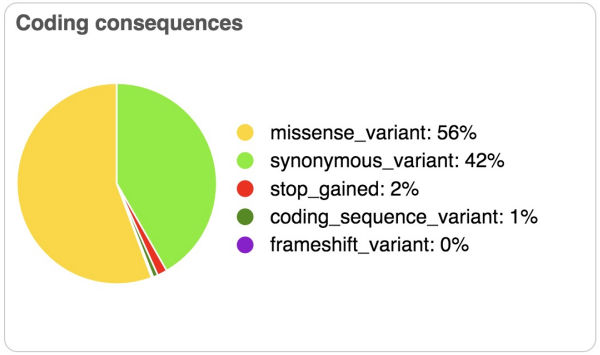
Visit the VEP website at http://useast.ensembl.org/Tools/VEP.

Just as with the Ebola data, it is essential to use the same genome build for predictions as for alignment. For human genomes, the process is more error prone since, unlike the Ebola virus, the chromosome names stay the same across builds. Hence there is no way to tell from the name of the sequence what genomic build it came from.

Needless to say, this is a source of significant and ongoing problem with reproducibility. It is exceedingly easy to misuse data and process it in the wrong context accidentally. It does not help that even the naming is inconsistent. As it happens, the UCSC genome build called `hg19` corresponds to genome build `GRCh37`. It is only starting with the latest GRCh38 release where at least the numbers will match `GRCh38` is called `hg38`).

In this case, our data was obtained by aligning against genome build `GRCh37` so we need to switch VEP to that release. Uploading the results and running VEP produces a report with quite a bit of detail.



# The RNA-Seq puzzle

I thought up this problem while preparing a lecture on RNA-Seq data analysis. In the spur of the moment, I assigned it as a homework problem. Most students liked it - and I received quite a bit of positive feedback on it. I think that it reads almost like a puzzle in the Sunday paper.

Later I came to believe that this puzzle provides the means to assess how well one understands RNA-Seq analysis. When you can solve the puzzle it means that you understand how RNA-Seq analysis works behind the scenes - what assumptions it makes and how various effects manifest themselves in the data.

## How would I even solve this puzzle?

It is ok if you can't solve this puzzle yet. Read the next chapters, then come back here. We've put it here to show you where we are going.

The purpose of this "puzzle" is to get you to think about what the numbers mean, and how they are linked together -- and what it's like to obtain consistent and real data.

But, we admit it is also an unusual setup, because it reverses our typical thought process. In a usual RNA-Seq analysis, you are given data, and you are asked to determine the gene expression from it. In this puzzle, you will be told what the gene expressions are and you have to *make the data* that supports those statements, then solve it to show that your numbers do indeed work as intended.

Note -- you don't need a computer to solve this problem. Just paper and pencil, or a text file. A calculator might help later.

## The Pledge

Imagine that you have an organism with three genes `A`, `B`, and, `C`.

- `A`, with a length of 10bp
- `B`, with a length of 100bp
- `C`, with a length of 1000bp

You want to study this organism under two conditions:

- Wild type: `WT`
- Heat shock: `HEAT`

You know from other sources that, in the `WT` condition, gene `A` expresses at levels that are twice as high as gene `B`.

You also know that only one gene's expression (but you don't know which gene) changes between `WT` and `HEAT` conditions. Assume that the change is substantial enough to be detectable. The rest of the genes are expressed at the same level in `WT` and `HEAT`.

Imagine that you have performed an RNA-Seq experiment to compare the wild-type `WT` and treatment `HEAT` - with just one replicate per experiment.

You have made one mistake, however. You have mixed the samples incorrectly, and you ended up placing twice as much DNA for the `WT` than for the treatment `HEAT`. You can still tell the samples apart since they are barcoded. You just mixed and sequenced twice as much DNA (mRNA) for `WT` as `HEAT`.

# The Turn

Pick the numbers for read coverage that accurately represent the situation above. You can make up the numbers - the goal is to make them express what you know based on the description above. Create a 3x2 count table that shows your made up read counts. Each `?` will need to have a number. So you have to come up with six numbers. That's the puzzle!

```
ID   WT   HEAT
A    ?     ?
B    ?     ?
C    ?     ?
```

# The Prestige

Show that your numbers work. When you can answer them all, you understand how RNA-Seq works.

- How can you tell from your data that you placed twice as much `WT` material in the instrument?
- What is the `CPM` for each gene under each condition?
- What is the `RPKM` for each gene under each condition?
- What is the `TPM` for each gene under each condition?
- How can you tell that gene `A` expresses at twice the level of gene `B` within the `WT` sample?
- Can you tell which gene's expression level changes between `WT` and `HEAT`?
- Is the puzzle always solvable when correct values are specified in the "Turn"?

Now think about this:

- How many reads would you need to sequence for the `CPM` to be a "nice" number.
- How many reads would you need to sequence for the `RPKM` to be a "nice" number.
- How many reads would you need to sequence for the `TPM` to be a "nice" number.
- Does it make any sense to introduce measures like these above, that have arbitrary scaling factors, just to make numbers look "nice"?

# RNA-Seq Analysis

## Why should I consider RNA-seq (over alternatives, e.g. microarray, qPCR)?

The invention of oligo-based microarrays allowed interrogation of multiple genes (and/or the entire "known" gene complement) from a genome at once. Microarrays opened the door for new opportunities of quantitative and diagnostic applications of gene expression. RNA-seq extends that functionality by allowing more accurate quantitation of gene expression compared to microarrays (think `digital` vs `analog`). RNA-seq has a much wider dynamic range compared to microarrays (10^5 vs 10^3).

RNA-seq is not immediately replacing microarrays as the method of choice for estimation of gene expression. In specific cases (e.g., a narrow research project restricted to well known/characterized genes), microarrays may still provide an effective alternative; they also have low cost per sample and short turnaround time.

## What is quantifying?

While the first applications of sequencing were designed to ascertain the nucleotide composition of a target DNA molecule, over time a radically different use emerged, one that has become increasingly more useful for life scientists. Collectively we call these type of applications as "quantifying via sequencing."

In practice, this means that sequencing is used to determine not the composition of DNA fragments but their abundance. The usefulness of this lies in that when we have biological processes affecting the abundance of different DNA fragments, then by measuring these abundances, we measure the natural process itself.

## What are gene isoforms?

Gene isoforms are mRNAs that are produced from the same locus but are different in their transcription start sites (TSSs), protein-coding DNA sequences (CDSs) and untranslated regions (UTRs), potentially altering gene function.

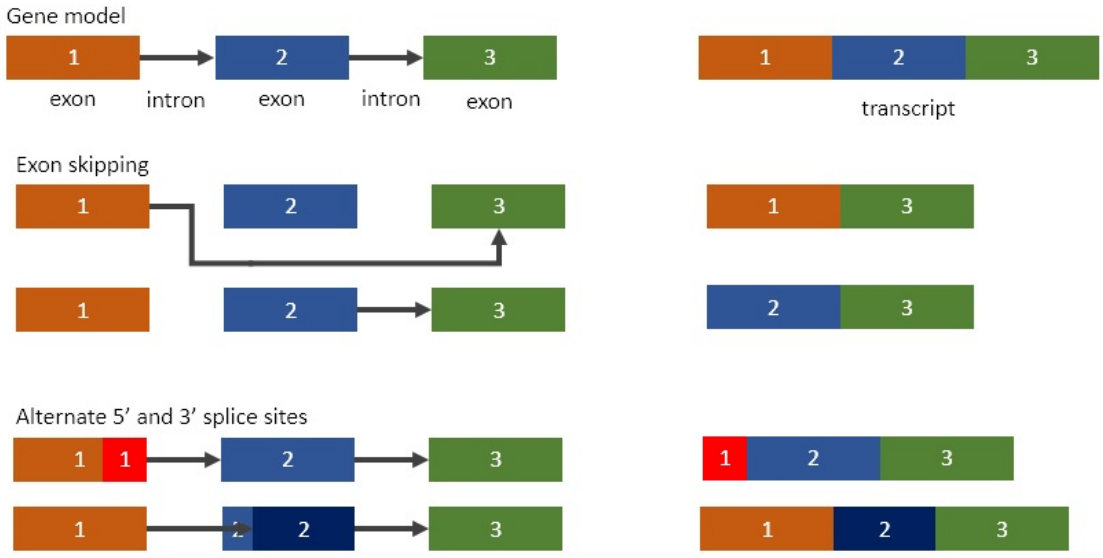
Above we show three isoforms of a gene is are composed of three exons.

# What kind of splicing events exist?

Five primary modes of alternative splicing are recognized:

1. Exon skipping or cassette exon.
2. Mutually exclusive exons.
3. Alternative donor (5') site.
4. Alternative acceptor (3') site.
5. Intron retention.

A few of these examples shown below:




# How does RNA-seq analysis work?

The RNA-seq protocol turns the RNA produced by a cell into DNA (cDNA, complementary DNA) via a process known as reverse transcription. The resulting DNA is then sequenced, and from the observed abundances of DNA, we attempt to infer the original amounts of RNA in the cell.

Conceptually the task looks simple. It seems that all we need to do is count is how many DNA fragments are there. If we get higher numbers under a particular condition, it means that the cell must be producing RNA at higher levels. In reality, the problems of counting and comparing are more complicated.

# What is the common outcome of an RNA-Seq analysis?

The goal of most RNA-Seq analyses is to find genes or transcripts that change across experimental conditions. This change is called differential expression. By finding these genes and transcripts, we can infer the functional characteristics of the different conditions.

# What complicates RNA-Seq analysis?

First, what we measure via sequencing is almost always a relative measure. To use a term from statistics we draw a "statistical sample" from the total amount of DNA. When we do that, we measure relative ratios. This means that the same absolute amount of one kind of RNA may appear at different abundances merely based on the presence and abundances of other types of RNAs that are present.

For example, say you have 10 As and 10 Bs in a population and you've estimated their abundance via sampling. The abundance of A would turn out to be 50%. But if you had 10 As, 10 Bs and 10 Cs in the next experiment, the abundance of A would be shown as 33.3% of the total even though the same amount of gene A was present. If you did not know how many categories there were to begin with, you could be led to believe that the abundance of A decreased in the second experiment whereas only the relative amount actually decreased.

Second, most current sequencing instruments measure only a short segment of each fragment. Transcription in higher eukaryotes makes use of splicing where introns get removed, and exons are joined in different configurations. Since the splicing of different exons will form similar transcripts, it can be difficult to match a sequencing read to its most likely originating transcript.

Third, unlike DNA, which is static, the mRNA abundances change over time. You will need to ensure not only that you observed a change but that this change is correlated with the experimental conditions. Typically this is achieved by measuring the same state multiple times.

The process of repeating the same measurement is called replication. The differential expression is valid when the changes between replicates of one condition are smaller than the differences between replicates across conditions. There are several methods to compare replicates and estimate the magnitude of changes.

Several competing methods have been proposed to account for these problems. Also, there are several (too many perhaps) scientific publications that attempt to finally put these questions to rest and come up with a recommendation for the "best" method. Alas, the results of these papers are often underwhelming, and end up with "it depends" and hand-waving statements that defer responsibility back to the reader.

Just as with variant calling, you have to recognize that the information encoded in the data significantly affects the performance of a given method. For some types of data, it does not matter **at all** what method you pick. For other datasets each process produces radically different results - and you have to decide which one to trust.

# How many replicates do I need?

While we can make some inferences from a single measurement, in the current state of the field the minimal number of replicates typically recommended is three, and five is even better. As a general rule of thumb when you expect the effects to be subtle and biological variation significant (e.g. experiments with live animals) more replicates are better than fewer. Note that by generating five replicates across two conditions, each producing two files you are already starting out with 20 FASTQ files being produced by the sequencing instrument. The ability to automate processes is more important than ever.

# Will there ever be an optimal RNA-Seq method?

There is an analogy to a mathematical paradox called the voting (Condorcet) paradox. This paradox states that when tallying votes for more than two options, we could end up being in conflict with the individual wishes of a majority of voters, and there is no optimal way to count the votes that would avoid all possible conflicts.

When RNA-Seq methods assign measurements to genes, a sort of voting scheme applies. Because often there are ambiguities on how to resolve a measure, decisions have to be made to allocate it to its most likely origin ("candidate") transcript. Unsurprisingly the Condorcet paradox will apply, and no matter what method we choose it is possible to count reads in ways that do not correspond to reality.

What is the take-home message here though? We believe that the explanation is as simple as it is counter-intuitive:

> There is no best method for RNA-Seq. There are only methods that are *good enough*. The meaning of *good enough* evolves over time. Good enough is a method that will likely be able to identify the gene expression changes if they exist and are relevant to the study.
>
> But there is always a chance that you'll have unusual phenomena to quantify for which your initial choice of method is not optimal. The most important skill then is to recognize it.
>
> You will need not just to learn how to perform and RNA-Seq analysis but to understand when a process does not seem to produce reasonable or correct results.
>
> Tip: You understand RNA-Seq when you can easily analyze a dataset with at least two different methods.

In our opinion, the usability and documentation of a method are among its most essential ingredients. Treating an RNA-Seq analysis as a black box is a recipe if not for disaster then for superficial and uninformative conclusions.

Having a clear understanding of the processes will allow you to make informed decisions. You will see that it is not that difficult to use entirely different methods to analyze the same data. What is more challenging is to make an informed decision.

# What is the first decision when performing an RNA-Seq analysis?

In general, the most important distinguishing factor between methods is the choice of the reference frame:

1. You may want to quantify against a **genome**, a database of all the DNA of the organism
2. You may choose to compare against a **transcriptome**, a database of all known transcripts for the organism.

Using a genome allows for the discovery of potentially new transcripts and gene isoforms. Using a transcriptome usually means more accurate quantification but only against a predetermined "ground" truth.

# What are the steps of an RNA-Seq analysis?

An RNA-Seq analysis is composed of three separate stages:

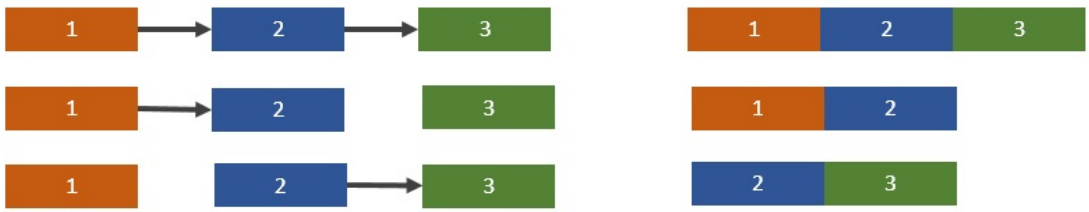
1. Assign reads to transcripts (alignments or mapping).
2. Estimate abundances (read counting).
3. Compare abundances (differential expression).

There is a lot of disagreement on what the "correct" way to do each of these steps is, leading to numerous alternatives, each with its own benefits and trade-offs. Also, we sometimes mix and match different methods if it appears appropriate for a given data set.

Depending on the implementation details, some of these stages may be combined into what seems to be a single action. For example `kallisto` will perform assignment and abundance estimation in a single step, or `cuffdiff` will carry out the read counting and differential expression computation as a single operation.

# What is a splice-aware aligner?

A coding region of the DNA may produce multiple gene isoforms. Since we sequence DNA fragments by breaking the transcripts into smaller fragments, matching reads to exons alone is usually insufficient to establish which transcripts were present. For example, if our gene model supports the following three transcripts:




Then matching to exon 1 would only inform us that we have a transcript from one of the first two pairs, whereas matching to exon 2 would not help at all to resolve the question of which transcripts are present.

The reads coming from the locations that join two exons (these are the so-called splice sites) will align to the genome with a large gap (the intron) between them. These are so-called spliced alignments. Thus, to align our RNA-Seq data to a genome, we need to use aligners that are "splice-aware". These aligners are able to align reads with very large gaps in between without penalizing the alignment.

It is only the spliced reads, a tiny subset of all reads, that carry the information of which neighboring exons were joined. But even these joins will only resolve immediate neighbors, and can't connect second or third neighbors. Hence, when aligning short reads, we will be unable to resolve the entire transcript from measurements alone. All we can see are junctions that tell us partial information.

Tools then estimate which transcripts exist by building a mathematical or probabilistic model to form the simplest possible arrangement that could explain all the observed junctions. As you might suspect, life rarely works by formulas - an organism wants to survive. While there are forces that drive so-called "parsimony", this property is not required for the functioning of a cell.

Recall how aligners differ in far more than just producing the alignment. This is even more true for splice-aware aligners. There are even more features and parameters to consider. Thus different methods and software often show surprising and unexpected differences when paired up with one another.

# Which splice-aware aligner should I use?

Our recommendation for splice aware aligners are:

- HiSat2: similar algorithms as Bowtie2/Tophat2 but performs at much faster speeds
- Subjunc: designed specifically for gene expression analysis.
- BWA: the MEM algorithm of bwa supports spliced alignments.
- BBMap: Is capable of detecting very long splicing.
- STAR: Really fast, produces counts for you too.

As you can see, we are in just the first phase, yet the number of alternatives is already growing rapidly. Later sections of the book will provide examples for the same analysis with different approaches.

# How do I quantify mRNA abundances?

The history of estimating the magnitude of gene expression is short and convoluted. You may be surprised to learn that even today there is quite a bit of disagreement regarding what value we should compute, how to quantify, and how to compare the amount of mRNA in different samples. The most common measures used currently are:

1. Counts: The number of reads overlapping with a transcript.
2. RPKM/FPKM: Reads/Fragments per kilobase of transcript per millions of read mapped.
3. TPM: Transcripts per million

Again the rule of **good enough** applies. You need to choose a method that is good enough for your purpose. The challenge, of course, is how do you know when a method is good enough? Understandably most scientist define the good enough method as the one "that seems to work."

# How do I compare mRNA abundances?

The troubled history of quantifying mRNA abundance goes hand in hand with the efforts to come up with a measure that can also be used to compare the levels of expression of transcripts. Two types of comparisons may be necessary, and different criteria should be used for each:

1. *Within-sample* comparisons. In this case, we compare the expression of genes within the same experiment. For example: in this experiment does gene A express at a higher level than gene B?
2. *Between-sample* comparisons. In this case, we compare the expression of genes across experimental conditions. For example, has the gene expression for gene A gene changed across different experimental conditions?

# RNA-Seq Terminology

## What is a sample?

The commonly used word "sample" in sequencing lingo means a biological sample (extraction) that is subjected to sequencing. When we talk about samples we assume that these are grouped into conditions and within each conditions we have replicates. For example 10 samples over 2 conditions with 5 replicates per condition.

Not to be confused with "statistical sample" -- that means measuring just a subset of a population.

## What is normalization?

As we assign values to the same measures in different samples it is essential that these values are comparable across the samples. The process of ensuring that these values are expressed on the same scale is called *normalization*. There are several different normalization methods that operate on different types of assumptions, we describe a number of these in the following sections.

## What is a library size normalization?

The term "library size" is a frequently used but improper term that usually means to correct for the differences in sequencing coverage (depths) across experimental measurements. For example, it might be that for experiment A we have loaded twice as much material into the instrument than for experiment B. Your quantification methods need to be able to detect and account for this. In most cases this is handled for you by the software that you will use - the only time you may need to remember this is when visualizing data in its original form.

## What is the effective length?

The sequencing coverage drops towards the ends of DNA fragments due to the lower chances of producing fragments that cover the ends. For genomes this is usually less of a problem but when comparing transcripts these edge effects will affect shorter transcripts more than longer ones. A correction is usually applied to account for this effect - most software will compute a so called "effective length" for each feature.

## What gets counted in a gene level analysis?

The gene level analysis treats every gene as a single transcript that contains all exons of the gene. In some cases this is sufficient to get some useful results. In other cases a more fine-grained analysis is necessary wherein abundances are quantified at the transcript level.

# What is the RPKM?

If we wanted to compare the number of reads mapped to a transcript to another transcript, we have to account for the fact that longer transcripts will produce more DNA fragments simply because they are longer.

If `N` were the total number of reads mapped to a transcript, and `C` were the total number of reads measured, we cannot just take `N / C` as our measure of gene expression. A single copy of a longer transcript can be broken into more fragments (larger `N`) that a single copy of a shorter transcript.

Instead, you can divide the apparent gene expression by the length of the transcript:

```
gene expression = N / C  *  1 / L
```

If you consider `Ni` to be the reads mapped to transcript `i` then `C = sum Ni`. Basically we first compute the fraction of the total that maps to a transcript then divide that by the length of the transcript.

This number would typically be very small since just a small subset of reads of the total will align to any gene and we are dividing by the gene lenghts that could be large numbers. Other sciences solve this by using words such as *nano*, *pico* etc. to indicate the scale of the number.

Those who came up with the concept of RPKM yearned to create a more "user friendly" representation to avoid "confusing biologists" and decided that the best way to get there will be to express `L` in kilobases (`10^3`) and `C` in terms of millions of reads (`10^6`) in essence adding a factor of a billion to the formula above:

```
RPKM = 10^9 * N / L  * 1 / C
```

Hence a weird unit was born the RPKM, that, in our opintion only ends up being more confusing than other alternatives. In addition those with training in, say physics, will note immediately that the RPKM as defined above has a "dimension" to it. Whereas the `N` and `C` are integer numbers the `1/L` is an inverse of a distance. Which should give everyone pause. Why is it appropriate to measure gene expression by a quantity that is some sort of inverse of a distance?

Life would have been a whole lot simpler if the original group of scientists that invented this measure would have just called this quantity a *pachter*, as a hat tip to Lior Pachter, one of our favorite academic rebels. RPKM would have been called a *pico-pachter* and may have turned out to be a better understood quantity.

As we learn more about its limitations, the consensus appears to be that RPKM is not a good measure of gene expression and its use should be curbed and eliminated. Yet, as you will see, some of the most commonly used software continues to produce results in RPKM.

# What is TPM?

One limitation of RPKM is that ignores the possibility that new and different transcripts may be present when experimental conditions change. Its dimension of 1/distance indicates that instead of being a quantity that indicates amounts, it is a quantity that depends on the distance of something. Values can be

only compared when that "distance" is the same. As it turns out that "distance" that RPKM tacitly assumes to be the same is the total transcript length. It assumes the reads are distributed over the same "distance" of DNA. A different normalization should divide not with `C` but a value `T` that accounts for the different lengths.

```
gene expression = N / L * 1 / T
```

A way to incorporate both the number of counts and the length into T is to sum the rates:

```
T = sum Ni/Li
```

where `i` goes over all observed transcripts and `Ni` are the reads mapped to a transcript of length `Li`. Not to be outdone by **RPKM**, a new measure was born, this time called the **TPM** where we multiply the above by a million to protect *poor biologists* from the unbearable mental overload of having to divide by a million:

```
TMP = 10^6 N / L * 1 / sum(Ni/Li)
```

Life would have been a whole lot simpler if the original group of scientists that invented this measure would have just called this quantity a *salzberg*,as a hat tip to Steven Salzberg the Greatest Bioinformatician of All Time (GBOAT). The TPM would have been called a *milli-salzberg* and may have turned out to be a better understood quantity.

Since there is a distance dimension both in the numerator and denominator the TPM is dimensionless (unlike RPKM).

# What is TMM (edgeR) normalization?

Trimmed mean of M values (TMM) normalization estimates sequencing depth after excluding genes for which the ratio of counts between a pair of experiments is too extreme or for which the average expression is too extreme. The edgeR software implements a TMM normalization.

# What is DESeq normalization?

The DESeq normalization method (implemented in the DEseq R package) estimates sequencing depth based on the count of the gene with the median count ratio across all genes.

# Do I always need an advanced statistical analysis?

Surprisingly, the answer is no. The methods that need to be employed depend on the goals of your experiment.

If, before starting the experiment, you knew which gene you wanted to know more about and you care only about this gene, then the law of large numbers works in your favor. This is to say that it is very unlikely that your gene of interest was singled out by chance and was affected in a way that misleads you. The simplest of statistical tests and common sense suffice. Unfortunately, this is an uncommon situation.

If you did not know which genes might change and you wanted to reliably determine that out of tens of thousands of alternatives and their combinations then more sophisticated methods are necessary to ensure that whatever change you observe was not caused by random fluctuations or natural variation in the data.

# What is a "spike-in" control?

The goal of the spike-in control is to determine how well we can measure and reproduce data with known (expected) properties. A common product called the "ERCC ExFold RNA Spike-In Control Mix" can be added in different mixtures. This spike-in consists of 92 transcripts that are present in known concentrations across a wide abundance range (from very few copies to many copies).

# How should I name samples?

With RNA-seq analysis you may need to work with many dozens of samples. One of the skills that you have to develop is to parse file names and connect them to known sample information. File naming practices vary immensely but having an odd naming scheme can be the source of unexpected challenges. We suggest the following practices:

1. Make each attribute of the data be represented by a single, isolated region of the name.
2. If the there is a hierarchy to the information then start with the most generic piece of information and work your way back from there.

For example this is an appropriate naming scheme.

```
HBR_1_R1.fq
HBR_2_R1.fq
UHR_1_R2.fq
UHR_2_R2.fq
```

The first unit indicates samples: `HBR` and `UHR`, then comes the replicate number `1`, `2` and `3`, then the paired files `R1` and `R2`.

A bad naming scheme would be one such encodes more sample specific information into the sample without grouping them properly:

```
HBR_1_Boston_R1_.fq
HBR_2_Boston_R1_.fq
UHR_1_Atlanta_R2_.fq
UHR_2_Atlanta_R2_.fq
```

Above both `HBR` and `Boston` represent sample specific information whereas `1` and `2` are replicate specific information. These file names will make your programs more difficult to write. The names are less well suited to be generated automatically and you'll have to work around this limitation under the most unexpected circumstances.

In a nutshell, it is much easier to automate and summarize processes when the sample information is properly structured. You'd be surprised how few data analysts understand this - only to end up with programs that are more complicated than need to be.

Simplicity is key to success!

# RNA-Seq statistics: R and Bioconductor

## Why does statistics play a role in RNA-Seq?

Whereas alignments or counting overlaps are mathematically well-defined concepts, the goals of a typical experiment are more complicated. The data itself may be affected by several competing factors as well as random and systematic errors. Statistics gives us tools that can help us extract more information from our data and can help us assign a level of confidence or a level of uncertainty to each estimate that we make.

## When do I need to make use of statistics?

In the most simple formulation, one that likely draws the ire of some, statistics starts once you have a table of numbers, or a matrix. Then, in most cases, interpreting the information in either a row, a column or a cell needs to be done in the context of those other numbers in the table.

```
   ID         Condition 1         Condition 2          Condition3
SEPT3     1887.75036923533     81.1993358490033     2399.647399233
SEZ6L     1053.93741152703    530.9988446730548     211.73983343458
MICALL1   136.421402611593    197.470430842325      120.9483772358
```

A statistical test is a process by which you make quantitative decisions about the data.

## What kind of questions can we answer with a statistical test?

Here is a selection:

- How accurate (close to reality) are these results?
- How precise (how many digits are meaningful) are the values?
- For which gene do values change between conditions?
- For which gene is there at least one change across conditions?
- Are there genes for which there is a trend to the data?
- What kinds of trends are there?
- Which genes vary the same way?

Statistical tests operate with principles such as margins of error, probabilities of observing outcomes and other somewhat indirect measures. These measures can easily be misinterpreted and scientists often do make mistakes when summarizing and reformulating statements derived from statistical tests. For an example, see the section on p-values later on this page.

## What types of statistical tests are common?

The pairwise comparison is one the most common and conceptually simple tests. For example, a pairwise comparison would compare the expressions of a gene between any two conditions. A gene that is found to have changed its expression is called differentially expressed. The set of all genes with modified expression forms what is called differential expression (DE).

Here, it is important to understand the type of results that a pairwise comparison produces. Almost always you want to answer the question of whether a gene's expression level has changed. But instead what you will typically obtain is the probability that there was no difference between conditions (i.e., the probability of the null hypothesis).

When this probability is low (small) you can reject the null hypothesis, and you conclude that there is a change. The expression "reject the null hypothesis" may seem like mincing words or trying to sound clever. But when further investigations are necessary it is important to use these results in their proper context and meaning.

# Do I need to involve a statistician?

Ideally, of course, the answer is yes.

But we're in the practical advice business here and, in reality, it is not always all that easy to find a good statistical collaborator. It would be a mistake to oversimplify statistics into a purely procedural skill: "any statistician can do it." You would need to find a collaborator who understands the characteristics of biological data as well as the challenges of working with it.

We believe that understanding and performing a pairwise analysis is well within anyone's reach and in this book we will provide you with alternative ways for doing it. We also plan to show (though this chapter is not yet completed) a more advanced data modeling example. The challenge with the advanced cases is that what you learn from them may not be transferable to your problem.

For more sophisticated data modeling, for example, time course analysis or comparing several samples at once, you would either need a statistical collaborator or you will need to spend some time and effort understanding the statistics behind it. Statistics is not as complicated as it looks -- so don't be discouraged. There is a wealth of information available on more advanced statistical modeling and more resources than on bioinformatics in general.

# What is R?

Unlike most other approaches, where we install and run command line tools, statistical analysis in general and the differential expression detection in particular, are typically performed using packages that run within the `R` programming environment: The R Project for Statistical Computing

Learning how to program in `R` is not easy. Those who claim otherwise are probably the lucky ones whose minds just happen to fit `R`.

You see, the `R` language was designed before computer scientists understood how a programming language should work, what features it should have, and what data types are useful. So don't kick yourself if you can't seem to easily learn R, it is without a doubt harder to learn than many other computational

languages.

That being said, thankfully, it is far less complicated to learn how to run tools written by others. As with other programming languages, an `R` script is simply a list of commands instructing `R` to perform certain actions.

# How do I install R?

If you have conda installed the simplest is to install it with that:

```
conda install r
conda install -y bioconductor-deseq bioconductor-deseq2 bioconductor-edger r-gplots
```

Use the `Rscript` command to run `R` as a command line program.

# Are there other options to install R?

You can also install R with a downloadable installer, see the R Home Page.

If you work in R, a good option (especially when learning) is to run it via RStudio, a graphical user interface to R.

# What is Bioconductor?

Bioconductor https://www.bioconductor.org/ is a project that collects R-based code for genomic data analysis.

If you installed `R` with `conda` you are ready to go, but if you have installed `R` from a binary package paste the following into the command interface to install Bioconductor.

```
# Install the heatmap.2 package.
install.packages("gplots")

# Install the Bioconductor modules.
source("http://bioconductor.org/biocLite.R")
biocLite("DESeq")
biocLite("DESeq2")
biocLite("edgeR")
```

The commands above will churn for a while and will install all the required packages necessary for the subsequent commands. To run a differential expression study we need data and an `R` program that does the job.

# How can I run RNA-Seq differential expression scripts from the command line?

For this book, we have developed multiple R scripts that make use of several methods and can be run from the command line.

- A script called `deseq1.r` that makes use of the DESeq method.
- A script called `deseq2.r` that makes use of the DESeq2 method
- A script called `edger.r` that makes use of the EdgeR method

Each of these scripts can be obtained and run from the command line. To obtain the scripts execute:

```
curl -O http://data.biostarhandbook.com/rnaseq/code/deseq1.r
curl -O http://data.biostarhandbook.com/rnaseq/code/deseq2.r
curl -O http://data.biostarhandbook.com/rnaseq/code/edger.r
```

To see how these work we need a file with sample counts of known proportions, for example as those described in Analyzing control samples. We set this up so that you can continue on here by getting the data:

```
curl -O http://data.biostarhandbook.com/rnaseq/code/counts.txt
```

This file has the following content:

```
ERCC-00002      37892   47258   42234   39986   25978   33998
ERCC-00003      2904    3170    3038    3488    2202    2680
ERCC-00004      910     1078    996     9200    6678    7396
...
```

Most of our scripts read their input from the standard input and write to the standard output. So they will be run in the "middle":

```
cat mydata.txt | Rscript do-something.r > output.txt
```

The differential expression scripts also take a so-called design parameter that describes how many columns correspond to each condition.

For example, the data above was summarized over 6 count columns, where the first 3 columns correspond to the first condition, and the second 3 columns correspond to the second condition. We will call this a `3x3` design and indicate that as such. So you could run either of our scripts as:

```
# Analyze the counts with DESeq1.
cat counts.txt | Rscript deseq1.r 3x3 > results_deseq1.txt

# Analyze the counts with DESeq2.
cat counts.txt | Rscript deseq2.r 3x3 > results_deseq2.txt

# Analyze the counts with EdgeR.
cat counts.txt | Rscript edger.r 3x3 > results_edger.txt
```

You may be wondering -- why do these scripts use the '|' to communicate? Why all that redirection? You will see that this makes your life simpler as you don't have to remember where to put each file, is it the first or second.

It is very clear: data enters on the left and the output comes out on the right.

# Do I need to run all three scripts?

No. Pick one method - though we ourselves had a hard time deciding. We'll talk more about these choices on the How to choose an RNA-Seq analysis page.

# What are all those extra files there?

When you run say `deseq1` tools, you will see that it produces a file called `norm-matrix-deseq1.txt`. It will be named similarly for other methods.

This file is usually not even shown to the end user yet it carries quite a bit of importance. Therefore our scripts produce it for you.

```
ERCC-00002      52501.03  47122.57  45819.00   27185.80   29896.75   28304.06
ERCC-00003      4023.62   3160.91   3295.87    2371.43    2534.16    2231.15
ERCC-00004      1260.84   1074.91   1080.54    6254.92    7685.36    6157.32
```

The counts in this file come from the original matrix, but are "normalized." As you will learn later, "normalize" means that the numbers in this matrix were brought to be on the same scale and comparable to one another. It is worth looking at the original matrix to see what the initial values were.

```
ERCC-00002      37892     47258     42234     39986       25978      33998
ERCC-00003      2904      3170      3038      3488        2202       2680
ERCC-00004      910       1078      996       9200        6678       7396
```

If you want to compute a new measure on your data, or if you want to visualize or derive a quantity that you were interested in you would have to use the normalized matrix and not your original data.

# How can I plot my normalized matrix?

First of all, plotting is the act of doing science - so no one can plot for you. You have to find the charting tool and technique that works for you and learn to use that. Some people plot in R, others in Python, and so on. As you always need to tweak the plot, change colors, labels etc., we can't give you a script that does it all.

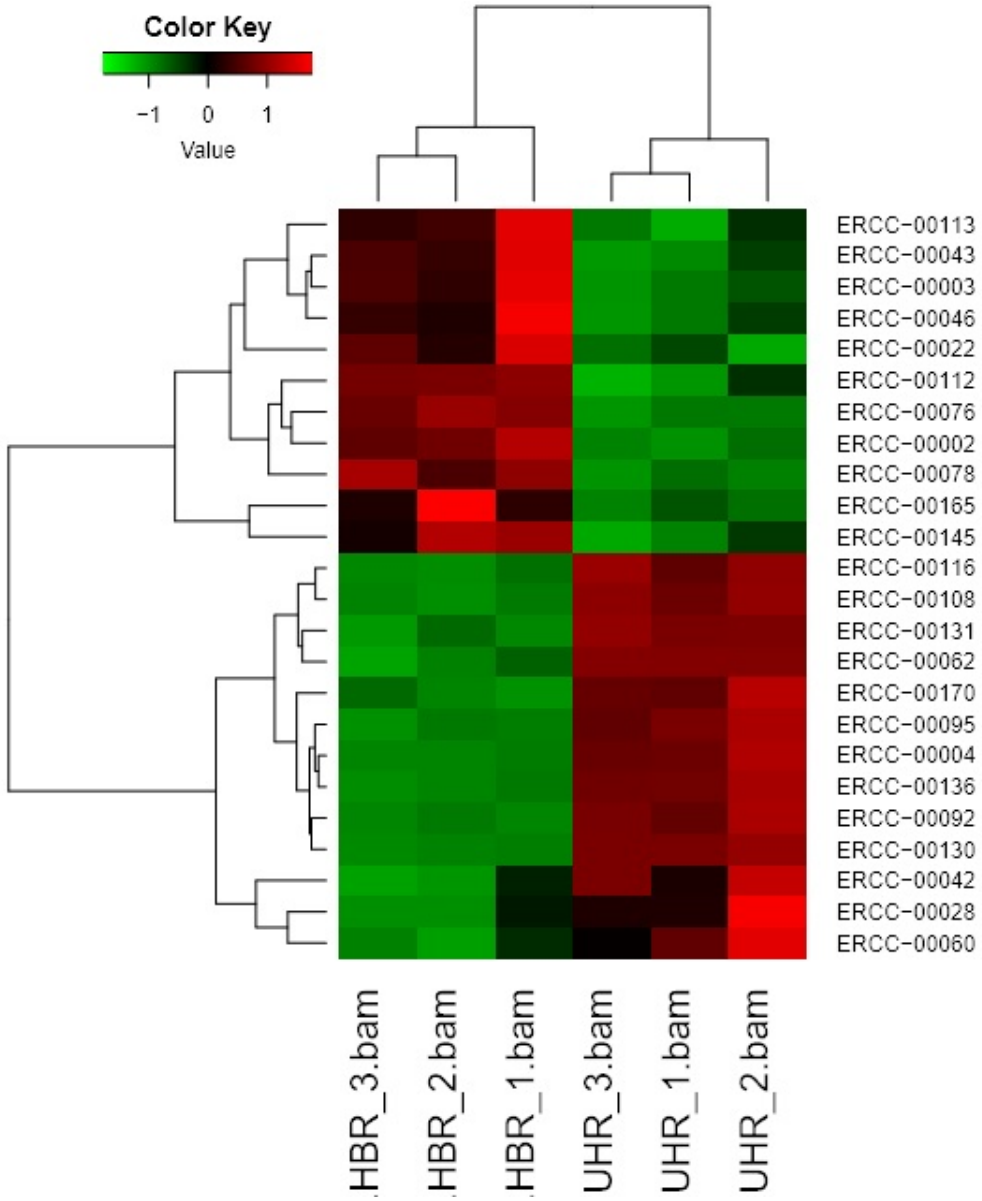
But one particular plot, the clustered heatmap is so commonly used and useful, yet so non-trivial to achieve, that we included a script that does it for you.

```
curl -O http://data.biostarhandbook.com/rnaseq/code/draw-heatmap.r
```

This script produces a PDF output on the standard output. Run it as:

```
cat norm-matrix-deseq1.txt | Rscript draw-heatmap.r > output.pdf
```

Note how this script also writes to standard out, though this time it writes a PDF file. What is in this PDF file? It is a hierarchically clustered heatmap image of your samples and differentially expressed genes:




Use this script to very quickly visualize what your data looks like.

# What does a p-value mean?

You probably don't know what a p-value means. Don't sweat it. We have yet to meet someone that has always interpreted p-values correctly.

We have of course met people who think they understand p-values and most of them are quite convinced of that. In our personal experience and observation, even statisticians giving a talk on p-values as well as statistical textbooks routinely misuse the term. No wonder confusion reigns.

A common misuse of a p-value is formulating a stronger statement than what it was created for. Perhaps the most common misuse of a p-value is expressed like so: *"our small p-value shows that the result was not produced by random chance."* This is not what a p-value indicates.

Another very common abuse of a p-value is to consider a smaller p-value to be a stronger indication that an effect is right. Sorting by p-value should be done only to apply a cutoff more quickly and not to put "better" results first.

To avoid confusing yourself and others here is what we recommend on p-values:

> 1. Think of the p-value as the probability of obtaining an effect of the size that you observe due to random chance. When the p-value is small you may say that the probability of getting an effect of this magnitude by chance is low.
> 2. Think of p-values as selection cutoffs. Use them to select values that you choose to trust. Do NOT use p-values as an indicator for more reliable or less reliable results nor as indicators of stronger or weaker effects.

The problems caused by misusing p-values are well documented, refer to the following brief summary:

- Statisticians issue warning over misuse of P values

# How to choose the "right" RNA-Seq analysis?

Note: *This chapter is slated for significant expansion, we were not quite able to finish it for launch. We'll be working on it next.*

## Which statistical methodology should I choose?

First a terminology:

- True Positives (TP): correctly identified
- False Positives (FP): incorrectly identified
- True Negatives (TN): correctly rejected
- False Negatives (FN): incorrectly rejected

from these, other values can be derived such as:

- Sensitivity = True Positive rate = TP/(TP + FN)
- Specificity = True Negative rate = TN/(TN + FP)
- False discovery rate = FP/(FP+TP)

See more definitions at https://en.wikipedia.org/wiki/Sensitivity_and_specificity. Ideally we want methods with high TP and TN and low FP and FN rates.

In our observations, `deseq1` makes fewer mistakes but also finds fewer of the differentially expressed genes. That means it has a low FP and higher FN.

In contrast the `deseq2` and `edgeR` methods find more of the differentially expressed genes but at the cost of also producing more false values. They have higher TP and but also higher FP.

You have to decide -- what do you want to minimize, FP or FN? Often the data gives you little choice. `deseq1` is less "robust" (in a statistical sense)- meaning it is more sensitive to outliers in the data and in some experiments may not identify any differentially expressed genes. On the other hand `deseq2` and `edger` might be way too generous and smother you with false positives.

We like to start with `deseq1`, even though the tool itself, after running it suggests using `deseq2`. It gives you a sense of how consistent the data truly is.

Here are a few more pointers to research works:

- Power analysis and sample size estimation for RNA-Seq differential expression
- Modeling of RNA-seq fragment sequence bias reduces systematic errors in transcript abundance estimation

# RNA-Seq: Griffith Test Data

This examples uses the data from the following publication:

- Informatics for RNA-seq: A web resource for analysis on the cloud. 11(8):e1004393. PLoS Computational Biology (2015) by Malachi Griffith, Jason R. Walker, Nicholas C. Spies, Benjamin J. Ainscough, Obi L. Griffith.

A detailed tutorial is available online at https://github.com/griffithlab/rnaseq_tutorial/wiki

We have greatly simplified the data naming and organization. We recommend the original resource as an alternative tutorial and source of information.

# How do I automate the Griffith data download?

We have placed the commands needed to obtain the data in the script called `griffith-getdata.sh` that you can download and run yourself.

First switch to a newly made folder then download and run the script:

```
# Create a new folder to work in.
mkdir -p griffith
cd griffith

# Download the script.
curl -O http://data.biostarhandbook.com/rnaseq/projects/griffith/griffith-getdata.sh

# Run the script.
bash griffith-getdata.sh
```

Once the process completes you will have three directories:

- `reads` containing the sequencing reads.
- `refs` containing the prebuilt indices and gene information.

The data is ready for analysis. Read on to understand what the data is like and what this script does.

# Does this data have "spike-in" control?

Yes there are two mixes: ERCC Mix 1 and ERCC Mix2. The spike-in consists of 92 transcripts that are present in known concentrations across a wide abundance range (from very few copies to many copies).

# What type of data is included?

The data consists of two commercially available RNA samples:

- Universal Human Reference (UHR) is total RNA isolated from a diverse set of 10 cancer cell lines.

- Human Brain Reference (HBR) is total RNA isolated from the brains of 23 Caucasians, male and female, of varying age but mostly 60-80 years old.

The data was produced in three replicates for each condition.

So to summarize this data consists of the following:

1. UHR + ERCC Mix1, Replicate 1, `HBR_1`
2. UHR + ERCC Mix1, Replicate 2, `HBR_2`
3. UHR + ERCC Mix1, Replicate 3, `HBR_3`
4. HBR + ERCC Mix2, Replicate 1, `UHR_1`
5. HBR + ERCC Mix2, Replicate 2, `UHR_2`
6. HBR + ERCC Mix2, Replicate 3, `UHR_3`

Since this is a paired end dataset with 6 samples we will have 12 files in total. Let's get the data (145MB):

```
# This is the URL for the data
URL=http://data.biostarhandbook.com/rnaseq/projects/griffith/griffith-data.tar.gz

# Download and unpack the data
curl $URL | tar zxv
```

To display the data do

```
ls -1 reads
```

That produces:

```
HBR_1_R1.fq
HBR_1_R2.fq
HBR_2_R1.fq
HBR_2_R2.fq
HBR_3_R1.fq
HBR_3_R2.fq
UHR_1_R1.fq
UHR_1_R2.fq
UHR_2_R1.fq
UHR_2_R2.fq
UHR_3_R1.fq
UHR_3_R2.fq
```

The folder `refs` contains the annotations for the ERCC mixtures:

```
ls -1 refs
```

For this data we will use the human chromosome 22 as reference to make the examples go faster. Do note how it is impossible to tell from the data naming and content alone which genomic build they correspond to.

```
22.fa
22.gtf
ERCC92.fa
ERCC92.gtf
...
```

# RNA-Seq: Analyzing control samples

In this section we will analyze only the "spike control" data of the research paper described before where the authors added so-called "spike-in" controls to each of their experimental conditions.

The three stages of RNA-Seq will be performed with:

1. Alignment: `hisat2`
2. Quantification: `featureCounts`
3. Differential expression: `DESeq`

# How do I automate the Griffith data processing?

We have placed the commands for the stages of RNA-Seq analysis into three separate scripts:

1. `griffith-getdata.sh` downloads and prepares the data.
2. `griffith-align.sh` aligns the data.
3. `griffith-count.sh` quantifies the data.

Download the scripts like so:

```
curl -O http://data.biostarhandbook.com/rnaseq/projects/griffith/griffith-getdata.sh
curl -O http://data.biostarhandbook.com/rnaseq/projects/griffith/griffith-align.sh
curl -O http://data.biostarhandbook.com/rnaseq/projects/griffith/griffith-count.sh
```

then run each like:

```
# Get the data.
bash griffith-getdata.sh

# Align the data.
bash griffith-align.sh

# Estimate abundances.
bash griffith-count.sh
```

If your system is set up correctly these commands will run and finish processing the control samples within a few minutes.

Read on to find out what takes place in these scripts.

# What is a spike-in control?

The goal of the spike-in is to determine just how well can we measure and reproduce data with known (expected) properties. A common product called the ERCC ExFold RNA Spike-In Control Mix can be added in different mixtures.

This spike-in consists of 92 transcripts that are present in known concentrations across a wide abundance range (from very few copies to many copies). Here is an example:

```
Name           Mix1     Mix2       M1/M2     log2(M1/M2)
ERCC-00130    30000     7500         4          2
ERCC-00092      234       58         4          2
ERCC-00083    0.028    0.007         4          2
ERCC-00096    15000    15000         1          0
ERCC-00060      234      234         1          0
ERCC-00117    0.057    0.057         1          0
ERCC-00002    15000    30000       0.5         -1
ERCC-00079       58      117       0.5         -1
ERCC-00061    0.057    0.114       0.5         -1
```

If we were to add Mix 1 as condition 1 and Mix 2 as condition 2 then we would expect that our RNA-Seq experiment recovers the fold changes listed above.

| A | B | C | D | E | F |
|---|---|---|---|---|---|
| **ERCC ID** | **subgroup** | **concentration in Mix 1 (** | **concentration in Mix 2 (a** | **expected fold-change ratio** | **log2(Mix 1/Mix 2)** |
| ERCC-00130 | A | 30000.00 | 7500.00 | 4 | 2 |
| ERCC-00004 | A | 7500.00 | 1875.00 | 4 | 2 |
| ERCC-00136 | A | 1875.00 | 468.75 | 4 | 2 |
| ERCC-00108 | A | 937.50 | 234.38 | 4 | 2 |
| ERCC-00116 | A | 468.75 | 117.19 | 4 | 2 |
| ERCC-00092 | A | 234.38 | 58.59 | 4 | 2 |
| ERCC-00095 | A | 117.19 | 29.30 | 4 | 2 |
| ERCC-00131 | A | 117.19 | 29.30 | 4 | 2 |
| ERCC-00062 | A | 58.59 | 14.65 | 4 | 2 |
| ERCC-00019 | A | 29.30 | 7.32 | 4 | 2 |
| ERCC-00144 | A | 29.30 | 7.32 | 4 | 2 |
| ERCC-00170 | A | 14.65 | 3.66 | 4 | 2 |
| ERCC-00154 | A | 7.32 | 1.83 | 4 | 2 |

Note how in theory a 4-fold change could be observed for both highly expressed genes `30000/7500` or low expression genes `0.028/0.07`. We also know that the latter will produce much fewer measurements, hence we expect that the method will deteriorate for transcripts of low expression levels. The question is at what concentration we lose the ability to reliably detect a fold change.

Download and view the expected differential expressions from ERCC-datasheet.csv

# How do I align an RNA-seq sample?

We select a splice aware aligner, in this case `hisat2` and build an index from our control sequences.

```
# This is the reference genome.
REF=refs/ERCC92.fa

# Name the prefix for the index the same as the REF.
IDX=refs/ERCC92.fa

# Build a hisat2 index for the genome.
hisat2-build $REF $IDX
```

For every script our goal is to create it using "reusable" components. This means that we want to create a script that is easy to adapt to another dataset. In this case we want to set up our script in a way that we can run it on the spiked data and later on the real RNA-Seq dataset as well. To do that we need to consolidate the changeable elements of our script into variables that we can set. Let's also set up a few more shortcuts to simplify our commands:

```
# The sequencing data that contain the first in pair
R1=reads/UHR_1_R1.fq

# The sequencing data that contain the second in pair
R2=reads/UHR_1_R2.fq

# The name under which we store the BAM file.
BAM=bam/UHR_1.bam
```

To run our spliced aligner we invoke it as:

```
# Make sure we have this folder.
mkdir -p bam

# Align then sort and convert to BAM file.
hisat2 $IDX -1 $R1 -2 $R2 | samtools sort > $BAM

# Index the bam file.
samtools index $BAM
```

Note how generic and re-usable this is:

```
hisat2 $IDX -1 $R1 -2 $R2 | samtools sort > $BAM
```

This command will run on any index and read pair and we avoided the so-called "hard-coding" of information into the sections of the script that do the work. We can replace the read names, the bam file, yet the command stays the same. This is an approach that you should always strive for.

# How do I automate my code for all samples?

The simplest possible solution could be to just list every command separately and put that information in a file that can be executed like so:

```
# The name of the index
IDX=refs/ERCC92.fa

# Make the directory if it does not exist.
mkdir -p bam

R1=reads/UHR_1_R1.fq
R2=reads/UHR_1_R2.fq
BAM=bam/UHR_1.bam
hisat2 $IDX -1 $R1 -2 $R2 | samtools sort > $BAM

R1=reads/UHR_2_R1.fq
R2=reads/UHR_2_R2.fq
BAM=bam/UHR_2.bam
hisat2 $IDX -1 $R1 -2 $R2 | samtools sort > $BAM

R1=reads/UHR_3_R1.fq
R2=reads/UHR_2_R2.fq
BAM=bam/UHR_3.bam
hisat2 $IDX -1 $R1 -2 $R2 | samtools sort > $BAM
...
```

and so on. It is redundant but it is very explicit in what it does. The command itself does not change, only what it gets executed on.

The main danger is that you forget to change one or more of the file names and you end up using the wrong data. Did you notice the error that we made above? Look again!

Since you've stored it in a file, you can inspect it, double-check for errors, and re-run it any time. While it is a simplistic approach, it is an excellent start to creating a reusable workflow. You don't have to write a more complex script unless you want to.

# How to create a script to run all datasets?

Those with a desire to write non-redundant programs will frown at the solution above. It is not overly complicated to write loops that generate the desired file names by concatenating sample names and replicate numbers. A script that achieves that could look like:

```
# Create output folder
mkdir -p bam

# Exit this script on any error.
set -euo pipefail

# This the name of the index, set to control.
IDX=refs/ERCC92.fa

for SAMPLE in HBR UHR;
do
    for REPLICATE in 1 2 3;
    do
        # Build the name of the files.
        R1=reads/${SAMPLE}_${REPLICATE}_R1.fq
        R2=reads/${SAMPLE}_${REPLICATE}_R2.fq
        BAM=bam/${SAMPLE}_${REPLICATE}.bam

        # Run the aligner.
        hisat2 $IDX -1 $R1 -2 $R2 | samtools sort > $BAM
        samtools index $BAM
    done
done
```

Place the above into a script `align.sh` and run it with

```
bash align.sh
```

The value of the script above is that only the reference and index need to be changed and the same script can be used to align the same data against the human genome. In addition it would be easy to change the sample names or replicate numbers and your script would still work.

With that we have just created a pipeline that churns through all 12 files in a repeatable manner. We can modify the alignment command if we wish so and re-run everything with ease. The result will be 6 bam files in the `bam` folder named `UHR_1.bam`, `UHR_2.bam`, `UHR_3.bam` and `HBR_1.bam`, `HBR_2.bam`, `HBR_3.bam`.

# How do I estimate the abundance for a single sample?

We evaluate the abundance by counting the number of alignments that fall over a certain interval. Over the years a great many tools have been proposed, but our all time favorite is `featureCounts`, a program that takes as input a gene feature file and one (or many) bam files.

```
# These are the coordinates of the genes.
GTF=refs/ERCC92.gtf

featureCounts -a $GTF -o counts.txt  bam/HBR_1.bam
```

By default the `featureCounts` program uses the `gene_id` attribute in the GTF file. We can override that and instruct `featureCounts` to use the `gene_name` attribute instead with:

```
featureCounts -a $GTF -g gene_name -o counts.txt  bam/HBR_1.bam
```

The resulting file `counts.txt` is a tab delimited file where the first six columns contain feature specific information and the rest of the columns contain the readcounts that overlap with that feature.

```
Geneid            Chr         Start     End  Strand   Length  bam/HBR_1.bam
ERCC-00002      ERCC-00002      1       1061    +       1061    37892
ERCC-00003      ERCC-00003      1       1023    +       1023    2904
ERCC-00004      ERCC-00004      1       523     +       523     910
ERCC-00009      ERCC-00009      1       984     +       984     638
```

To find the sequences with most hits we can sort by column 7:

```
cat counts.txt | sort -rn -k 7 | head
```

This produces:

```
ERCC-00002   ERCC-00002    1   1061   +   1061    37892
ERCC-00074   ERCC-00074    1    522   +    522    20982
ERCC-00096   ERCC-00096    1   1107   +   1107    20708
ERCC-00130   ERCC-00130    1   1059   +   1059    8088
ERCC-00113   ERCC-00113    1    840   +    840    7096
```

We can see how the counts in the last column correlate very closely with the expected abundances for Mix2 in the ECC Mix file.

# Are there different ways to count overlaps?

Yes there are.

As with many other tools counting how many reads overlap with a feature is also fraught with subjectivity. We may or may not include reads that overlap only partially or reads that align to more than one feature. We may also choose to count both pairs as one count or as two counts and so on. In some cases these choices don't make any difference, in others they do.

# How do I estimate abundances for all samples?

We can list more than one bam file for `featureCounts` . For example using the shell metacharacter '*' we can list all HBR and all UHR samples in a compact form:

```
featureCounts -a $GTF -g gene_name -o counts.txt  bam/HBR*.bam  bam/UHR*.bam
```

In this case the `counts.txt` file will contain a column for each of the samples, for a total of 13 columns. Note how each sample will have the readcounts listed in each row that corresponds to each feature of the file.

```
  Geneid        Chr      Start   End  Str  Len   HBR_1    HBR_2   HBR_3   UHR_1   UHR_2   UHR_3
ERCC-00002   ERCC-00002   1    1061   +   1061   37892   47258   42234   39986   25978   33998
ERCC-00003   ERCC-00003   1    1023   +   1023   2904     3170    3038    3488    2202    2680
ERCC-00004   ERCC-00004   1     523   +    523   910      1078    996     9200    6678    7396
...
```

All that is left is to compare the replicates for `HBR` to the replicates in `UHR` to find those features for which the replicates within one condition are consistently different from those in the other condition.

# How do I find differential expression?

We have set up several R scripts that can produce differential expression computation for you using count information similar to the one that the `featureCounts` program produces. Get our R script as described on RNA-Seq with Bioconductor page:

```
curl -O http://data.biostarhandbook.com/rnaseq/code/deseq1.r
```

The script expects a file that contains only gene names and counts. Basically we need to remove the intermediate columns.

```
cat counts.txt | cut -f 1,7-12 > simple_counts.txt
```

So that our file now is just:

```
ERCC-00002      37892   47258   42234   39986   25978   33998
ERCC-00003      2904    3170    3038    3488    2202    2680
ERCC-00004      910     1078    996     9200    6678    7396
ERCC-00009      638     778     708     1384    954     1108
...
```

Then pass this file through the script by specifying the design of the experiment in this case 3 replicates for each of the two conditions so `3x3`.

```
cat simple_counts.txt | Rscript deseq1.r 3x3 > results.txt
```

The `Rscript` command launches `R` with our `deseq1.r` script that takes its input from the left and writes into `results.txt`.

# What does a differential expression file look like?

A differential expression file describes the changes in gene expression across two conditions. It will be similar to:

```
id            baseMean   baseMeanA     baseMeanB   foldChange  log2FoldChange    pval       p
adj
ERCC-00130      29681        10455        48907        4.67        2.22         1.16e-88    9.
10e-87
ERCC-00108        808          264         1352        5.10        2.35         2.40e-62    9.
39e-61
ERCC-00136       1898          615         3180        5.16        2.36         2.80e-58    7.
30e-57
```

Typically you get a column (though the naming will depend on the tool you use) for each of the following:

- `id` : Gene or transcript name that the differential expression is computed for,
- `baseMean` : The average normalized value across all samples,
- `baseMeanA` , `baseMeanB` : The average normalized gene expression for each condition,
- `foldChange` : The ratio `baseMeanB/baseMeanA` ,
- `log2FoldChange` : log2 transform of `foldChange` . When we apply a 2-based logarithm the values become symmetrical around 0. A log2 fold change of 1 means a doubling of the expression level, a log2 fold change of -1 shows show a halving of the expression level.
- `pval` : The probability that this effect is observed by chance,
- `padj` : The adjusted probability that this effect is observed by chance.

You may use `pval` if you already selected your target gene before evaluating these results.

You have to use `padj` in all other cases as this adjusted value corrects for the so-called multiple testing errors - basically accounts for the many alternatives and their chances of influencing the results that we see.

# Did our RNA-Seq analysis reproduce the expected outcomes?

Let's look at the first few rows. `ERCC-00130` was generated at the highest concentration (highest gene expression) at a fold change of `4` . We obtain a fold change of `4.67` so we overshot it, though not by much.

`ERCC-00108` was also deposited at a fold change of `4` but since it has 30 times smaller concentration we expect less accuracy. Indeed we observe a fold change of 5.10 for it. We can line up the expected table with the obtained one like so:

```
# We need to sort data by ID so that can be pasted in columns.
curl http://data.biostarhandbook.com/rnaseq/ERCC/ERCC-datasheet.csv | grep ERCC-00 | cut -f 1,
5,6 -d , | sort > table1
cat results.txt | grep ERCC- | sort | cut -f 1,5,6 > table2

# Joining the tablesthat contains both datasets for comparison
paste table1 table2 > compare.txt
```

The results are summarized in an Excel table that you too can download from ERCC-results.csv

In summary, for a fold change of 4 or higher, transcripts expressing with as much as 1000x fold difference were reliably detected. In all the RNA-Seq method worked very well.

| ERCC ID | Mix 1 | Mix 2 | Expected Fold Change | Measured Fold Change | Error |
|---|---|---|---|---|---|
| ERCC-00130 | 30000 | 7500 | 4 | 4.7 | 0.7 |
| ERCC-00004 | 7500 | 1875 | 4 | 5.9 | 1.9 |
| ERCC-00136 | 1875 | 468.75 | 4 | 5.2 | 1.2 |
| ERCC-00108 | 937.5 | 234.375 | 4 | 5.1 | 1.1 |
| ERCC-00116 | 468.75 | 117.1875 | 4 | 4.6 | 0.6 |
| ERCC-00092 | 234.375 | 58.59375 | 4 | 5.4 | 1.4 |
| ERCC-00095 | 117.1875 | 29.296875 | 4 | 4.4 | 0.4 |
| ERCC-00131 | 117.1875 | 29.296875 | 4 | 3.8 | -0.2 |
| ERCC-00062 | 58.59375 | 14.6484375 | 4 | 5.8 | 1.8 |
| ERCC-00019 | 29.296875 | 7.32421875 | 4 | 2.8 | -1.2 |
| ERCC-00144 | 29.296875 | 7.32421875 | 4 | 2.4 | -1.6 |
| ERCC-00170 | 14.6484375 | 3.66210938 | 4 | 6.2 | 2.2 |
| ERCC-00085 | 7.32421875 | 1.83105469 | 4 | 4.1 | 0.1 |
| ERCC-00154 | 7.32421875 | 1.83105469 | 4 | 4.9 | 0.9 |
| ERCC-00028 | 3.66210938 | 0.91552734 | 4 | 6.3 | 2.3 |
| ERCC-00033 | 1.83105469 | 0.45776367 | 4 | Inf | NaN |
| ERCC-00134 | 1.83105469 | 0.45776367 | 4 | 0 | -4 |
| ERCC-00147 | 0.91552734 | 0.22888184 | 4 | NaN | NaN |
| ERCC-00097 | 0.45776367 | 0.11444092 | 4 | Inf | NaN |
| ERCC-00156 | 0.45776367 | 0.11444092 | 4 | NaN | NaN |

Note how for data with insufficient measures we get infinite or NaN (Not a Number) values indicating a division by zero or other numeric overflow.

# Are there other differential expression computation tools?

There are quite a few. See the RNA-Seq: Statistical analysis page for more details.

# RNA-Seq with alignment based methods on the Human Brain Data

We assume that the data was installed according to RNA-Seq: Griffith Data and that your terminal is opened into the directory that contains the data. In addition we assume that you have at least partially completed the Analyzing control samples section an understand the origins of the script that we start with.

## How do I automate the Griffith data processing?

We have placed the commands for the stages of RNA-Seq analysis into three separate scripts:

1. `griffith-getdata.sh` downloads and prepares the data.
2. `griffith-align.sh` aligns the data.
3. `griffith-count.sh` quantifies the data.

Download and run the scripts like so:

```
curl -O http://data.biostarhandbook.com/rnaseq/projects/griffith/griffith-getdata.sh
curl -O http://data.biostarhandbook.com/rnaseq/projects/griffith/griffith-align.sh
curl -O http://data.biostarhandbook.com/rnaseq/projects/griffith/griffith-count.sh
```

While these scripts were designed for the control samples you can re-use each once you make a small change in them.

## How do I adapt scripts to new data?

As we described in Analyzing control samples the most important feature of any automation is to isolate the variable parts of the analysis from the repetitive segments. You then move the variable parts of the analysis to the beginning of the script where you can easily and reliably adapt the script to new data.

Note how easy it is to adapt a script that operates on control data to the realistic data. All we needed to do is edit these two lines to read:

```
# This the new name of the index
IDX=refs/22.fa

# These are the new genomic genes.
GTF=refs/22.gtf
```

You can actually make that replacement at the command line with `sed` while editing in place `-i ''`:

```
sed -i '' 's/ERCC92.fa/22.fa/' griffith-align.sh
sed -i '' 's/ERCC92.gtf/22.gtf/' griffith-count.sh
```

And you can rerun the automated scripts with:

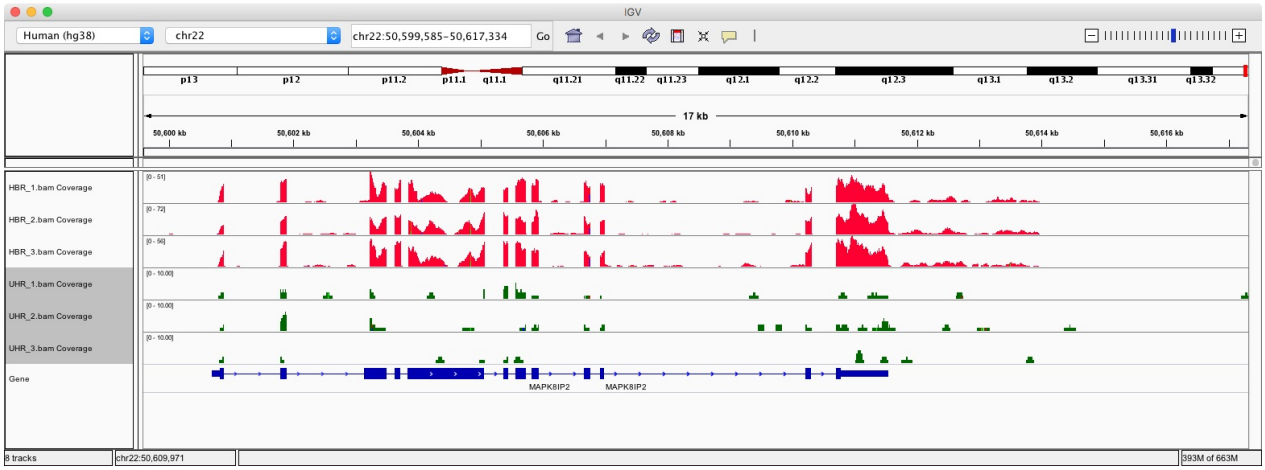
```
bash griffith-align.sh
bash griffith-count.sh
```

# How do I generate the list of differentially expressed features?

The exact same steps need to be followed as in the previous example. You need to edit the `griffith-align.sh` and the `griffith-count.sh` scripts and change what target they operate on.

```
featureCounts -a $GTF -g gene_name -o counts.txt  bam/U*.bam  bam/H*.bam
```

Visualizing chromosome 22 in IGV we can see the following for gene `MAPK8IP2` located `chr22:50600685-50613981` (type the gene name `MAPK8IP2` into the IGV coordinate box):




From the image it is clear that this gene has systematically more data in the `HBR` samples than the `UHR` samples. The next sections explain how we can find these regions automatically.

# What does the differential expression file look like?

A differential expression file describes the changes in gene expression across two conditions. It will be similar to:

```
id        baseMean   baseMeanA   baseMeanB   foldChange   log2FoldChange   pval        padj
MAPK8IP2   561.83     22.66       1101.01      48.58        5.60             0           0
SYNGR1    1050.43     82.72       2018.13      24.39        4.60             0           0
SEPT3      984.47     81.19       1887.75      23.24        4.53             0           0
SEZ6L      542.46     30.99       1053.93      33.99        5.08             7.7e-285    1.8e-282
YWHAH     1594.19    470.05       2718.34       5.78        2.53             1.0e-253    1.9e-251
SULT4A1    355.27      8.42        702.12      83.33        6.38             3.1e-231    5.0e-229
...
```

- Gene or transcript name that the differential expression is computed for `id`.

- The average normalized value across all samples `baseMean`
- The average normalized gene expression for each condition `baseMeanA`, `baseMeanB`.
- The ratio of values called `foldChange` computed as `baseMeanB/baseMeanA`.
- A log2 transform of `foldChange` called `log2FoldChange`. When we apply a 2 based logarithm then the values become symmetrical around 0. A log2 fold change shown of 1 and -1 will show a doubling or halving of the expression levels.
- A probability that this effect is observed by chance: `pval`.
- An adjusted probability that this effect is observed by chance: `padj`.

You may use `pval` if you already had selected your target gene before evaluating these results.

You have to use `padj` in all other cases as this adjusted value corrects for the so called multiple testing errors - basically accounts for the many alternatives and their chances of influencing the results that we see.

The genes `MAPK8IP2`, `SYNGR1`, `SEPT3` and 293 more appear to be expressing at different levels across our experimental conditions.

# How do I interpret the results?

The `results.txt` contains the genes sorted by their adjusted p-values (last column). Imposing a filter to this column, for example selecting genes that have changed at `0.05` level:

```
cat results.txt | awk ' $8 < 0.05 { print $0 }' > diffgenes.txt

# How many differentially expressed genes do we have?
cat diffgenes.txt | wc -l
# 293
```

At this step you we can perform Gene Enrichment studies.

# RNA-Seq by "mapping" (pseudoalignments)

The pseaudoalignments is a concept introduced to the world by the work of Rob Patro and Carl Kingsford at the Lane Center for Computational Biology at Carnegie Mellon University. In a nutshell, pseudoalignment-based methods provide a "mapping" that identifies locations based on the patterns in the DNA rather than via alignment type algorithms. The advantage of pattern matching is that it operates many orders of magnitude faster than alignment and makes it possible to analyze very large datasets even on a laptop.

After the introduction of the concept the method was extended to RNA-Seq data analysis. Two implementations of the method exist, Kallisto and Salmon, with others probably in development.

## What are Kallisto and Salmon?

Kallisto and Salmon are software packages from different authors for quantifying transcript abundances. The tools perform a *pseudoalignment* of reads against a transcriptome. In pseudoalignment, the program tries to identify for each read the target that it originates from but not where in the the target it aligns. This makes the algorithm much faster than a 'real' alignment algorithm.

## How do I automate the Kallisto run?

We have placed the commands for the stages of RNA-Seq analysis into three separate scripts:

1. `griffith-getdata.sh` downloads and prepares the data.
2. `griffith-kallisto.sh` downloads and prepares the data and performs an RNA-Seq estimation for both control and human transcriptome samples.

Download and run the scripts like so:

```
curl -O http://data.biostarhandbook.com/rnaseq/projects/griffith/griffith-getdata.sh
curl -O http://data.biostarhandbook.com/rnaseq/projects/griffith/griffith-kallisto.sh
```

then run them with:

```
bash griffith-getdata.sh
bash griffith-kallisto.sh
```

Read on to understand what these scripts do. We describe the process from the point of view of the Kallisto tool but Salmon works almost identically.

## How do I build a Kallisto index?

Kallisto uses a transcriptome as reference. We can build a Kallisto index for ERCC control sequence as shown below.

```
# This is the reference.
REF=refs/ERCC92.fa

# This the name of the index
IDX=refs/ERCC92.idx

# Build kallisto index
kallisto index -i $IDX  $REF
```

# How do I quantify transcripts with Kallisto?

Kallisto assigns reads to transcripts and calculates their abundance in one step. The algorithm can be invoked using `quant` command.

To quantify the paired end data in our example

```
# Set shortcuts and directories
mkdir -p kallisto

R1=reads/HBR_1_R1.fq
R2=reads/HBR_1_R2.fq

# Kallisto will generate multiple outputs.
OUTDIR=out

# Run kallisto quantification.
kallisto quant -i $IDX -o $OUTDIR -b 100 $R1 $R2
```

The above command produces output files in the output directory called `out` specified by `-o` option. `-b` option specifies the number of bootstrap samples. Setting this parameter is useful if one is using a program called `Sleuth` for downstream differential expression analysis.

# How do I quantify all samples with Kallisto?

Because Kallisto creates a directory with multiple files in it a little more bookkeeping is required to keep everything tidy.

The script below creates a Kallisto index and then quantifies all samples. The Kallisto outputs are placed in the 'kallisto' directory. Also, the script produces `sample_counts.txt` by concatenating the 'est_counts' column from `abundance.txt` file of all samples.

```
# Create output folder
mkdir -p results

# Exit this script on any error.
set -euo pipefail

# This is the reference genome.
REF=refs/ERCC92.fa

# This the name of the index
IDX=refs/ERCC92.idx

# Build kallisto index
kallisto index -i $IDX  $REF

for SAMPLE in HBR UHR;
do
    for REPLICATE in 1 2 3;
    do
        # Build the name of the files.
        R1=reads/${SAMPLE}_${REPLICATE}_R1.fq
        R2=reads/${SAMPLE}_${REPLICATE}_R2.fq

        # The kallisto output directory.
        OUTDIR=results/${SAMPLE}_${REPLICATE}

        # Run kallisto quantification.
        kallisto quant -i $IDX -o $OUTDIR -b 100 $R1 $R2

        # Copy the abundance file to a proper name.
        cp $OUTDIR/abundance.tsv $OUTDIR.counts.tsv
    done
done
```

This generates a directoy `results` that contains a separate directory for each sample. As we noted elsewhere, this is an improper way to create data as critical information is being stored in the path names. A file named `results/HBR_1/abundance.tsv` contains information on sample `HBR_1` but note how you could not tell that from the file name alone. It is a bad (yet common) practice that we dearly wish would go away. So we copy the abundance file into an unambiguous name `HBR_1.tsv`

To concatenate all counts to create a count file

```
paste results/H*.tsv  results/U*.tsv | cut -f 1,4,9,14,19,24,29  > counts.txt
```

The counts file `counts.txt` will be as shown below. This can be used as input to downstream DEseq differential expression analysis.

```
target_id   est_counts est_counts est_counts est_counts est_counts est_counts
ERCC-00002  18946      23629      21117      19993      12989      16999
ERCC-00003  1452       1585       1519       1744       1101       1340
ERCC-00004  455        539        498        4600       3339       3698
ERCC-00009  319        389        354        692        477        554
```

You may change the header to include the sample names.

# What are the outputs of Kallisto?

All Kallisto output files will be placed in the output directory specified by the `-o` option. Transcript abundance estimate results are in a tab-delimited file named `abundance.txt`. Transcript abundance contains estimated counts and the TMP measure.

```
target_id   length eff_length est_counts tpm
ERCC-00002  1061   891.059      18946      243099
ERCC-00003  1023   853.059      1452       19460.7
ERCC-00004  523       353.059      455        14734.5
ERCC-00009  984       814.059      319         4480.29
```

Effective length (3rd column) scales the transcript length by fragment length distribution. It takes into account the part of the transcript where fragments would overlap fully in any location. It is usually calculated as `transcript length - mean(fragment length) +1`. Abundances are reported in “estimated counts” (est_counts) and in Transcripts Per Million (TPM).

Other outputs include `run_info.json` file which contains a summary of the run and bootstrp results in `abundance.h5` HDF5 format file.

# RNA-Seq with the Tuxedo suite

Whereas there is a lot of excitement about pseudoalignment methods, which are the present and future, it is worth looking back to the past.

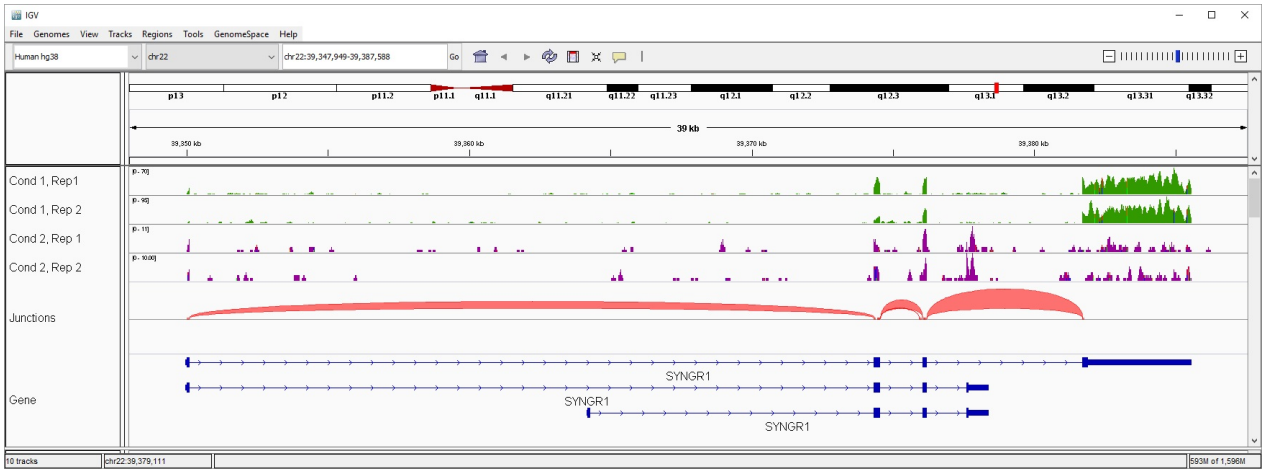
> Please note that we do not recommend that you use TopHat for your analysis. More efficient and more sensitive alternative tools now exist.

Or as the authors of TopHat tool state:

> Please note that TopHat has entered a low maintenance, low support stage as it is now largely superseded by HISAT2 which provides the same core functionality (i.e. spliced alignment of RNA-Seq reads), in a more accurate and much more efficient way.

The Tuxedo "suite": `bowtie` , `tophat` , `cufflinks` was the first automated RNA-Seq pipeline that was able to seamlessly analyze the data. Its usability, simplicity and performance was unmatched by its alternatives. It maintained this lead for many years and, for many, it has become a synonym for RNA-Seq analysis. By the current (2016) standards Tophat is less accurate, less efficient, 10x-100x slower than other options and there has been a decline in its use.

The Tuxedo suite made it the easiest to see the changes in the RNA-Seq expression with minimal fuss.




For the forseeable future the Tuxedo suite will most likely remain the RNA-seq tool that produced the largest number of publications. This software package is a monument of ingenuity, designed and created by a single individual with outstanding programming skill.

With the Tuxedo suite the three stages of RNA-Seq are performed with:

1. Alignment: `tophat` (behind the scenes uses the `bowtie2` aligner)
2. Quantification and Differential Expression: `cufflinks`

Note: TopHat may fail to run correctly on macOS and cufdiff is known to exhibit unexpected errors on this platform. You may not be able to run this example to completion.

In our case replicate `UHR_2` fails to run for us on a macOS but completes on Linux. In a way this shows that `tophat` is getting a little long in the tooth and clearly has internal coding errors, hence should probably be replaced with other techniques.

# Is there a newer Tuxedo suite?

Yes, the new Tuxedo suite includes `hisat2`, `stringtie` and `ballgown` triplet where

1. Alignment: `hisat2`
2. Quantification: `stringtie`
3. Differential Expression: `ballgown` (an R package)

Unfortunately with the "new and improved" version of the Tuxedo suite the old charm and TopHat has been lost. TopHat was always about simplicity, it was a self-contained program that required no other tool or method to run, and immediately provided the information you sought. The "new" method is more complicated and difficult to follow along.

# How do I run the "original" (OLD) Tuxedo suite?

We go back to our script that we wrote when analyzing control samples; we have already re-used most of this script when processing the true mRNA samples in the second analysis. We can again use the same exact script for running a different analysis, this time with the Tuxedo suite.

```
# This is the reference genome.
REF=annot/22.fa

# Name the prefix for the index the same as the REF.
IDX=annot/22.fa

# The sequencing data that contain the first in pair
R1=reads/UHR_1_R1.fq

# The sequencing data that contain the second in pair
R2=reads/UHR_1_R2.fq

# These are the coordinates of the genes.
GTF=annot/22.gtf

# Build a bowtie2 index for the genome.
bowtie2-build $REF $IDX
```

The only line we need to change is the invocation of the aligner itself. It changes to

```
tophat -G $GTF -o tophat_hbr1 $IDX $R1 $R2
```

TopHat produces more than just an alignment file hence it needs an output directory named as `tophat_hbr1` above to store these files. The file that has the alignment is called `accepted_hits.bam`.

It is very annoying and makes analyses error prone that the result of every TopHat run is an identically named file: `accepted_hits.bam`.

It is a well-known bad practice to distinguish files by their location alone. That is really what happens here, sadly other tools seem to embrace and adopt this practice. We recommend that you rename this file right away and give it a name that matches your samples.

We modify our script as follows (note again how little did we have to change) relative to the original:

```
# Create output folder
mkdir -p bam

# Exit this script on any error.
set -euo pipefail

# This is the reference genome.
REF=annot/22.fa

# This the name of the index
IDX=annot/22.fa

# These are the coordinates of the genes.
GTF=annot/22.gtf

for SAMPLE in HBR UHR;
do
    for REPLICATE in 1 2 3;
    do
        # Build the name of the files.
        R1=reads/${SAMPLE}_${REPLICATE}_R1.fq
        R2=reads/${SAMPLE}_${REPLICATE}_R2.fq
        BAM=bam/${SAMPLE}_${REPLICATE}.bam

        # Run the tophat aligner.
        tophat2 -G $GTF -o ${SAMPLE}_${REPLICATE} $IDX $R1 $R2

        # Rename the tophat alignment file.
        mv ${SAMPLE}_${REPLICATE}/accepted_hits.bam $BAM

        # Index the BAM file.
        samtools index $BAM
    done
done
```

Place the content above into a script called `tophat.sh` and run it:

```
bash tophat.sh
```

TopHat will run noticeably slower than the newest alternatives. It can be up to 20x to 100x slower than other tools. The result of the run will be as many directories as you have samples, and each directory will contain a number of files.

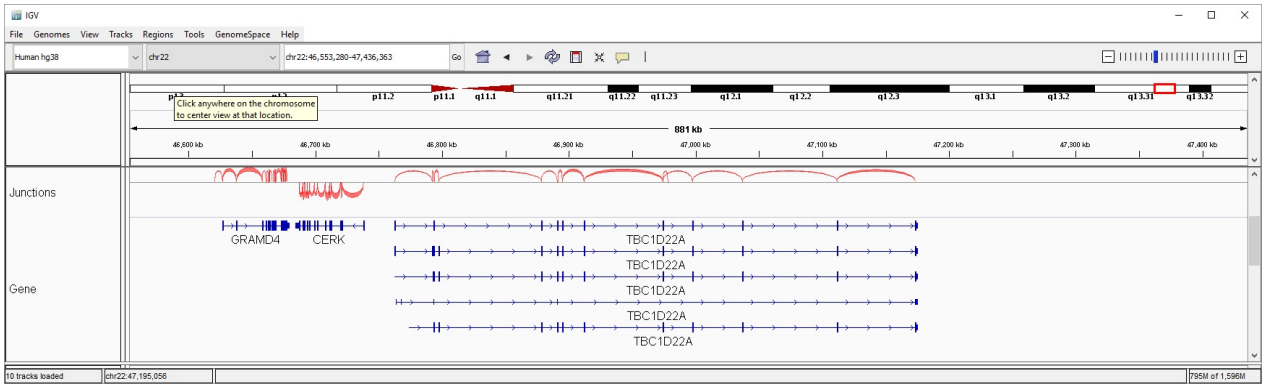
# What other information does TopHat generate?

There are several interesting files in the tophat_UHR_1 directory:

- junctions.bed
- insertions.bed
- deletions.bed

Whereas the `insertions.bed` , and `deletions.bed` are meant to detect variations in the transcriptome, the information lacks the accuracy of a proper variant calling algorithm. The file called `junctions.bed` however is of great utility. It allows us to see the splice junctions - which pairs of exons were joined.

Visualizing the alignment and the BED files against the hg38 human genome build for gene name `SYNGR1` shows




# How do I perform differential expression with Cuffdiff?

To run cuffdiff for a pairwise comparison, list your files as comma -separated groups. Within a group files are separated with commas whereas the groups are separated with a space: `A1,A2,A3 B1,B2,B3` . When the file names are long this can be a bit hard to see.

```
cuffdiff  -o results  $GTF  bam/UHR_1.bam,bam/UHR_2.bam,bam/UHR_3.bam  bam/HBR_1.bam,bam/HBR_2
.bam,bam/HBR_3.bam
```

This run will create files in a folder called `results` that contains a fairly large number of files.

# What type of analysis does CuffDiff perform?

Unlike other tools, `cuffdiff` will at the same time perform a gene-level and multiple transcript level analyses. Hence the need to create a new directory.

```
ls results/
```

Cuffdiff will automatically generate counts at gene, transcript and coding sequence levels. This convenience is actually annoying in that it gives us too many things that we did not ask for. The gene level differential expression can be seen in

```
head results/gene_exp.diff
```

It will produce:

```
test_id    gene_id gene  locus   sample_1  sample_2  status  value_1  value_2  log2(fold_change
)  test_stat  p_value q_value significant
ENSG00000008735.13      ENSG00000008735.13      MAPK8IP2   chr22:50568860-50613981 q1   q2   O
K   61.8108 3065.76  5.63224    7.21919     5e-05    0.00014     yes
ENSG00000015475.18      ENSG00000015475.18      BID        chr22:17734137-17774770 q1   q2   O
K   1719.95 876.279 -0.972901  -1.99275     5e-05    0.00014     yes
ENSG00000025708.12      ENSG00000025708.12      TYMP       chr22:50523567-50532580 q1   q2   O
K   381.412 330.405 -0.207114  -0.354974    0.47205  0.528807    no
ENSG00000025770.18      ENSG00000025770.18      NCAPH2     chr22:50508215-50523472 q1   q2   O
K   1929.37 1139.62 -0.759585  -2.80244     5e-05    0.00014     yes
ENSG00000040608.13      ENSG00000040608.13      RTN4R      chr22:20241414-20283246 q1   q2   O
K   124.515 1157.97  3.21721    2.85563     5e-05    0.00014     yes
ENSG00000054611.13      ENSG00000054611.13      TBC1D22A   chr22:46762616-47175699 q1   q2   O
K   500.343 415.63  -0.267616  -0.543656    0.24845  0.310157    no
```

# RNA-Seq : Zika infected human samples

The example used here is from a publication that was then reanalyzed by three other research groups. We will evaluate the original and the reanalyzed data.

The data for this paper is fairly large and may need a few hours to download. The analysis, however, can be performed much faster.

# How do I automate the Zika data download?

We have placed the commands needed to obtain the data in the script called `zika-getdata.sh` that you can download and run yourself:

```
curl -O http://data.biostarhandbook.com/rnaseq/projects/zika/zika-getdata.sh
```

then run it with:

```
bash zika-getdata.sh
```

This is a recent and realistic data of considerable size (about 20GB). While the download only needs to take place once, it may take some time depending on the internet bandwidth that you have access to.

Note: By default only 1 million reads are extracted. Edit the `zika-getdata.sh` script to change the limit. Make it more than `100 million` (or remove the limit from `fastq-dump` ) to get all the reads.

Once the process completes you will have three directories:

- `info` with the run info.
- `reads` containing the sequencing reads.
- `refs` containing the prebuilt indices and gene information.

The data will be ready to be analyzed. Read on to understand what this script does.

# Where is the original publication?

Original paper:

- Zika Virus Targets Human Cortical Neural Precursors and Attenuates Their Growth in Cell Stem Cell. 2016 May 5
- Data for the paper from NCBI BioProject: PRJNA313294 and GEO: GSE78711

The two accession numbers that we need to keep track of are `PRJNA313294` and `GSE78711`

The two reanalyses are:

- An open RNA-Seq data analysis pipeline tutorial with an example of reprocessing data from a recent Zika virus study

- Zika infection of neural progenitor cells perturbs transcription in neurodevelopmental pathways

# What data does the project contain?

The dataset consists of RNA-Seq libraries of zika infected (treatment) and mock infected (control) human neural progenitor cells (hNPCs). 75 base pair paired-end reads were sequenced using an Illumina MiSeq. There are 2 replicates for both mock and zika samples.

```
Sample    Accession   Condition   Library-type   Seq-machine Total-Reads
Mock1-1   SRR3191542     mock      paired-end      MiSeq       15855554
Mock2-1   SRR3191543     mock      paired-end      MiSeq       14782152
ZIKV1-1   SRR3191544     zika      paired-end      MiSeq       14723054
ZIKV2-1   SRR3191545     zika      paired-end      MiSeq       15242694
```

Also, the same data was sequenced again using an Illumina NextSeq in 75 bp single-end mode.

```
Sample    Accession   Condition   Library-type   Seq-machine Total-Reads
Mock1-2   SRR3194428     mock      single-end      NextSeq     72983243
Mock2-2   SRR3194429     mock      single-end      NextSeq     94729809
ZIKV1-2   SRR3194430     zika      single-end      NextSeq     71055823
ZIKV2-2   SRR3194431     zika      single-end      NextSeq     66528035
```

# How do I analyze data generated on different sequencing platforms?

Having two rounds of sequencing in the dataset leads to questions about how to analyze such a dataset - Should we pool the data from the two runs or should we keep them separate and treat them as another replicate? Another difficulty here -- since the dataset consists of both paired-end and single-end samples, commands need to be run separately for each.

# How do I obtain the gene expression results?

Installing the `geodata` script as described in http://data.biostarhandbook.com/scripts/README.html allows you to query the data:

```
geodata GSE78711
```

produces the output in so called "soft" format that among many lines contains:

```
!Series_supplementary_file = ftp://ftp-trace.ncbi.nlm.nih.gov/sra/sra-instant/reads/ByStudy/sra/SRP/SRP070/SRP070895
!Series_supplementary_file = ftp://ftp.ncbi.nlm.nih.gov/geo/series/GSE78nnn/GSE78711/suppl/GSE78711_Table.S2A.reads_R1.xlsx
!Series_supplementary_file = ftp://ftp.ncbi.nlm.nih.gov/geo/series/GSE78nnn/GSE78711/suppl/GSE78711_gene.exp.all.txt.gz
```

You can write a simple script to extract the right fields or just copy and paste them (since there are only two files) like so:

```
curl -Ok ftp://ftp.ncbi.nlm.nih.gov/geo/series/GSE78nnn/GSE78711/suppl/GSE78711_Table.S2A.reads_R1.xlsx
curl -Ok ftp://ftp.ncbi.nlm.nih.gov/geo/series/GSE78nnn/GSE78711/suppl/GSE78711_gene.exp.all.txt.gz
```

Unzip the result:

```
gunzip GSE78711_gene.exp.all.txt.gz
```

The resulting file called `GSE78711_gene.exp.all.txt` contains the results of the analysis:

```
cat GSE78711_gene.exp.all.txt | grep yes | head
```

A region of the `GSE78711_gene.exp.all.txt` file is shown:

```
gene          sample_1  sample_2  value_1    value_2     log2.fold      p_value   significant
SIGLEC10        Mock      ZIKV     0.810017   0.0664947   -3.60664        0.00005      yes
CDC20B          Mock      ZIKV     3.83895    0.353524    -3.44083        0.0002       yes
LOC100129083    Mock      ZIKV     2.41317    0.227106    -3.4095         0.0193       yes
OR51E2          Mock      ZIKV     0.914167   0.109813    -3.05742        0.0001       yes
CCNO            Mock      ZIKV     4.30137    0.53558     -3.00562        0.00005      yes
SLCO4A1         Mock      ZIKV     1.01092    0.128567    -2.97508        0.0229       yes
SGPP2           Mock      ZIKV     1.46204    0.226524    -2.69025        0.00035      yes
RRM2            Mock      ZIKV     34.1419    5.48555     -2.63783        0.00005      yes
SFXN2           Mock      ZIKV     5.93441    0.967533    -2.61672        0.00005      yes
COLEC12         Mock      ZIKV     0.875222   0.143901    -2.60458        0.00005      yes
```

From this view, it is clear that the authors performed a gene level analysis where they characterize all exons that can form a gene at the same time rather than investigating the individual transcripts that are observed.

Our goal is to produce this or a similar result.

# How do I get the sequencing data?

We can download the data from SRA using SRA-toolkit.

Get the data for the zika run:

```
esearch -db sra -query PRJNA313294 | efetch -format runinfo > zika.csv
```

Since we have both paired-end and single-end sequencing libraries, we have to do more work to separate them out.

Isolate the run ids by library type information.

```
cat zika.csv |cut -f 1,16 -d , | grep  SRR.*SINGLE | cut -f 1 -d "," > single_ids.txt
```

The content of file `single_ids.txt` file is:

```
SRR3194428
SRR3194429
SRR3194430
SRR3194431
```

Similarly, we can get SRA ids for paired end samples like this:

```
cat zika.csv | cut -f 1,16 -d ',' | grep  SRR.*PAIRED | cut -f 1 -d "," > paired_ids.txt
```

Now that we have the accessions, we can get the sequence files in fastq format using `fastq-dump` command in SRA-toolkit. Since we'll have lots of files let's place them into a directory called `reads`

```
mkdir -p reads
```

If you are using `Bash on Windows` your will need to run wonderdump first, otherwise skip the next line:

```
cat single_ids.txt paired_ids.txt | xargs -n 1 echo wonderdump | bash
```

Get fastq files for single-end samples:

```
cat single_ids.txt  | xargs -n 1 fastq-dump -O reads
```

Get fastq files for paired-end samples. The `--split-files` option will split the file into pairs:

```
cat paired_ids.txt  | xargs -n 1 fastq-dump --split-files -O reads
```

We can run `seqkit` to generate a report on files with:

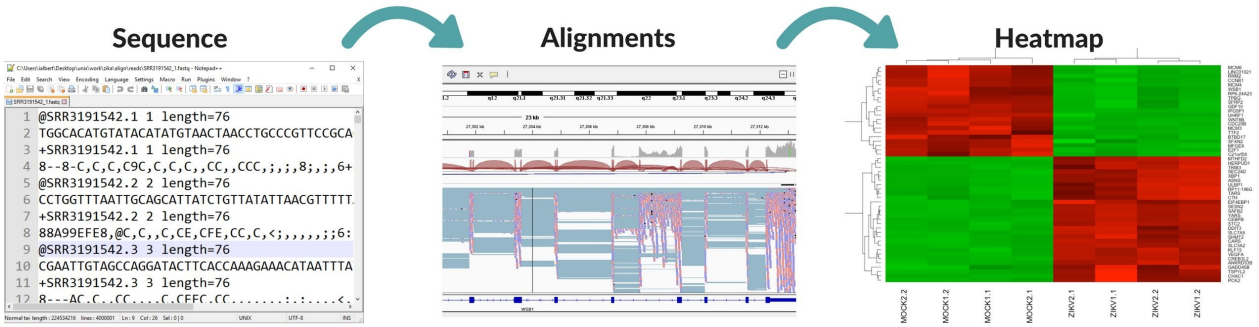
```
seqkit stat reads/*
```

This produces the output. We can see that the paired-end files contain far fewer reads than single-end reads (7 million vs 70 million). We can also see that the read lengths vary -- meaning some sort of length trimming was applied to these datasets. Without trimming, the sequencers produced the same length reads (in this case, 76 bases).

| file | format | type | num_seqs | sum_len | min_len | avg_len | max_len |
|---|---|---|---|---|---|---|---|
| reads/SRR3191542_1.fastq | FASTQ | DNA | 7,927,777 | 598,459,167 | 35 | 75.5 | 76 |
| reads/SRR3191542_2.fastq | FASTQ | DNA | 7,927,777 | 598,471,205 | 35 | 75.5 | 76 |
| reads/SRR3191543_1.fastq | FASTQ | DNA | 7,391,076 | 557,684,818 | 35 | 75.5 | 76 |
| reads/SRR3191543_2.fastq | FASTQ | DNA | 7,391,076 | 557,625,834 | 35 | 75.4 | 76 |
| reads/SRR3191544_1.fastq | FASTQ | DNA | 7,361,527 | 555,374,256 | 35 | 75.4 | 76 |
| reads/SRR3191544_2.fastq | FASTQ | DNA | 7,361,527 | 555,388,034 | 35 | 75.4 | 76 |
| reads/SRR3191545_1.fastq | FASTQ | DNA | 7,621,347 | 574,157,249 | 35 | 75.3 | 76 |
| reads/SRR3191545_2.fastq | FASTQ | DNA | 7,621,347 | 574,145,168 | 35 | 75.3 | 76 |
| reads/SRR3194428.fastq | FASTQ | DNA | 72,983,243 | 5,503,017,518 | 35 | 75.4 | 76 |
| reads/SRR3194429.fastq | FASTQ | DNA | 94,729,809 | 7,137,391,714 | 35 | 75.3 | 76 |
| reads/SRR3194430.fastq | FASTQ | DNA | 76,299,868 | 5,747,471,557 | 35 | 75.3 | 76 |
| reads/SRR3194431.fastq | FASTQ | DNA | 66,528,035 | 5,008,831,278 | 35 | 75.3 | 76 |

# Zika RNA-Seq study: Alignment based analysis

## What will be done in this section?




The three stages of RNA-Seq will be performed with:

1. Alignment: `HISAT2`
2. Quantification: `featureCounts`
3. Differential expression: `DESeq` .

HISAT2 is a splice-aware aligner for mapping next-generation sequence data against a reference genome. It uses similar algorithms as Bowtie2 but has much improved speed. HISAT2 produces output alignments in SAM format.

## How do I automate the Zika data processing?

We have placed the commands for the stages of RNA-Seq analysis into three separate scripts:

1. `zika-getdata.sh` obtains the data.
2. `zika-align.sh` aligns the data.
3. `zika-count.sh` quantifies the data.

You can download each script from the data site http://data.biostarhandbook.com/rnaseq/projects/zika/.

But we also have a `Makefile` called zike-makefile for this project. If you haven't yet dealt with this concept see the chapter on Automation and Make). You can get the `Makefile` like this:

```
curl -O http://data.biostarhandbook.com/rnaseq/projects/zika/zikamake
```

We use this file to demonstrate a better data analysis practice. Look into the file to see what it contains. Now run the file to see what it can do:

```
make -f zikamakefile
```

It will print the available tasks that you can run:

```
Select a task: getscripts, getdata, align, count, deseq1, deseq2, edger
```

Now you can execute each in turn:

```
make -f zikamake getscripts
```

This Will obtain all the scripts that need to be run. Here it is important to make a point. I decided to do THIS analysis using a makefile. It is not how ALL analyses have to be done, nor is it a testament that this is the BEST way to do the analysis. The benefit it gives you is a clear track record of what and how you have performed the analysis.

Note that if you rename the file `zikamake` to `Makefile` you would not even need to type the `-f zikamake` anymore, but just `make getdata`, etc. Anyway, now we can run the following:

```
make -f zikamake getdata
make -f zikamake align
make -f zikamake count
make -f zikamake deseq1
```

And just like that our analysis is now done.

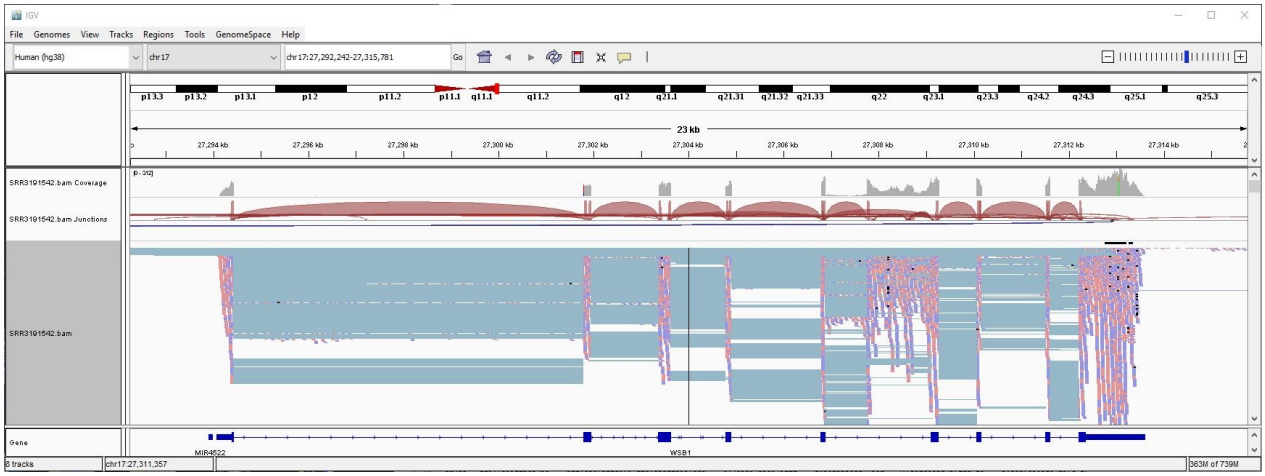
Since this is a recent and realistic dataset with fairly large read counts, even with a fast aligner like `HISAT2`, the alignment may take up to 6 hours on a desktop type system. We sped this up and in the example above by extracting only 1 million reads in the getdata script. This will allow the run to complete in minutes. Remove this limitation from `zika-getdata.sh` to perform a full analysis.

# What just happened when I ran these scripts?

When you run a script from another source (or even your own after the passage of time) it can be disorienting because the script will likely run and generate lots of files, and you may not know what they all are.

How do you figure those out? A Star Wars related joke applies: "Use the Source, Luke!". You have to remember that all these scripts are text files full of instructions. Open them in a text editor and read the commands. Though it can be difficult to understand someone else's code (or your own after enough time), this is your best chance to tell what happened.

For example, in the above script called `zika-align.sh` we've set up a folder that collects the bam files into the folder called `bam`. You can visualize these alignments with IGV.




# How do I visualize the differentially expressed genes?

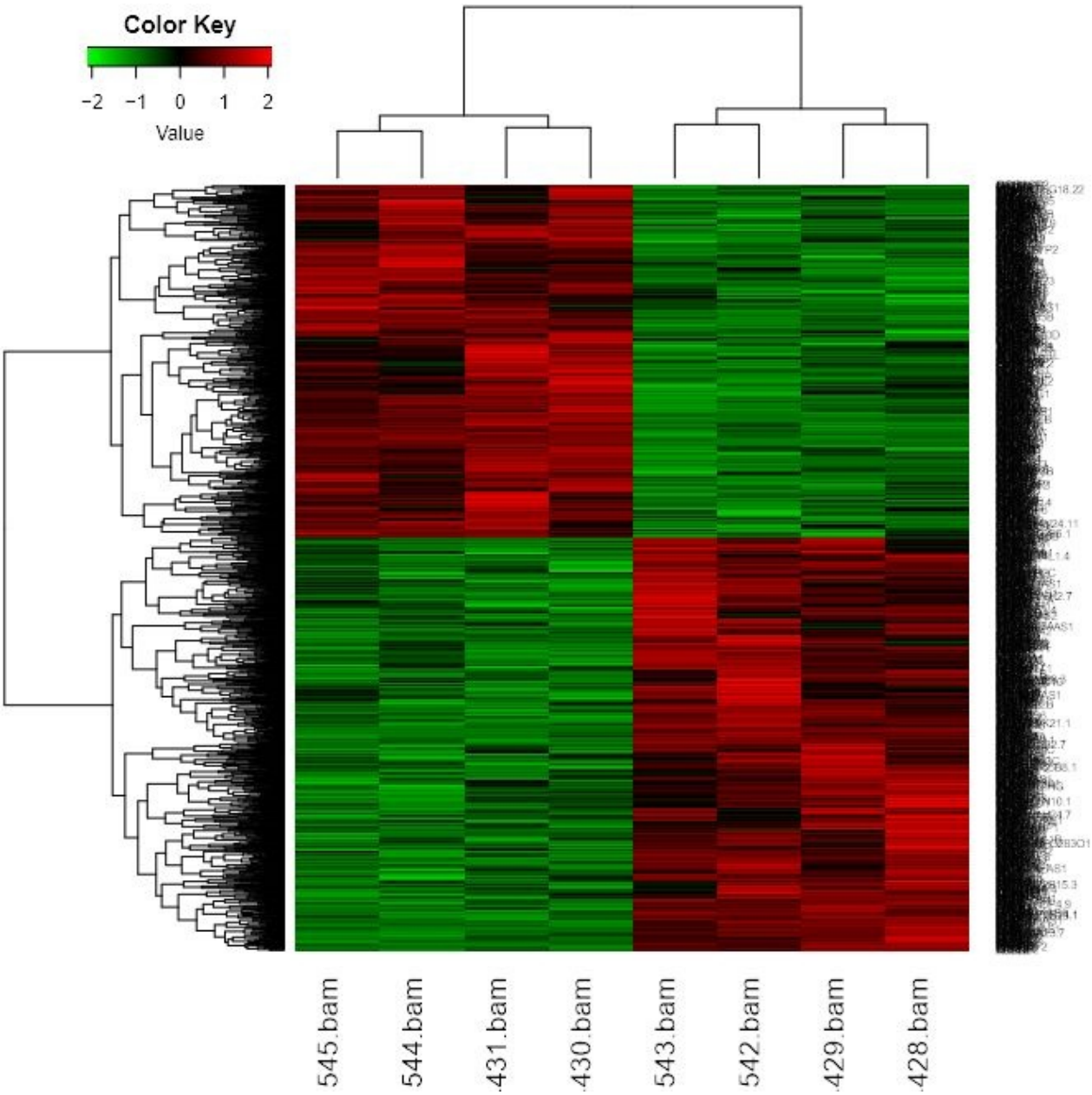
A popular method for visualizing differentially expressed genes is a heatmap. A particular variant of it, the clustered heatmap is so commonly used and so useful (yet non-trivial) to obtain that we've included a script that will generate one for you.

```
curl -O http://data.biostarhandbook.com/rnaseq/code/draw-heatmap.r
```

This script produces a PDF output on the standard output. When you ran your DE script it also produced an additional file called a "normalized matrix". You can generate your heatmap with:

```
cat norm-matrix-deseq1.txt | Rscript draw-heatmap.r > zika-output.pdf
```

Note how this script also writes to standard out, though this time it writes a PDF file. What is in this PDF file? It is a hierarchically clustered heatmap image of your samples and differentially expressed genes:




The plot looks pretty messy at this point, and demonstrates some of the problems with automatically generating results. To fix it, you can edit the file called `norm-matrix-deseq1.txt`, relabel it, and perhaps filter it in some way.

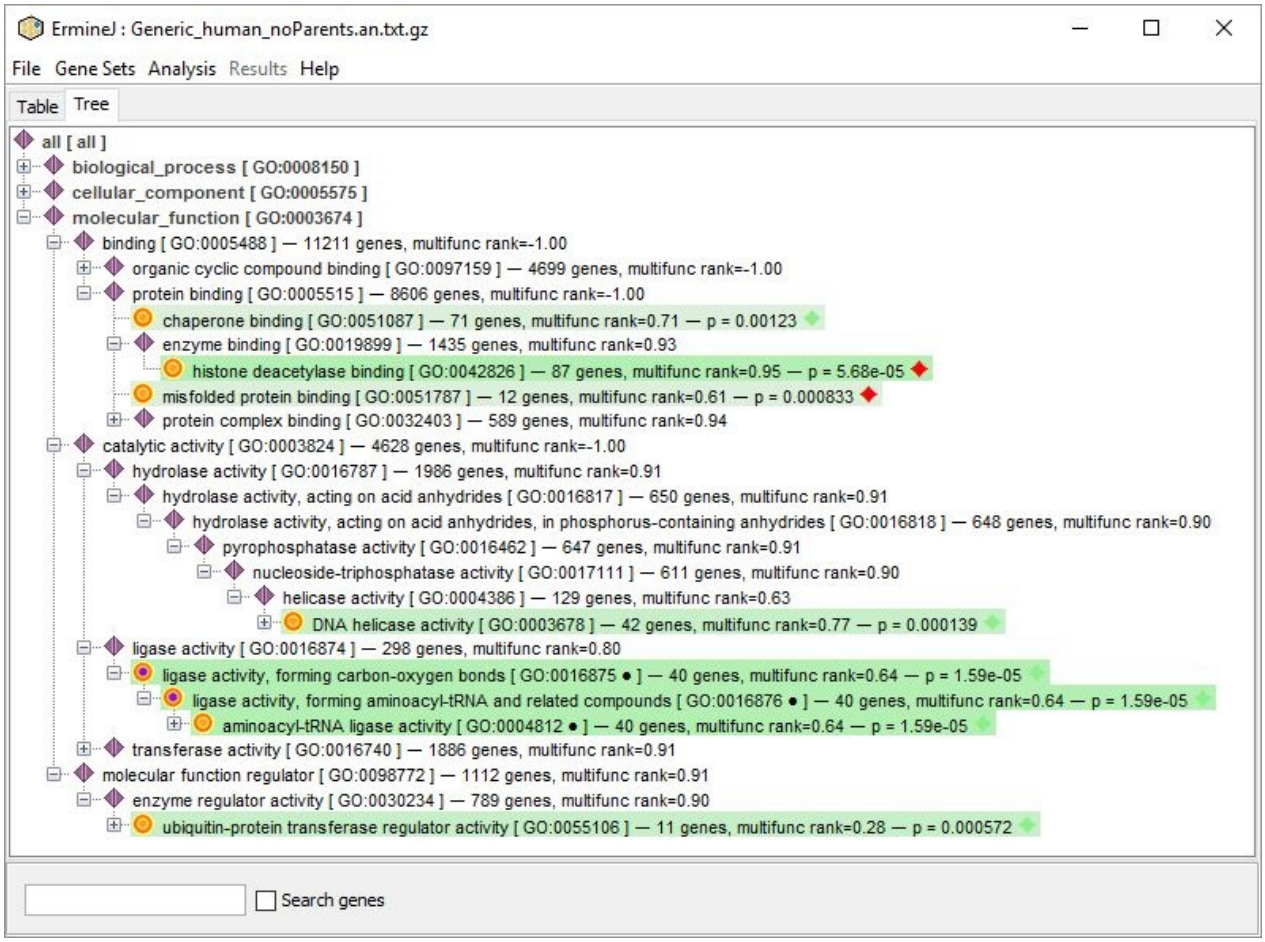
That being said, the plot clearly demonstrates that there are genes that are up/down regulated.

# How do I make sense of this data?

This is where we go back to one of the first chapters in this book. The chapters describing what words mean, how functions are assigned to genes and what gene enrichment analysis is.

The differential expression file that you get above and the normalized matrix can be entered into various tools to investigate the biology behind the measurements.

Here is what we find with the data that we ran above:




# So, how does this all work?

Up to this point we gave you a way to do this analysis quickly. But what is in each of these scripts, how do they work, and how can you customize them? That is what is covered in the next sections.

Remember the steps we mentioned earlier?

1. Alignment: `HISAT2`
2. Quantification: `featureCounts`
3. Differential expression: `DESeq`

Let's get started! To run the `hisat` aligner, first we need an index.

# How do I build a HISAT2 index?

Like any other aligner, HISAT2 needs a reference genome to align against. This reference needs to be indexed using the `hisat2-build` command. Indexing the entire human genome is a compute-intensive task. To avoid everyone having to generate duplicate indices for commonly used genomes, a number of indices have been generated already and can be downloaded from the [hisat2][hisat] website.

In the case of `HISAT2` , in addition to the reference genome index, other annotation files are also present. A pre-built index for GRCh38 assembly can be obtained from the command line:

```
# make a directory to store the references.
mkdir -p ./refs

# This is the URL for the index.
URL=ftp://ftp.ccb.jhu.edu/pub/infphilo/hisat2/data/grch38.tar.gz

# Download and unpack the index.
curl $URL | tar zxv

# Move the index directory to the reference directory
mv grch38 ./refs

# The index has the prefix of "genome".
IDX=refs/grch38/genome
```

If we wanted to index the genome ourselves, we would need to get the FASTA file and run `hisat2-build` on it.

# How do I run a HISAT2 analysis on the zika data?

Note -- this data is slightly unusual. This is a very common occurance that we need to be ready for. Since the zika infected human sample has both paired end and single end library preparations, we can't run them in one step. We need to run HISAT2 separately in single end (SE) and paired end (PE) mode.

Single-end samples can be aligned with

```
# Make a directory to store the alignments.
mkdir -p bam

# Set up some shortcuts to refer to SRR id, input, and output files.
SRR=SRR3194428
R1=reads/${SRR}.fastq
BAM=bam/${SRR}.bam

# The index has the prefix of "genome".
IDX=refs/grch38/genome

# Run the HISAT2 aligner in single end mode.
hisat2 -x $IDX -U $R1  | samtools sort > $BAM
```

HISAT2 alignment of paired-end reads is as follows:

```
# Make a directory to store the alignments. -p will make it only if it doesn't yet exist.
mkdir -p bam

# Set up some shortcuts to refer to SRR id, input, and output files.
SRR=SRR3191542
R1=reads/${SRR}_1.fastq
R2=reads/${SRR}_2.fastq
BAM=bam/${SRR}.bam

# Run the HISAT2 aligner in paired-end mode.
hisat2 -x $IDX -1 $R1 -2 $R2 | samtools sort > $BAM
samtools index $BAM
```

# How do I automate HISAT2 analysis?

You can download or copy the pipeline below. HISAT2 alignments for all samples can produced using the script below. This assumes you have downloaded all the fastq files to a directory called `data` and that you have created files named 'single_ids.txt' and 'paired_ids.txt'. All the output alignments will be collected in the `bam` directory.

```
# Stop on any error.
set -ueo pipefail

# Make a directory to store bam files.
mkdir -p bam

# How many CPU cores are on the system?
CPUS=4

# The HISAT2 index name.
IDX=refs/grch38/genome

# Keep track of what the aligners print out.
RUNLOG=runlog.txt

echo "Run started by `whoami` on `date`" > $RUNLOG
# HISAT2 run on PE samples.
for SAMPLE in $(cat paired_ids.txt)
do
    R1=reads/${SAMPLE}_1.fastq
    R2=reads/${SAMPLE}_2.fastq
    BAM=bam/${SAMPLE}.bam
    SUMMARY=bam/${SAMPLE}_summary.txt

    echo "Running HISAT2 on paired end $SAMPLE"
    hisat2 -p $CPUS -x $IDX -1 $R1 -2 $R2 | samtools sort > $BAM  2> $RUNLOG
    samtools index $BAM
done

# HISAT2 run on single end samples.
for SAMPLE in $(cat single_ids.txt)
do
    R1=reads/${SAMPLE}.fastq
    BAM=bam/${SAMPLE}.bam
    SUMMARY=bam/${SAMPLE}_summary.txt

    echo "Running Hisat2 on single end: $SAMPLE"
    hisat2 -p $CPUS -x $IDX -U $R1 | samtools sort > $BAM 2> $RUNLOG
    samtools index $BAM
done
```

HISAT2 alignments can be visualized by loading the bam files in Integrative Genomics Viewer (IGV).

# How do I quantify the HISAT2 alignments?

First, you need a file that contains the genomic coordinates of annotated genes:

```
# Set up some shortcuts.
mkdir -p refs
GTF=refs/GRCh38.gtf
URL=ftp://ftp.ensembl.org/pub/release-86/gtf/homo_sapiens/Homo_sapiens.GRCh38.86.gtf.gz

#Download feature file from Ensembl and unzip before saving.
curl $URL | gunzip -c > $GTF
```

Run featureCounts on a single dataset to count reads on genes:

```
featureCounts -a $GTF -o counts.txt bam/SRR3191542.bam
```

The sample layout is the following:

```
# These are the control samples.
MOCK=(SRR3191542 SRR3191543 SRR3194428 SRR3194429)

# These are the ZIKV numbers.
ZIKV=(SRR3191544 SRR3191545 SRR3194430 SRR3194431)
```

It is very error prone to generate filenames from meaningless numbers like the ones above. It is better to use programming constructs to create filenames like so:

```
# Iterate over arrays and create file names from them.
MOCK=(SRR3191542 SRR3191543 SRR3194428 SRR3194429)
for NAME in ${MOCK[@]}; do
    MOCK_BAMS+="bam/${NAME}.bam "
done

ZIKV=(SRR3191544 SRR3191545 SRR3194430 SRR3194431)
for NAME in ${ZIKV[@]}; do
    ZIKV_BAMS+="bam/${NAME}.bam "
done
```

all that these commands do is generate values for the variables `$MOCK_BAMS` and `$ZIKV_BAMS`. The variables will contain the following:

```
echo $MOCK_BAMS
```

writes:

```
bam/SRR3191542.bam bam/SRR3191543.bam bam/SRR3194428.bam bam/SRR3194429.bam
```

and:

```
echo $ZIKV_BAMS
```

prints:

```
bam/SRR3191544.bam bam/SRR3191545.bam bam/SRR3194430.bam bam/SRR3194431.bam
```

Now, was it really worth writing that mini program just to generate these name? You'll be the judge of what works for you. Each approach has advantages and disadvantages. It is pretty clear that if you had many more samples it would be error prone to manually edit names.

We can now launch `featureCounts` knowing we got all files:

```
featureCounts -p -g gene_name -a $GTF -o counts.txt $MOCK_BAMS $ZIKV_BAMS
```

It produces the file called `counts.txt` with our estimates.

# How do I find the differentially expressed genes?

We have not just one, but three methods at your disposal. But please make sure you also read the chapter How do I choose an RNA-Seq method

In order to find differentially expressed genes using DESeq, you can use our R script as described on the RNA-Seq with Bioconductor page.

Get the script like this:

```
curl -O http://data.biostarhandbook.com/rnaseq/code/deseq1.r
```

The script requires an input file with gene names and counts. We can remove the intermediate columns:

```
cat counts.txt | cut -f 1,7-14 > sample_counts.txt
```

`sample_counts.txt` should look like

```
SCNN1D   78    98     653    946    51     61    499     422
ACAP3    1528  1610   11871  17377  1649   1509  14102   10830
MIR6726  0     0      0      1      0      0     0       0
PUSL1    67    68     437    670    81     88    656     553
```

Then, we can run the differential expression analysis script like this:

```
cat sample_counts.txt | Rscript deseq1.r 4x4 > results_deseq1.txt
```

The `4x4` parameter specifies the design of the experiment. We mean 4 replicates for each of MOCK and ZIKAV conditions.

# What is the result of the differential expression analysis?

The differential expression analysis script produces a table with changes in gene expression across the two conditions. The change in gene expression is shown as fold change. The p-value and adjusted p-values generated by DESeq are also given.

```
id      baseMean     baseMeanA     baseMeanB     foldChange    log2FoldChange  pval      padj
SEC24D  613.732089   129.834166    1097.630012   8.454092196   3.079649846     7.09E-73  2.91E
-68
SLC7A5  5451.691973  1661.110476   9242.273469   5.563912576   2.476099751     2.13E-69  4.36E
-65
XBP1    2481.903296  746.6520304   4217.154561   5.648085573   2.497761947     2.43E-68  3.33E
-64
HERPUD1 2328.813848  741.1048934   3916.522803   5.28470779    2.401823702     1.36E-65  1.40E
-61
```

- `id` - Gene or transcript name that the differential expression is computed for
- `baseMean` - The average normalized value across all samples
- `baseMeanA` and `baseMeanB` - The average normalized gene expression for each condition
- `foldChange` - The ratio of values, computed as `baseMeanB/baseMeanA`.
- `log2FoldChange` - A log2 transform of `foldChange`. When we apply a base 2 logarithm, the values become symmetrical around 0. A log2 fold change of 1 and -1 indicate a doubling or halving of the expression level, respectively.
- `pval` - The p-value generated by DESeq.
- `padj` - The p-value corrected by multiple hypothesis testing.

# How do the results relate to expected results?

```
cat results_deseq1.txt | awk '$8 <0.05 {print $0}' | wc -l
4451

cat results_deseq1.txt | awk '$7 <0.05 {print $0}' | wc -l
7131
```

We have 4451 differentially expressed genes with `padjusted <0.05`. If we consider `p-value <0.05`, then there are 7131 significant genes. You may use p-value if you already had selected your target gene before evaluating these results. In all other cases, the adjusted p-value should be used as this corrects for multiple testing errors.

If we had run `DESeq2`, we would have 9497 differentially expressed genes with `p-adjusted <0.05`.

From the `GSE78711_gene.exp.all.txt` file that you've already downloaded (see zika-data chapter) there are 6864 differentially expressed genes with `p-value < 0.05`. Adjusted p-values are not given.

```
cat GSE78711_gene.exp.all.txt | grep "yes" | wc -l
6864
```

Now, let's take a look at some individual genes and see how the fold change varies between the methods.

The fold change for gene `IFIT2` from the downloaded results is `5.23`. Deseq1 reports `4.45` and DESeq2 reports `3.4` fold change for the same gene. But for gene `CEBPB` downloaded results and both DESeq1 and DESeq2 results are in agreement. (3.08,3.07,2.99)

Please note that since different methods use different statistics and have different sensitivities to false positives, there will always be some variation in the results.

# Zika RNA-Seq study: Mapping based analysis

## What will be done in this section?

The three stages of RNA-Seq will be performed with:

1. Alignment: `kallisto`
2. Quantification: `kallisto`
3. Differential expression: `DESeq`

Kallisto is a tool for quantifying transcript abundances. It performs a *pseudoalignment* of reads against a transcriptome. In a pseudoalignment, the program tries to specify for each read the target that it originates from (using k-mers) and not where in the the target it aligned to. This makes the algorithm much faster than a 'real' alignment algorithm.

Both kallisto and sleuth are programs from Lior Patcher's group. Sleuth can work only with kallisto quantified data.

Typing `kallisto` on the commandline will show all the available commands that can be used. The `index` command builds the kallisto index and the `quant` command runs the transcript abundance quantification algorithm.

## How do I automate a Kallisto run?

We have placed the commands for the stages of RNA-Seq analysis into two separate scripts:

1. `zika-getdata.sh` downloads and prepares the data.
2. `zika-kallisto.sh` performs an RNA-Seq quantification with kallisto.

Download and run the scripts like so:

```
curl -O http://data.biostarhandbook.com/rnaseq/projects/zika/zika-getdata.sh
curl -O http://data.biostarhandbook.com/rnaseq/projects/zika/zika-kallisto.sh
```

then run then with:

```
bash zika-getdata.sh
bash zika-kallisto.sh
```

Read on to understand more about what these scripts do.

## How do I build a Kallisto index?

Kallisto uses a transcriptome as a reference. The first step to run Kallisto is to get the transcriptome and build a Kallisto specific index. Kallisto does not provide any pre-built indices.

To perform an RNA-Seq analysis for the zika-infected human dataset with the Kallisto-Sleuth pipeline, we need to get a file with all human transcripts.

Download all GRCh38 assembly specific cDNA sequences for the human genome:

```
# Set directories and a shortcut for the reference.
mkdir -p refs
REF=refs/GRCh38.cdna.fa
IDX=refs/GRCh38.cdna.fa.idx

# Download the transcriptome from Ensembl.
curl ftp://ftp.ensembl.org/pub/release-86/fasta/homo_sapiens/cdna/Homo_sapiens.GRCh38.cdna.all
.fa.gz | gunzip -c > $REF
```

Build the kallisto index

```
kallisto index -i $IDX $REF
```

# How do I quantify transcripts with Kallisto?

Kallisto assigns reads to transcripts and calculates the abundance in one step. The algorithm can be invoked using `quant` command.

We are using the zika-infected samples that we have already downloaded in the 'data' directory.

Paired-end data can be quantified like this:

```
# Create an output directory.
mkdir -p results

# Define the locations of the reference and transcriptome index.
REF=refs/GRCh38.cdna.fa
IDX=refs/GRCh38.cdna.fa.idx

# Store the run id in a variable.
SAMPLE=SRR3191542
R1=reads/${SAMPLE}_1.fastq
R2=reads/${SAMPLE}_2.fastq

# Run the kallisto quant command.
kallisto quant -i $IDX -o $SAMPLE -b 100 $R1 $R2

# The output of kallisto is a directory. Move it to a known location.
mv $SAMPLE results
```

The above command creates a directory specified with the `-o` flag, in this case `SRR3191542`, and places the result files there. This whole directory is then moved to the `results` directory so that all the results are kept in one place. The `-b` option specifies number of bootstraps. Specifying this option is advantageous as the sleuth program can make use of this to improve differential expression detection.

To quantify single end data, the `--single` option needs to be specified. Fragment length ( `--l` ) and standard deviation ( `--s` ) of the fragment length are also mandatory parameters. According to kallisto manual "Illumina libraries produce fragment lengths ranging from 180–200 bp but it's best to determine this from a library quantification with an instrument such as an Agilent Bioanalyzer."

The dataset used in this example is based on this paper from kallisto group where they are using 187 as fragment length and 70 as standard deviation.

```
SAMPLE=SRR3194428
R1=reads/${SAMPLE}_1.fastq

# Run the kallisto quant command in single-end mode.
kallisto quant -i $IDX -o $SAMPLE -b 100 --single -l 187 -s 70 $R1

# Move the output to the results folder.
mv $SAMPLE results
```

# What are the outputs of Kallisto?

All kallisto produced output files will be placed in the specified output directory ( `-o` ). Transcript abundance estimate results are in a tab delimited file named `abundance.tsv` . `abundance.tsv` file looks like this:

```
target_id           length  eff_length  est_counts  tpm
ENST00000628275.2   453        275.603     2.76617     2.97517
ENST00000628424.1   322     148.792     1.94712     3.87909
ENST00000630311.1   270     102.019     0.675219    1.96191
ENST00000628947.1   535     357.24      6.5         5.39349
ENST00000628339.2   2953    2775.11     1.87061     0.199811
```

Effective length (the third column) scales the transcript length by the fragment length distribution. It takes into account the part of the transcript where fragments would overlap fully in any location. It is usually calculated as `transcript length - mean(fragment length) + 1` . Abundances are reported in "estimated counts" (est_counts) and in Transcripts Per Million (TPM).

# How do I run kallisto for all the samples?

The script below can be used to automate kallisto runs for all samples. This assumes you have downloaded all the fastq files to the `data` directory and you have 'single_ids.txt' and 'paired_ids.txt' files. All the output alignments will be collected in the `kallisto` directory.

```
# Run kallisto on paired end reads.
for SRR in $(cat info/paired.csv)
do
    R1=data/${SRR}_1.fastq
    R2=data/${SRR}_2.fastq

    echo "*** Running kalisto on paired end sample: $SRR"
    kallisto quant -i $IDX -o $SRR -b 100 $R1 $R1
done

# Run kallisto on single end reads.
for SRR in $(cat single.csv)
do
    R1=data/${SRR}.fastq
    echo "*** Running kalisto on single end sample: $SRR"
    kallisto quant -i $IDX -o $SRR -b 100 --single -l 187 -s 70 $R1
done
```

# How do I quantify the results?

By the end of the run above, we have an output directory named after each sample.

```
ls -d SRR*
```

will list:

```
SRR3191542  SRR3191543  SRR3191544  SRR3191545  SRR3194428  SRR3194429  SRR3194430  SRR3194431
```

As in the alignment based analysis, we can build variables to match sample names to conditions.

```
# These are the control samples.
MOCK=(SRR3191542 SRR3191543 SRR3194428 SRR3194429)

# These are the ZIKV numbers.
ZIKV=(SRR3191544 SRR3191545 SRR3194430 SRR3194431)
```

And you can create some variables to help create count files:

```
MOCK_FILES=''
for NAME in ${MOCK[@]}; do
    MOCK_FILES+="${NAME}/abundance.tsv "
done

ZIKV_FILES=''
for NAME in ${ZIKV[@]}; do
    ZIKV_FILES+="${NAME}/abundance.tsv "
done
```

check that the names are correct with:

```
ls $MOCK_FILES $ZIKV_FILES
```

We built these variables so that we don't mess up the order here:

```
paste $MOCK_FILES  $ZIKV_FILES | cut -f 1,4,9,14,19,24,29,34,39  > counts.txt
```

Counts file `counts.txt` will be as shown below. This can be used as input to downstream DEseq differential expression analysis.

```
target_id          est_counts  est_counts est_counts  est_counts  est_counts  est_counts
ENST00000472225.6    6.11331    1.28737     10.0567     2.23768     2.1543      0
ENST00000496578.3    3.12574    3.97029     8.18468     0           1.6547      1.49019
ENST00000624726.1    6.73189    3.0668      1.16443     2.39545     4.31444     1.26448
```

You may change the header to include the sample names.

# How do I find differentially expressed transcripts?

Whereas the previous analyses were gene based (though we could have used transcripts there as well), kallisto works on transcript data. You could, of course, emulate the "gene" level analysis by creating "fake" transcripts that are made up of concatenated exons (but there is no reason to).

Let's get our differential expression script and run it:

```
curl -O http://data.biostarhandbook.com/rnaseq/code/deseq1.r
```

But, surprise! The script won't run. You'll have a bit of a problem here. At the transcript level your data may have many transcripts that are not detected at all. These produce zeros for all samples. Usually, you don't want to include these in the statistical analysis. These values may skew the analysis. You need a method to filter out the empty rows.

For example, you might choose to only keep rows where the sum of all counts is larger than 25. Basically keep only the transcripts supported by at least 25 reads. It is never clear (at least not to us) how to find this threshold in an optimal way. Most do it by trial and error. So let's filter by counts:

```
cat counts.txt | awk ' ($2+$3+$4+$5+$6+$7+$8+$9) > 25 { print $0 } ' > temp.txt
```

This drops the number of rows from 178,126 to 26,759.

You actually have another problem here as well. The kallisto estimator uses effective lengths and produces floating point numbers rather than integers for counts. Some tools (like DESeq2) will balk at that, and you need to convert them to integers instead. You can do this inside R or with an awk script by applying the formatting operator `%3.0f` on each element like so:

```
 cat temp.txt | awk ' { printf("%s\t%3.0f\t%3.0f\t%3.0f\t%3.0f\t%3.0f\t%3.0f\t%3.0f\t%3.0f\n",
$1, $2, $3, $4, $5, $6, $7, $8, $9) } ' > valid.txt
```

We admit the above construct looks pretty absurd -- but hey, sometimes awk can save you a lot of time and effort. Just be sure to add a comment with what you were intending to do. Now your chest might swell with pride. Look at this pretty data!

```
ENST00000628983.1        40      42      32      47      64      64      79
ENST00000630362.2        32      32      47      38      24      25      20
ENST00000629233.2         8      18      47       9       0      29      28
ENST00000628523.2         0       0       0       5      24       4       0
ENST00000630001.2        50      40      16      34      22      22      11
```

Let's run our differential expression estimator:

```
cat valid.txt | Rscript deseq1.r 4x4 > zika-results.txt
```

Unless you pick another method the `zika-results.txt` is your final answer. On to interpreting the results.

# Interval datatypes

Although genomes exist in three-dimensional space, in bioinformatics we often simplify our model of the genome to one or two dimensions in order to facilitate visualization and analysis tasks. Hence, coordinates for spatially describing genomic loci tend to follow the general pattern of `chromosome`, `start`, `end` where "start" and "end" are both integers that represent the left and right positions of an interval. Since nucleic acid molecules are polar there is a directionality to each interval typically indicated by the `strand` column of the data.

There are two common interval representation formats: `BED` and `GFF`. Each is a tab delimited format where columns have different designations:

- GFF3: Generic Feature Format
- BED: Browser Extensible Data

In addition GFF has prior versions such as:

- GFF2: Generic Feature Format
- GTF: Gene Transfer Format

## What is an interval data-type?

An interval data type usually represents 1D or 2D information along the coordinates of a sequence (usually a chromosome). The following classes of information may be represented:

- Span: A feature starts at a coordinate and ends at another.
- Hierarchy: A feature "belongs" to another feature, for example exons to a transcript.
- Value: The value at a coordinate is *100* (say).
- Annotation: The feature named *X* is also characterized by *Y* and *Z*.

## What are the major formats?

There are two competing formats:

- BED that was originally specified by the UCSC genome browser
- GFF General Feature Format and its variants: GFF2, GTF and GFF3

## What are the differences between the BED and GFF formats?

The most glaring of differences are in the coordinate systems that they use.

- `GFF`: 1 based, inclusive on both ends. This means that [1,5] contains 1,2,3,4,5 and that coordinate 1 is the first coordinate of the genome.

- `BED` : is 0 based, non-inclusive on the right. This means that the interval [1, 5) contains 1,2,3,4 and coordinate 1 is the second coordinate of the genome (0 is the first).

The second more subtle yet no less important difference relates to the way hierarchical information is represented

- In the `GFF` format the relationship is built from data distributed over multiple lines.
- In the `BED` format a single line record stores all the information on the block structure of a record.

# Why are there two formats?

It is a clash of philosophies. Technically speaking the zero based coordinate systems are superior in that they let us express certain concepts, such as say ranges more cleanly: first ten are (0, 10), second ten are (10, 20) third ten elements are (20, 30) and so on.

Whereas in a one based system we would write (1, 10), (11, 20), and (21, 30) respectively. So when programming, when coding it is more advantageous to use zero based systems.

But biology is not practiced in code! When talking about any interval with another human being it is very confusing to remember that 1 actually represents the second element and that the last coordinate of anything is not actually in that feature. A gene going from to 100 would not contain the index 100. In these cases the zero based coordinate systems make every operation more error prone.

# Why does the BED format exist?

It is the testament of the influence of the UCSC genome browser that this format is still around. They are the only organization that uses this format. It should have been long banished from bioinformatics.

# What format should I use?

Always use GFF based formats. Perhaps, with time and with your cooperation we can put an end to one of the most confusing, error prone and wasteful choices made in bioinformatics: the zero based coordinate systems.

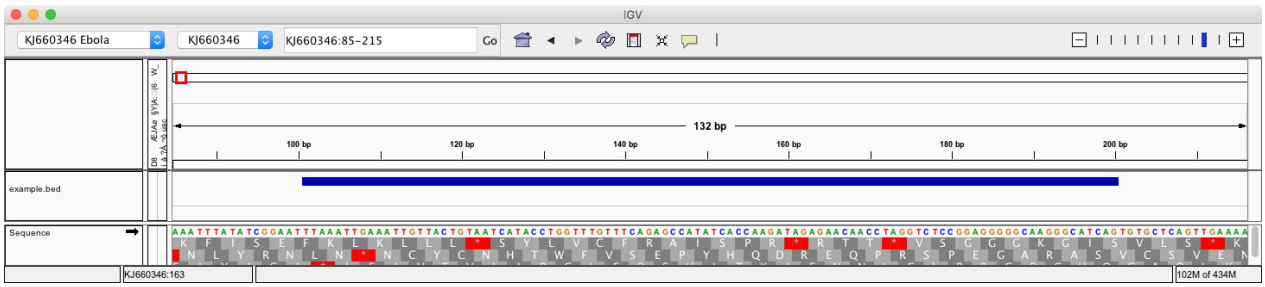
# What does the 3 column BED file look like?

```
REF=refs/KJ660346.fa
mkdir -p refs
efetch -db nucleotide -id KJ660346.1 -format fasta | seqret -filter -sid KJ660346 > $REF
```

Here is a simple BED file - the so called 3 column BED, the simplest interval form. Note that you cannot copy-paste this file since it has to be TAB separated. So make sure to verify that you have tabs.

```
KJ660346    100    200
```

When visualized in IGV (you may need to create a custom genome) this interval file will be:



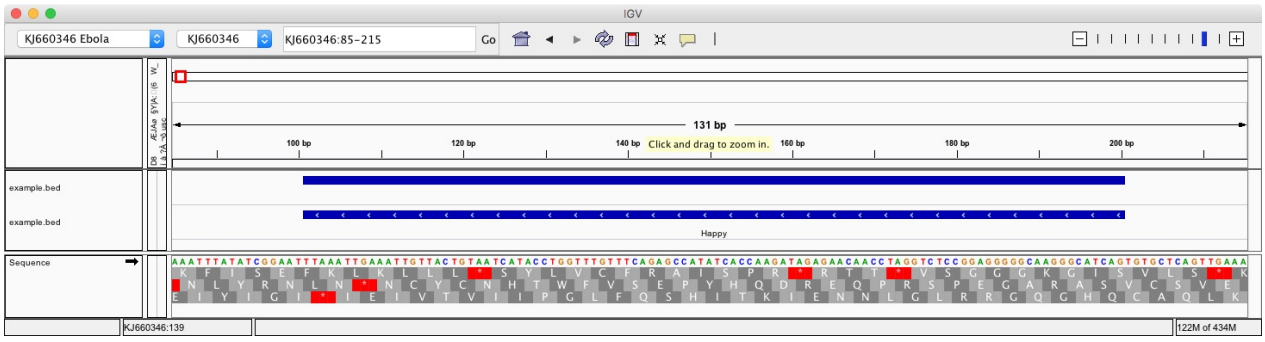

It is not visible on the image above but if you hover the mouse over the intervals, it reads (101-200) in essence the coordinates are being shown as 1 based intervals! This is a very common (and helpful) feature.

# What does a 6 column BED file look like?

As we add more information to the data the display gets more informative. The so called 6 column BED format

```
KJ660346    100    200    Happy    0    -
```

IGV now shows the fourth column as a label and since the feature has a strand, it knows its direction and adds "fishbones" to show the directionality on the glyph.




# What does a GTF/GFF file look like?

We will now represent the same information as GTF and GFF3 files. For GFF/GTF we need to add two more columns such as `source` and `type`. The primary difference between the two is in the last column and has to do with how attributes are represented.

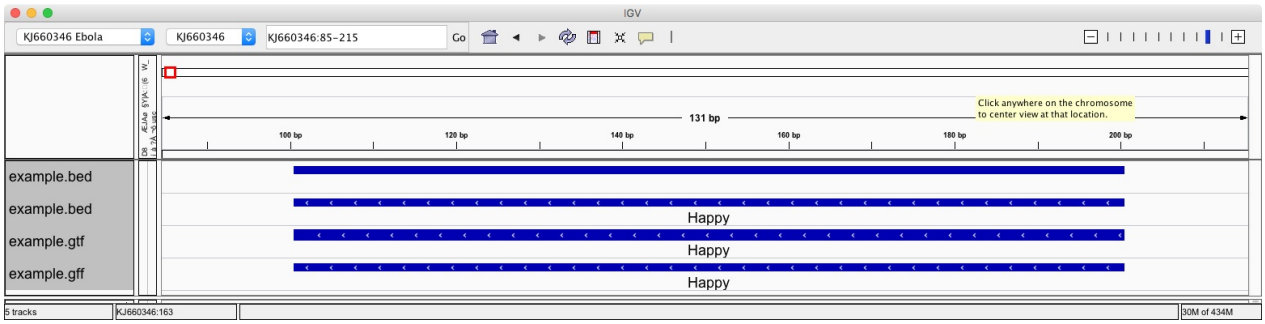
Example GTF file:

```
KJ660346    lecture    CDS    101    200    .    -    .    gene "Happy"; protein_id "HAP2"
```

Example GFF file:

```
##gff-version 3
KJ660346    lecture    CDS    100    200    .    -    .    gene=Happy; protein_id=HAP2
```

All together will look like this (note how the GTF is drawn slightly differently than the GFF). Visualizer such as IGV often make some assumptions that are never clearly articulated.




# How are hierarchical relationships represented?

A relationship indicates multiple features that belong together. For example we need to specify all exons of a transcript.

Lets take a gene called `Happy` that has four exons. These four exons can form three transcripts:

- Transcript `Tic` contains all four exons.
- Transcript `Tac` contains the 1st, 3rd and 4th exon.
- Transcript `Toe` contains the 1st and 3rd exon.

We'll build a representation for each of these transcripts with different standards.

The GFF2 representation repeats each entry as many times as needed even if it means entering redundant data. So for our gene, we have to enter 4 + 3 + 2 = 9 lines.

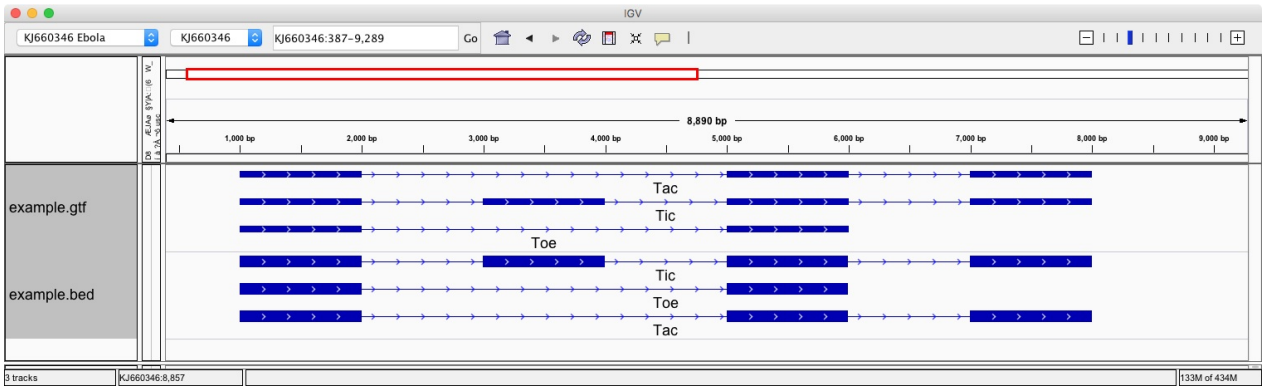
```
##gff-version 2
KJ660346    demo    exon    1000    2000    .    +    .    gene_id "Happy"; transcript_id "Tic"
;
KJ660346    demo    exon    3000    4000    .    +    .    gene_id "Happy"; transcript_id "Tic"
;
KJ660346    demo    exon    5000    6000    .    +    .    gene_id "Happy"; transcript_id "Tic"
;
KJ660346    demo    exon    7000    8000    .    +    .    gene_id "Happy"; transcript_id "Tic"
;

KJ660346    demo    exon    1000    2000    .    +    .    gene_id "Happy"; transcript_id "Tac"
;
KJ660346    demo    exon    5000    6000    .    +    .    gene_id "Happy"; transcript_id "Tac"
;
KJ660346    demo    exon    7000    8000    .    +    .    gene_id "Happy"; transcript_id "Tac"
;

KJ660346    demo    exon    1000    2000    .    +    .    gene_id "Happy"; transcript_id "Toe"
;
KJ660346    demo    exon    5000    6000    .    +    .    gene_id "Happy"; transcript_id "Toe"
;
```

GFF3 representation shortens the above by tagging each entry with the "Parent". This is the feature name it belongs to. This requires us to enter only 4 lines, one per each exon.

```
##gff-version 3
KJ660346    demo    exon    1000    2000    .    +    .    Parent=Tic,Tac,Toe;
KJ660346    demo    exon    3000    4000    .    +    .    Parent=Tic;
KJ660346    demo    exon    5000    6000    .    +    .    Parent=Tic,Tac,Toe;
KJ660346    demo    exon    7000    8000    .    +    .    Parent=Tic,Tac;
```

In the `BED` representation each transcript is fully specified on a SINGLE line. The columns 10, 11 and 12 called `blockCount`, `blockSize` and `blockStart` specify the units by which a feature is split. The `blockStart` is relative to the start coordinate at column 2. Note how the BED format specifies transcripts one each line whereas the BED format had exons on each line.

```
KJ660346    999    8000    Tic    0    +    999    8000    .    4    1000,1000,1000,1000 0,200
0,4000,6000
KJ660346    999    8000    Tac    0    +    999    8000    .    3    1000,1000,1000  0,4000,60
00
KJ660346    999    8000    Toe    0    +    999    8000    .    2    1000,1000   0,4000
```



# Sequence Assembly

## What does "sequence assembly" mean?

Assembly is a "catch-all" term used to describe all processes where we combine shorter individual measurements called "reads" into longer contiguous sequences typically called "contigs."

> ASSEMBLY is the process of turning shorter READS into longer CONTIGS.

Because the sequencing process works by "breaking" the original DNA into smaller fragments, the assembly process is conceptually similar to putting together an image puzzle from its many pieces.

The software that performs the assembly is called the "assembler."

## What applications does assembly have?

First, there is the "obvious" use of identifying the base composition of DNA. Since the majority of organisms have not yet been sequenced, there is plenty to do there.

Interestingly enough there is a resurgence of interest in using assembly methods to identify variants of already known genomes. Alignment based variation calling is inadequate whenever the length of the genomic changes is comparable to measurement (read) lengths. The reads spanning such change often fail to align altogether - or even worse; they align in another location. By assembling these reads into longer contigs, you can identify larger differences in the genomes.

## What are the challenges of assembly?

Sequence assembly is perhaps the application domain of bioinformatics where "skill" and "expertise" are the most difficult to identify and define.

As you will see, it is a field where the procedural method descriptions *"this is the command we used to assemble the genome"* hide the staggering complexity and challenges of finding the particular command reported to work well. Assemblers are quite unlike any other software tool you will ever use. Most come with a bewildering array of parameters - the purpose of which are not explained, yet many of which will have profound effects on the results that they produce.

Trial and error are one of the most commonly used strategies - you will have to keep tuning the parameters and rerun the entire process hoping that the results improve - sometimes in vain. As it turns out, genome assembly is the most computational demanding bioinformatics method of them all. assembling a large genome may take even weeks(!) and substantial computational resources. Thus any expertise built on trial and error will have to be accumulated over a much longer period of time.

Finally, even when assembly appears to work, almost always it will contain several severe and substantial errors. That is where, in our opinion, bioinformatics expertise matters more. THe ability to understande, visualize and correct the mistakes of an assembly has utility that will outlast the present and is more

valuable than knowing the exact invocation of a tool by heart.

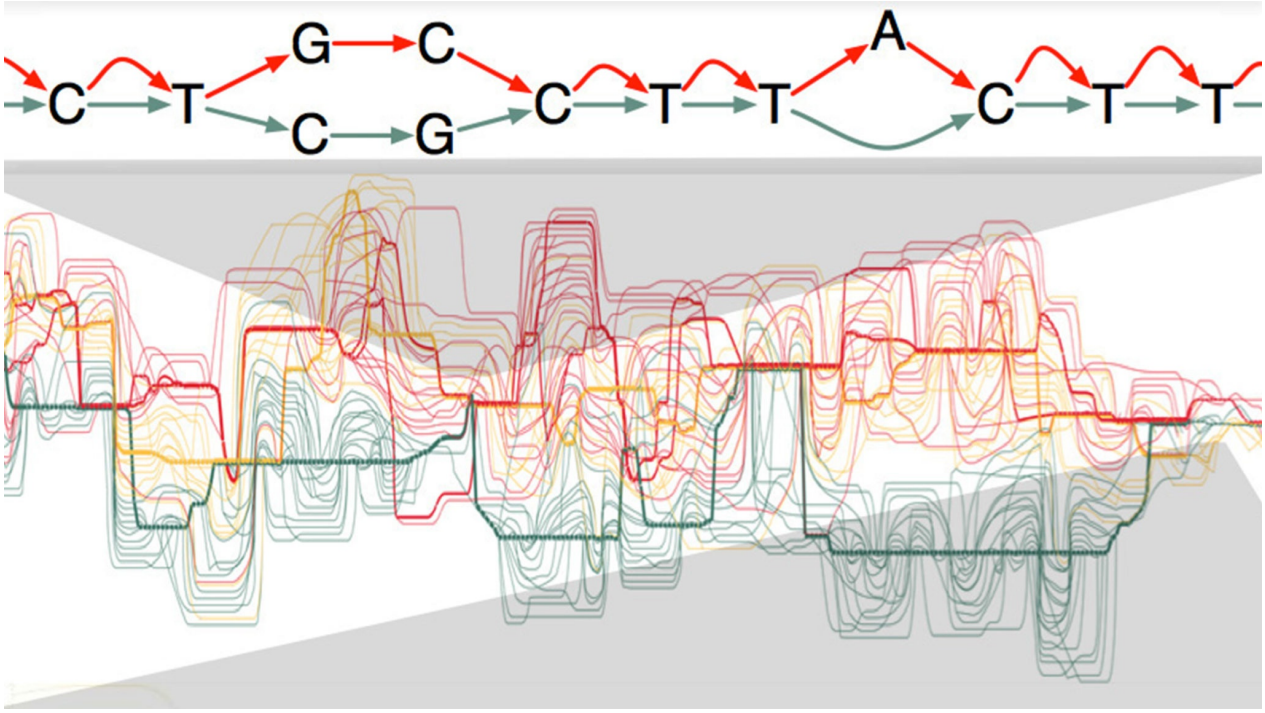
# What is the "white whale" of assembly?

In reference to Moby Dick, the "white whale" is something someone chases and chases then becomes obsessed with, but in the end, they can never get it.

The white whale of assembly is obtaining "single linear genome." What it means is that you hope that even when starting with the many millions of short reads you will end up with one, single, unabmigious, contiguous long sequence that corresponds to the "reference" genome that other scientists will now refer to in their work.

The reason that this is a white whale is that, there is no such thing as a single linear reference genome. The better we understand large-scale genomic variation, the more apparent it becomes that no single reference genome could characterize each member of the same species. Instead of a single genome the new school of thought is to approach the problem in terms of a network of genomes, and within that framework identify the path that describes our genome in this much larger graph space. Hence our goal should be to identify the segments and connectors in this graph space and worry less about the concept of single linear "reference". For a higher level overview we recommend the following article that describes the upcoming shift in the way we think about genomes:

- As DNA reveals its secrets, scientists are assembling a new picture of humanity




# What can be "assembled"?

- Genome assembly.
- Transcriptome assembly.

Applications to metagenomics:

- Meta-genome assembly.
- Meta-transcriptome assembly.

# Assembly concepts

## How do "assemblers" work?

The last decade has brought substantial innovation to this field, from so-called "overlap layout" algorithms that use multiple sequence alignments to k-mer and graph based algorithms that attempt to find the shortest path that connects shorter subsequences.

At the same time, different instrumentation requires radically different approaches - it is unlikely that a tool designed for Illumina data would perform well on reads produced by a PacBio instrument.

## What makes genome assembly difficult?

Since there are only four bases in the DNA and because genomes contain repetitive elements some regions of a longer sequence may allow for many different yet seemingly equally good alternatives in which these shorter segments could be pieced back together. In essence, there could be multiple, equally good solutions to the puzzle.

Assembly processes are more sensitive to the read lengths and the systematic errors of the sequencing process. Short reads may contain fewer identifiable regions. Systematic errors may accumulate and support spurious results.

> **Note**: Assembly as an application domain of bioinformatics is immature. The success of an assembly depends on (potentially hidden) attributes of the data.
>
> It is not uncommon for software to produce radically different outcomes when slightly modifying their settings and parameters. The algorithms require several levels of decision making, often with far-reaching consequences.

## Beyond technical limitations what else makes genome assembly difficult?

Sequencing measurements originating from non-identical genomic sequences cause a variety of problems. Diploid organism already contains two slightly different copies of DNA. Various members of the same species may exhibit even more divergence. These difference make the job of an assembly program even harder as it tries to reconcile and consolidate these differences into a single consensus sequence. Also, certain regions produce much lower coverages than others leading to uneven coverage.Collecting more data may not be sufficient to address systematic problems of this type.

## How good is good enough?

Let us make something very clear from the beginning. No genome assembly is *entirely correct*. Even to this day, vast tracts of the human genome remain unknown; different genomic builds and releases are used to publish ever more accurate genomes of even model organisms.

Your goal is always to define "how good is good enough." What quality of the assembly would you need to answer questions under study? That should be your driving motivation rather than producing a single, linear sequence that corresponds to a whole genome.

## Is data quality control more important?

Data consistency has a more profound effect on the process than for resequencing. In those cases, your processes have additional information that can assist you in making the right decisions. In contrast when assembling from scratch errors and mistakes compound and may lead to substantially lower performance.

## Does the sequencing platform matter?

Assembly as a field of science got jump started by the Illumina sequencers. Most of the published results, algorithms and evaluations refer to assembling data from Illumina instruments.

Long read technologies such as PacBio and MinION produce reads that are substantially longer. In a manner of speaking, they get a "head start" on assembly. At the same time since the error rate on these platforms are considerably higher, radically different algorithmic approaches are required to overlap and extend these measurements. "Classical" methods may not be appropriate at all.

## What is a hybrid assembly?

A hybrid assembly is a process that uses data from different platforms and tries to make use of the strengths of each. For example the Illumina platform provides higher coverage data whereas the PacBio platforms produce longer reads that span over longer intervals. Combining these data can significantly improve the assembly.

## Does more data lead to better assembly?

While intuitively we may assume that more data gives better chances to find overlaps in practice more data does not guarantee better assembly. The coverage of the genomes will eventually yield diminishing returns.

In general assembly, processes will perform better with long and paired-end reads. The data quality (having DNA from a single genome) and genome' complexity have the most pronounced effects. The less complex (unique) a region of a genome is the more difficult will be to assemble it.

## What does a genome assembly rely on?

The current experimental procedures for genome assembly demand that some of our measurements (reads) overlap with each other. The assembler attempts to detect the overlaps and will attempt to piece and extend the sequences by overlapping the identical (or similar) regions.

For example, suppose that the following were your "sequencing reads":

```
ATATCGAAGAA
GGAAATACATTTATTA
GTACAAACATA
AGAATAAGGAAAG
CATAGAAGGAGGAAA
AGGAAAGATGGCATTA
```

Then your assembler may find what it believes to be the most "optimal" way to overlap this data and may consider the following as the best arrangement:

```
ATATCGAAGAA                          GTACAAACATA
       AGAATAAGGAAAG                        CATAGAAGGAGGAAA
              AGGAAAGATGGCATTA                      GGAAATACATTTATTA
```

And from the overlaps above it forms two "contigs":

```
ATATCGAAGAATAAGGAAAGATGGCATTA         GTACAAACATAGAAGGAGGAAATACATTTATTA
```

Six reads have been assembled into two contigs. Important to be aware of the following limitations:

1. Contigs may get assembled on different strands.
2. Contigs do not indicate the order in which they present in the genome.

Then note that it instead of overlapping `1`, `4` and `6` our assembler could have overlapped just `1` and `6` like so:

```
ATATCGAAGAA
       AGGAAAGATGGCATTA
```

To produce a shorter and different contig:

```
ATATCGAAGAAAAGATGGCATTA
```

This contig was built at the cost of discarding read `2`. The series of choices that the "assembler" makes along the process may have substantial implications on the final result.

# Why are repetitive regions difficult to assemble?

Sequencing libraries are built by fragmenting the original DNA into smaller pieces. When regions are repetitive, it becomes difficult to tell if a repeated part come from the same or a different location. If your reads looked like so:

```
TROLLOLLOOTROLLOL
LOOTROLLOLLOTROLLOLL
ROLLOLLOTROLLOLLO
OLLOLLOOTROLLOLLOTROLLO
```

Each of the following contigs could be a valid assembly:

```
TROLLOLLOOTROLLOLLOOTROLO
```

or

```
ROLLOLLOOTROLLOLLOTROLLOLLOOTROLLOLLOOTROLLOLLO
```

or

```
OLLOLLOOTROLLOLLOOTROLLOLLOTROLLOLLOOTROLLOLLOOTROLLOLLO
```

As well as many alternatives. Note how the length of the read becomes essential in identifying the minimal "repeat" structure. But we can't tell the longer repeats. The read length has to be comparable to the repeat length to be able to resolve the repeats reliably.

# Re-analyze: GAGE: A critical evaluation of genome assemblies and assembly algorithms.

We are interested in accessing the data for the study titled

- GAGE: A critical evaluation of genome assemblies and assembly algorithms

The paper was published in 2012 in the Genome Research journal. Specifically, we will discuss the results for the sequence data deposited for the bacteria *Staphylococcus Aureus* known for its potential to cause various skin and soft tissue infections.

**Resource**

## GAGE: A critical evaluation of genome assemblies and assembly algorithms

Steven L. Salzberg,[1,7] Adam M. Phillippy,[2] Aleksey Zimin,[3] Daniela Puiu,[1] Tanja Magoc,[1] Sergey Koren,[2,4] Todd J. Treangen,[1] Michael C. Schatz,[5] Arthur L. Delcher,[6] Michael Roberts,[3] Guillaume Marçais,[3] Mihai Pop,[4] and James A. Yorke[3]

*[1]McKusick-Nathans Institute of Genetic Medicine, Johns Hopkins University School of Medicine, Baltimore, Maryland 21205, USA; [2]National Biodefense Analysis and Countermeasures Center, Battelle National Biodefense Institute, Frederick, Maryland 21702, USA; [3]Institute for Physical Sciences and Technology, University of Maryland, College Park, Maryland 20742, USA; [4]Center for Bioinformatics and Computational Biology, University of Maryland, College Park, Maryland 20742, USA; [5]Simons Center for Quantitative Biology, Cold Spring Harbor Laboratory, Cold Spring Harbor, New York 11724, USA; [6]Institute for Genome Sciences, University of Maryland School of Medicine, Baltimore, Maryland 21201, USA*

New sequencing technology has dramatically altered the landscape of whole-genome sequencing, allowing scientists to initiate numerous projects to decode the genomes of previously unsequenced organisms. The lowest-cost technology can generate deep coverage of most species, including mammals, in just a few days. The sequence data generated by one of these projects consist of millions or billions of short DNA sequences (reads) that range from 50 to 150 nt in length. These sequences must then be assembled de novo before most genome analyses can begin. Unfortunately, genome assembly remains a very difficult problem, made more difficult by shorter reads and unreliable long-range linking information. In this study, we evaluated several of the leading de novo assembly algorithms on four different short-read data sets, all generated by Illumina sequencers. Our results describe the relative performance of the different assemblers as well as other significant differences in assembly difficulty that appear to be inherent in the genomes themselves. Three overarching conclusions are apparent: first, that data quality, rather than the assembler itself, has a dramatic effect on the quality of an assembled genome; second, that the degree of contiguity of an assembly varies enormously among different assemblers and different genomes; and third, that the correctness of an assembly also varies widely and is not well correlated with statistics on contiguity. To enable others to replicate our results, all of our data and methods are freely available, as are all assemblers used in this study.

We do like this paper and have no doubt that the authors of it were (and probably still are) the most skilled "assemblers" on the planet. That being said we have to point out what we think are weaknesses of it. The entire paper suggests that assembly is a procedural task. They publish "recipes" and with that imply that you could download these recipes and run them on your data to get results of comparable quality.

Alas, that is not how assembly works. The recipes that they come up with match the "data" - they have most likely obtained these after a lengthy process that itself is not adequately discussed nor mentioned. We believe that hidden properties of the data are "baked into" the parameters themselves. We believe that is unlikely that these same recipes would produce optimal or even acceptable results on your data. Moreover, we want to note how there are no recipes for properly evaluating and correcting the assemblies - those steps are left mostly unexplained.

That being said we consider this paper as a recommended reading to anyone interested in deepening their understanding of this field.

## Where is the supporting material?

Commendably the authors of the publication maintain a website to distribute their results:

- http://gage.cbcb.umd.edu/index.html

## How does GAGE differ from the other assembly bake-offs?

Both the question and the answers are lifted right from their web page and are reproduced verbatim because we find them amusingly combative:

- "*GAGE is being run by assembly experts.* Our team has assembled hundreds of genomes and has written some of the leading genome assembly software. We have been evaluating assemblers for more than 10 years. All of the assemblies and the comparisons among them will be conducted by experts."
- "*All natural ingredients*. GAGE will use FOUR different whole-genome shotgun data sets, all from recent sequencing projects. Assemblathon and dnGASP will both use simulated data. Who can say how simulated data relates to real results? We prefer the real thing."
- "*Completely open protocols*. We will compare multiple genome assembly programs, and we will describe all the parameters used to run them. We will also explain all the steps we take in cleaning up the data (pre-processing) and scrubbing the output of the assembly programs (post-processing). All of our results will thus be reproducible by anyone with sufficient computing resources."

## How do I get the data for the paper?

As always any analysis reproducibility starts with obtaining the data itself. Helpfully the GAGE paper lists the accession numbers for the data. In this case, let's focus on *Staphylococcus aureus* datasets listed as shown below:

### Methods

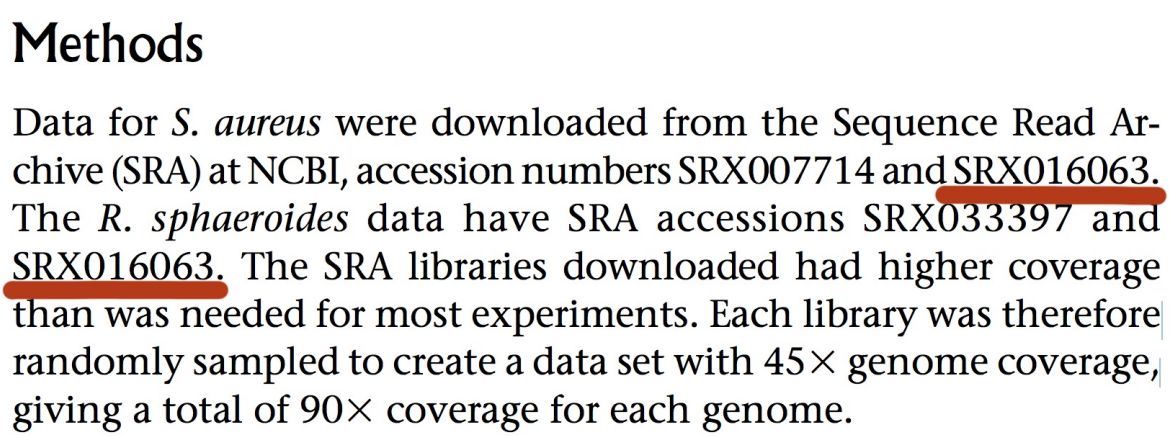
Data for *S. aureus* were downloaded from the Sequence Read Archive (SRA) at NCBI, accession numbers SRX007714 and SRX016063. The *R. sphaeroides* data have SRA accessions SRX033397 and SRX016063. The SRA libraries downloaded had higher coverage than was needed for most experiments. Each library was therefore randomly sampled to create a data set with 45× genome coverage, giving a total of 90× coverage for each genome.

What the paragraph above says is that the two data can be found at:

- https://www.ncbi.nlm.nih.gov/sra/?term=SRX007714 that corresponds to run id `SRR022868`
- https://www.ncbi.nlm.nih.gov/sra/?term=SRX016063 that corresponds to run id `SRR034528`

**Except the information above is not correct!**

There is a typo in the paper. The number `SRX016063` was entered for two entires, for two different organisms! See it for yourself:



## Methods

Data for *S. aureus* were downloaded from the Sequence Read Archive (SRA) at NCBI, accession numbers SRX007714 and SRX016063. The *R. sphaeroides* data have SRA accessions SRX033397 and SRX016063. The SRA libraries downloaded had higher coverage than was needed for most experiments. Each library was therefore randomly sampled to create a data set with 45× genome coverage, giving a total of 90× coverage for each genome.

Typos are nothing new - what is unique here is that the paper offers no other indication as to what the "correct" accession number is. Hence from the publication alone, it is impossible to identify what data has been analyzed!

Well, well, one more indication (if you needed more) on just how complicated data reproducibility is. Even authors who are acutely aware of its importance, who make a concerted effort to publish everything that they do and create detailed descriptions of their methods may commit errors that render the entire process *impossible from the very beginning*.

If anything this is a flaw of the old publishing process itself. Essential numbers should not need to be manually be embedded in a paper. Data should be cross-referenced, labeled automatically to make errors like this standout.

Did we find the correct SRR number? In this case, yes. Thanks to the author's foresight to create another, standalone site to distribute their data we were able to figure the accession number eventually. Another example of how bioinformatics requires problem-solving of unexpected nature.

Visit the GAGE data site http://gage.cbcb.umd.edu/data/

```
#
# Download the short jump file:
wget http://gage.cbcb.umd.edu/data/Staphylococcus_aureus/Data.original/shortjump_1.fastq.gz`

# Look at the first read name
gzcat shortjump_1.fastq.gz | head -1
```

It will print:

```
@SRR022865.7
```

From this SRR run number `SRR022865` we can find the real experiment id `SRX007711`:

- https://www.ncbi.nlm.nih.gov/sra/SRX007711

Success!

# Genome Assembly

## What are the steps of Genome Assembly?

The typical genome assembly process is composed of three different stages:

1. **Assembling contigs**: Merging reads into longer contigs.
2. **Scaffolding the contigs**: Spatially orienting contigs about one another.
3. **Finishing the genome**: Bridging over the spaces between the spatially oriented contigs and connecting them.

While possible it is very rare that a single assembly process would start with sequencing reads and would produce a finished genome. Perhaps if your coverage is high (over 50x), the data is of very high quality and the genome does not contain repetitive regions.

The steps above require increasing human intervention, especially when finishing genomes may need another laboratory protocol where specially targeted regions of the genome that are thought to be close to one another are re-sequenced with a low-throughput method that attempts to bridge the gaps.

## How to we quantify assembly quality?

What makes assembly processes difficult to evaluate is that we lack simple terms to quantify when is one result is superior to another. There is a shorthand notation called the 3Cs (CCC): contiguity, correctness, and completeness Ideally we expect assemblies to be:

1. Contiguity. Produce the longest possible contigs.
2. Correctness. Assemble contigs with few/no errors.
3. Completeness. Cover the entire original sequence and minimize missing regions.

Balancing these three requires trade-offs based on the requirement of the problem that is being solved. Expect to have to make these tradeoffs, and you should identify your priorities before you start the process: e.g. "I would rather be more accurate even if it means the assembly will be less complete."

Scientists working in the field have long been searching for various measures that would incorporate all the three Cs above. In practice measuring the first 'C' "contiguity" appears to be the dominant way to quantify assembly quality. And within that realm, a measure called the `N50` statistic is used as the primary differentiator. The N50 statistic was adopted as one that most closely captures the quality of the first step of assembly: extending reads to form longer contigs.

## What type of assembly errors will I get?

**Chaff** contigs that are comparable to a read size. These are indicative of dead ends in the assembly process.

**Misjoins**: joining two sequences that should not be together.

**Coverage misses**: regions of the genome that are not assembled at all.

**Repeat compression**: a common error that makes consecutive repeats appear as a single sequence. `ZZZABCDEABCDEABCDEABCDEABCDEZZZ` may look like as `ZZZABCDEZZZ`

# What is the N50 statistic?

Even though the N50 statistic is the most commonly used measure of the quality of a genome assembly, it might come as a surprise that its meaning is not rigorously defined. Also, no definition that we've seen appears to be quite right - think about that a bit.

The authors of GAGE: A critical evaluation of genome assemblies and assembly algorithms Genome Research (2012) states that:

> The N50 value is the size of the smallest contig (or scaffold) such that 50% of the *genome* is contained in contigs of size N50 or larger.

Let us note first that "GAGE" stands for "Genome Assembly Gold-Standard Evaluation." Calling something "gold standard" from the start is a common and annoying habit of some scientists. In our (though admittedly limited) experience papers declaring themselves as gold standard from the beginning are rarely that.

Then in the definition above you might immediately raise the question of how would you know what the *genome* size is when you are assembling it for the first time?. Now while there are experimental and comparative methods to estimating genome sizes, these are a different domain of science.

Many other scientists use different designations `N50` and `NG50` where `N50` refers to the sum of all contigs lengths of the assembly whereas `NG50` relates to using the length of the genome (if known or estimated). When the assembly length is reasonably close to the genome length the two measure are very similar. But note that there are legitimate reasons (not necessarily errors) that may cause an assembly to end up much longer or much shorter than the target genome.

Confusingly, several alternatives and slightly different definitions of N50 exist. Sometimes the median of contig lengths is used. For example in the Assemblathon 1: A competitive assessment of de novo short read assembly methods Genome Research, 2011 the authors overcomplicate the definition and state:

> The N50 of an assembly is a weighted median of the lengths of the sequences it contains, equal to the length of the longest sequence `s`, such that the sum of the lengths of sequences greater than or equal in length to s is greater than or equal to half the length of the genome being assembled.

I don't understand this definition - especially when it comes to weighted medians. I guess the authors agreed with this sentiment since in the Assemblathon 2: evaluating de novo methods of genome assembly in three vertebrate species Genome Research, 2013 the authors write:

> N50 is calculated by summing all sequence lengths, starting with the longest, and observing the length that takes the sum length past 50% of the total assembly length.

This definition is almost right, except it does not explicitly state that the summing should proceed by sorted lengths. It is noteworthy to observe just how difficult is to state even the definition for `N50` perhaps foreshadowing that this domain of application is far more complicated than bioinformaticians care to admit.

# So what is the N50 statistic?

A simpler explanation of N50, (in our own interpretation, that might not also be perfect), starts by ordering contigs by length. Suppose we have 10 different contigs (designated by `xxxxxx` ) and we ordered these by their decreasing sizes:

```
Contig         Length           Sum
XXXXXXXXXX       10             10
XXXXXXXXX         9             19
XXXXXXXX          8             27
XXXXXXX           7             34
XXXXXX            6             40
XXXXX             5             45
XXXX              4             49
XXX               3             52
XX                2             54
X                 1             55
```

The sum of these lengths starting with the longest is `55` . Half of that is `27.5` .

Go down on this list and add up the lengths to find the contig where the cumulative length exceeds this half value. When we hit contig number `7` we have `10 + 9 + 8 + 7 = 34` , this value is larger than `27.5` so it is at this point at least half of the genome is stored in contigs of size `7` or greater.

Our N50 is then `7` . Another simple way to state this (following the wording of this blogpost) is that

> "At least half of the nucleotides in this assembly belong to contigs of size 8bp or longer."

# What are the problems with N50?

One of the biggest problems using the N50 metric as the primary means of evaluating assembly quality is that it rewards a type of grave error "the misjoins." A "misjoin" is the error of concatenating separate contigs into a single segment. Such errors can happen when the evidence is insufficient, but the algorithm is tuned to be overly generous in accepting these pieces of evidence. Keeping two, related parts in separate contigs is, as a matter of fact, a far less troubling error than misjoining them.

Also what seems to questionable above is how to deal with situations when selecting contigs would lead us to be either well under or well over the 50% mark. For example, if the genome is 1 million long and we have one contig that is 499,999bp long, and the rest are 1bp long - does it make sense to call the `N50` to be 1bp? Half of the genome is assembled!

# So what is your N50 now?

While building artificial corner cases to demonstrate a weakness of a method or measure is akin to building straw men we could not resist it this time around. We invite you to do the math below and with that demonstrate to yourself that you do indeed understand how the N50 computation works.

Imagine that you have a genome of a size 1 million bp with and you ran an assembler that created one large contig of the length of `500,000bp` plus another 500,000 contigs of sizes of `1bp` each.

```
Contig                                      Length
XXXXXXXXXXXXXXXXXXX....XXXXXXXXXXXX       500,000
X                                               1
X                                               1
X                                               1
X                                               1
X                                               1
... 500,000 1bp long sequences ....
```

- What is your N50?

Now take the same example and make your largest contig just a single basepair shorter `499,999bp`:

```
Contig                                      Length
XXXXXXXXXXXXXXXXXXX....XXXXXXXXXXXX       499,999
X                                               1
X                                               1
X                                               1
X                                               1
X                                               1
... 500,000 1bp long sequences ....
```

- So what is your N50 now?

Well, what do you think?

# How to improve the assembly process

When you start out in the field of assembly you will note with surprise how much of the advice can be summarized as:

> Keep running your assembler with many different parameters until you get a proper assembly.

There is a "handwaving, " and almost "unscientific" feel to the recommended processes. Should you ever tweak and run any tool until it gives you what you want? Also, you will get somewhat ad-hoc guidance as to what a "good" assembly even means.

# Why are assemblers so sensitive to parameters?

We believe that assemblers are sensitive to parameters because we as scientists are not quite sure when they work correctly - hence the rationale is to allow for extensive customization. Then, the information content and structure of sequences will vary to a great extent, both within a genome and more across species. This variation poses serious challenges to any algorithmic approach - as principles and optimizations that work well in one region will not work for others.

Also 's hard to identify with clarity how the assembler works internally and what decision it makes - hence it hard to foresee the effect of each parameter value on the end result.Thus we are left with trial and error.

# Should I error correct my data?

As you recall error correction means correcting for sequencing errors using the data itself. There are many logical explanations why this should work. For example, error corrections will significantly reduce the number of k-mers in the data - and as such will make k-mer based algorithms job easier.

At the same time error correction is a new application domain that we have not had sufficient experience with. Later examples in this book may explore the effect of error correction and it, in general, it should be a choice that you evaluate yourself. Many authors, with experience believe that error correction is a requirement.

# What are the k-mer sizes?

Refer to the chapter title Sequence K-Mers for a definition of k-mers. In nutshell, a k-mer are all possible subsequences of size `k`. The rationale for breaking our reads into even smaller pieces, `k-mers` is that we want to identify "correct" k-mers, those that originate from the real data. The assumption is that whatever errors the reads may have these are distributed randomly - hence will produce different erroneous k-mers. The correct k-mers will always be the most abundant.

In general, it is true that the longer a k-mer is, the fewer identical k-mers to it exist. At the same time it is also true the longer a k-mer is the more likely is that it will contain an error. When it comes to assembly processes the rule of reasoning is that:

- A larger `k` value allows resolving more repetitions.
- A smaller `k` increases the chances of seeing a given k-mer.

Hence the selection of a `k-mer` is a tradeoff between longer repeats and more reliable measures. In general, you should assemble sequences using the largest k-mer size possible, such that the k-mer coverage is sufficient.

You may estimate these via trial and error (as stated above) but there are also tools like `kmergenie` that will assist you in determining the most likely k-mer sizes and coverage cutoffs. These tools sometimes work well, other times no so much.

# How to tune the expected coverages?

From experience, we found that parameters indicating the "expected coverage" have a pronounce and substantial effect on the resulting assembly. This parameter may take different names for different tools:

```
-abundance-min
-cov_cutoff
```

each indicates the minimal number of observations necessary to "trust" that a k-mer sequence is correct. Again it is never clear what the value should be from the start. Some tools allow for an automatic inference.

# The lucky bioinformatician's guide to genome assembly

Assembling a genome is a challenging process. Except when you get lucky. Then "it just works". But even then you need a little more than luck. You have to make your own luck.

In this section we will be using the Minia assembler to demonstrate the typical usage patterns during an assembly process. We like `minia` because it is easy to run, is super fast and has other tools that integrate with it.

## How do I run an assembly?

Genome assembly is an iterative process where you will:

1. Apply quality control measures
2. Estimate initial parameters (k-mers-sizes, coverages)
3. Run the assembly
4. Evaluate assembly (then go to 2 if necessary)

## Is quality control necessary?

Yes. Absolutely. Every single guide will tell you so. In fact, quality control for assembly is a more complicated process. You will need to correct not just "visible" errors that the sequencing instrument marks with its quality values but even the invisible ones that need to be fixed with the most sophisticated methods the human mind has ever invented.

Unless you get lucky. In which case you don't need to correct data at all. You "just carry on" and it will "just work".

## Do I need a lot of data?

Yes. Absolutely. Not only that, the consensus is that you will need the longest reads the instrument can generate, preferably 250bp or longer. Plus you'll need paired-end reads that help scaffold the resulting contigs. Also, you better collect more data than you think you need.

Unless of course, you are lucky. In which case you may be able to get by using a small fraction of the sequenced data, say a hundred thousand, single end, short reads.

## What does it look like when I am getting lucky?

Remember the study that we mentioned in the Redo section of the main book:

- Genomic surveillance elucidates Ebola virus origin and transmission during the 2014 outbreak

published in 2014 in the Science research journal. Let's obtain a sequencing run from it. It has hundreds of sequencing runs, Hmmm, where to even start... well, I don't know ... well let's pick a run ... how about run id SRR1553425:

Get the data, it has 1.8 million paired-end reads (3.6 million total reads) but for now, for the sake of this example let's take a subset of 100,000 reads. In fact, we did not even expect it to work, we only want to demonstrate the process:

```
fastq-dump SRR1553425 -X 100000 --split-files
```

The data is paired-end, but for, and for the sake of simplicity we'll focus on first pairs, ignoring that we have extra information and even more data in the other pairs. Notably, when we are taking 100K reads from a single file, hence we are using just 3% (!) of the total sequencing data:

```
READS=SRR1553425_1.fastq
```

Off we go with the assembly. Let's run `minia` with no other parameters set:

```
minia -in $READS -out genome
```

First of all, hats off to Dr. Rayan Chikhi. The process took only 5 seconds on a iMac! It is mind boggling that it only takes 5 seconds to build an internal graph structure out of millions of k-mers, that were spread over 10 million basepairs of 100,000 reads, then to churn through this network with a graph algorithm to find the shortest paths in them. We can unequivocally state that bioinformatics software developers are the best the world has ever seen!

The results of the process are stored in a file called `genome.contigs.fa` that we can run a quick stats on:

```
seqkit stat genome.contigs.fa
```

to produce:

```
file              format  type  num_seqs  sum_len  min_len  avg_len  max_len
genome.contigs.fa  FASTA   DNA     190     16,812      63     88.5      277
```

Okay, so we obtained:

- `190` contigs
- the average length is `88.5bp`
- the longest is `270bp`

This result seems useless, all *chaff*. Our read lengths were `101` long. How does one even "assemble" `101` long reads into an average of `88.5` ? ... Upon closer inspection, the numbers are more encouraging. See the `sum_leng` field? It says `16,812` . We do know that the Ebola genomes are of `18,000` bp size.

Now consider that we started with `100,000` reads where each read was `101bp` long. So the assembler was able to collapse `10,000,000` sequenced bases into a total length of `16,000` ! This assembly tells us that the algorithm has a good "sense" of what the individual pieces are, but it has a hard time connecting these parts. Things may not be all that grim.

# How do I get "luckier"?

You saw how the assembly above looked "wrong" from contig length perspective but promising when evaluating the total genome length. This information tells you it might be worth poking around a little more, perhaps rerun the assembly with other parameters, such as different k-mers. When the assembly runs in seconds like the example above this is feasible. For now, we continue, hoping to get even more "lucky."

The default k-mer size is `31` and we mentioned before how usually we want to raise this as high as possible, but still keeping sufficient coverages. Let's go higher to `100` .

```
minia -in $READS -out genome -kmer-size 100
```

Now the assembly reports a total assembly size of `233bp` - clearly well off the mark.

When you are searching for a parameter range you shouldn't change it small steps. First find the limits, then perform what is called a "binary-search" in between. Take the half between the explored intervals and chose the direction towards the better results. We saw how 31 seemed ok, but 100 was not. Half between is about 75

```
minia -in $READS -out genome -kmer-size 75
```

this improved our assembly quite a bit:

```
file              format  type  num_seqs  sum_len  min_len  avg_len  max_len
genome.contigs.fa  FASTA   DNA     21      19,896     156      947.4   3,539
```

We got `19` contigs, the longest is `3.5Kb` . Sounds good. Let's keep going higher with the k-mer sizes:

```
minia -in $READS -out genome -kmer-size 90
```

What? No way!

```
file              format  type  num_seqs  sum_len  min_len  avg_len  max_len
genome.contigs.fa  FASTA   DNA          1   18,820   18,820   18,820   18,820
```

Minia has now assembled our data into a single sequence. Do you recall that for `-kmer-size 100` the assembly was practically useless? And for `-kmer-size 90` we get complete genome in a single contig. Is this is skill or luck?

# How do I evaluate the assembly?

Let's align the assembled contig against the Zaire Ebola build of 1976.

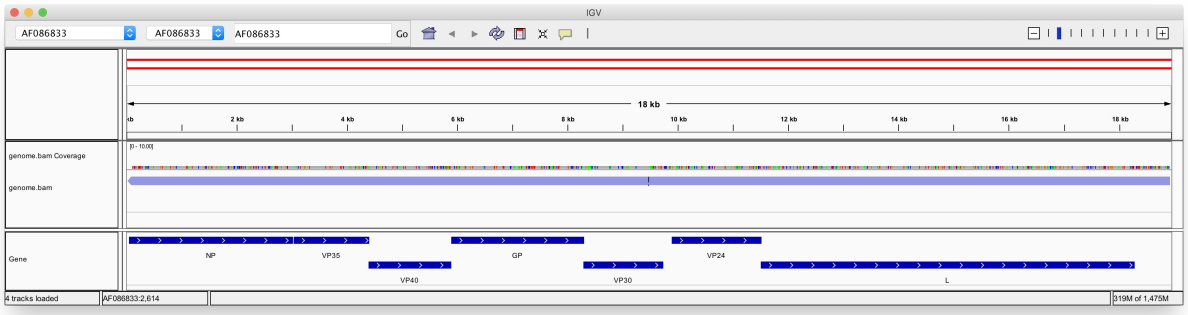
```
# Make a folder for the references.
mkdir -p refs

# Accesion number for the 1976 outbreak.
ACC=AF086833

# This is the reference name.
REF=refs/${ACC}.fa

# Get the reference sequence.
efetch -db=nuccore -format=fasta -id=$ACC | seqret -filter -sid $ACC > $REF

# Index reference for the aligner.
bwa index $REF
```

# What do I see in this assembly that I wouldn't otherwise?

Then align the assembled contigs against the 1976 strain:

```
CONTIGS=genome.contigs.fa
bwa mem $REF $CONTIGS | samtools sort > genome.bam
samtools index genome.bam
```

Visualizing the resulting `genome.bam` file in IGV we would see:




Note how the alignment is blue and not red. This alignment indicates that we have assembled the reverse strand! Since our data comes from a single stranded RNA virus that was transcribed into DNA for the purpose of sequencing you would need to reverse complement your assembly to find the real viral sequence.

```
cat genome.contigs.fa | seqkit seq -p -r > ebola.fa
```

Now align this reverse complemented sequence. It now aligns in the proper direction. When zoomed in you will see that genomic variants show up directly from the alignments and no SNP calling is necessary.

So there. You can produce and almost complete assembly and find all variants in an Ebola genome in four tries, totaling about 20 seconds. And you can do that by using only 3% of the collected measurements, using paired end reads in single end mode, without any error correction. Well this is how good is to be lucky.

# How do I get lucky?

The history of this chapter is rather amusing. We did not cherry pick this example out of the many hundreds that come with this publications. We randomly picked an SRR run and geared up to demonstrate the challenges and frustrations that one feels when attempting to assemble even a short and straightforward genome.

Much to our surprise the assembly "just worked." It also helped crystallize in our mind a long-standing suspicion that we've always had:

> Assembly is all about the data and not the method/protocol.
>
> Clean and well-defined data can be easily assembled.
>
> "Mosaic" data will give your troubles. What is "mosaic" data?
>
> - data obtained from the superposition of multiple genomes or
> - data containing regions with high self-similarity
>
> High quality, simple data does not need much filtering or effort. Difficult data may be impossible to assemble. It is very unfair!

You get lucky by making your own luck. Choose the problem wisely, recognize problems early, then don't blame yourself if the assembly does not work.

# How to perform a genome assembly

## How do I run an assembly?

Genome assembly is an iterative process where you will:

1. Apply quality control measures
2. Estimate initial parameters (k-mers-sizes, coverages)
3. Run the assembly
4. Evaluate assembly (then go to 2 if necessary)

## How to evaluate a genome assembly

The term "assembly evaluation" has two distinct and unrelated meanings:

1. Characterize the assembled contigs by their observed properties.
2. Characterize the assembled contigs relative to the "reality," typically a known, similar genome.

The two definitions often get confounded into a single term.

## What are the steps of genome assembly?

This section will work through genome assembly in the context of the assembly presented in the paper: GAGE: A critical evaluation of genome assemblies and assembly algorithms Genome Research (2012). We start with the sequence data deposited for the bacteria Staphylococcus aureus known for its potential to cause various skin and soft tissue infections.

```
# Make a directory for the reference.
mkdir -p refs

# Accession number for the 1976 outbreak.
ACC=NC_010079

# This is the reference name.
REF=refs/${ACC}.fa

# Get the reference sequence.
efetch -db=nuccore -format=fasta -id=$ACC | seqret -filter -sid $ACC > $REF
```

The data for a sequencing run for this bacteria is deposited in the Bioproject: PRJNA40073 The project describes 56 sequencing runs, out of which we start with `SRR022868` as the GAGE paper does. This sequencing run produces reads of length 101.

```
SRA=SRR022868
```

The genome of this bacteria is known and deposited in GenBank under the accession number `NC_010079`. Its size is reported to be 2.8 million. Now as you recall the coverage `C` of a genome is:

```
C = N * L / G
```

Where `N` are the number of reads, `L` is the length of a read, and `G` is the size of the genome. As we are interested in the number of reads of length 101 that

```
N = C * G / L
```

If we were aiming for say a 45x coverage, then for an L-101 we would need around 1.2 million reads. Since the data is paired end, we need to select half as many reads. Let's obtain those data:

```
# Make a directory to hold the original data
mkdir -p data
```

We could extract the reads.

```
SRA=SRR022868
fastq-dump $SRA -X 600000 --split-files -O data
```

When a sequencing run is affected by many errors (as this one is, as we shall see) it is inadvisable to build a smaller dataset by taking the first `N` reads of the file, as we did in other examples. Errors may manifest themselves more systematically with respect of the a physical layout of the flowcell. The order of the reads matches the flowcell hence the first `N` reads are not a proper statistical sample. The proper way to randomly sample the data would be to get all the data:

```
SRA=SRR022868
fastq-dump $SRA --split-files -O data
```

Then randomly extract reads from each dataset.

```
seqkit sample -2 -n 600000 data/SRR022868_1.fastq  > data/F1.fq
seqkit sample -2 -n 600000 data/SRR022868_2.fastq  > data/F2.fq
```

Set up variables to make the commands easier to rerun on other input:

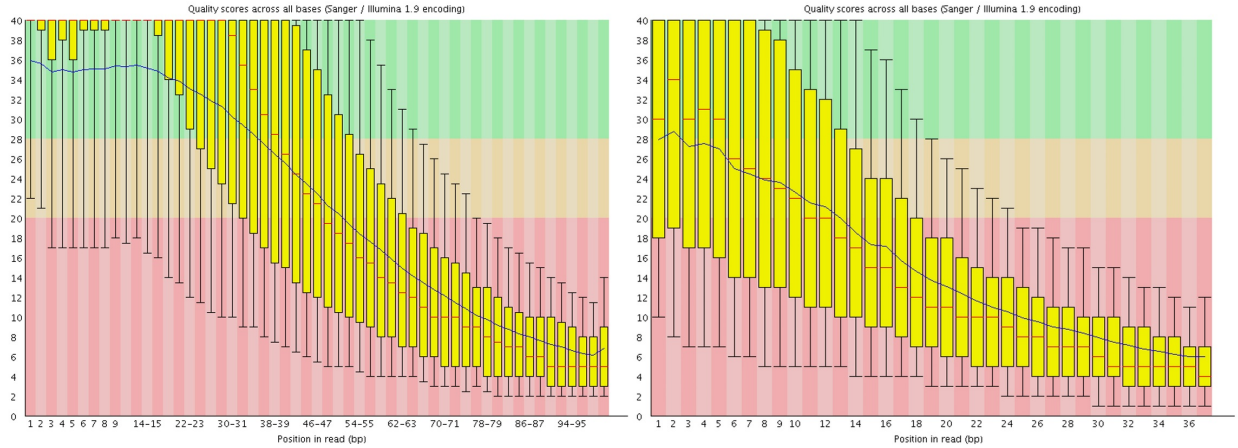
```
READ1=data/F1.fq
READ2=data/F2.fq
```

The paper states that the dataset has an insert size of `180` with a standard deviation of `20`. If we did not know this we could find out this information by aligning the sequences then computing the average template length. Below we filter (S=see the SAM filtering chapter)for paired, properly mapped reads on the forward strand.

```
bwa index $REF
bwa mem $REF $READ1 $READ2 | samtools view -F 4 -f 3 -F 16 | datamash mean 9
```

This command gives us `193` as the fragment (insert size).

The FastQC report shows data of surprisingly very low quality (Left is `SRR022868`, the right plot is for run `SRR022865`).




Looking at this data our first instinct is that the data is useless. In reality that is not the case. As scientists, we are trained to trust measurements and instruments, and we give credibility to quality values. The more I learn about the sequencing instruments, the more I come to believe that most FASTQ sequencing qualities are wildly inaccurate. Moreover the data above cannot be corrected in the "traditional" way. Try it, and you will see why. Filtering for say and average quality of 30 would remove just about all the data. It looks like a total loss. In fact, it is not.

First let's see what can we do if we took the data as is with if we were to put it through an easy to run assembler like Minia.

```
# Concatenate both paired file into one.
cat data/F1.fq data/F2.fq > data/minia.fq

# Assembly with Minia.
minia -in data/minia.fq -out minia
```

Minia completes in about 27 seconds but it is not a "pair-end aware" assembler, and with that we are losing information. It does finish though and the again, just as in the previous example to total assembled genome size is surprisingly accurate.

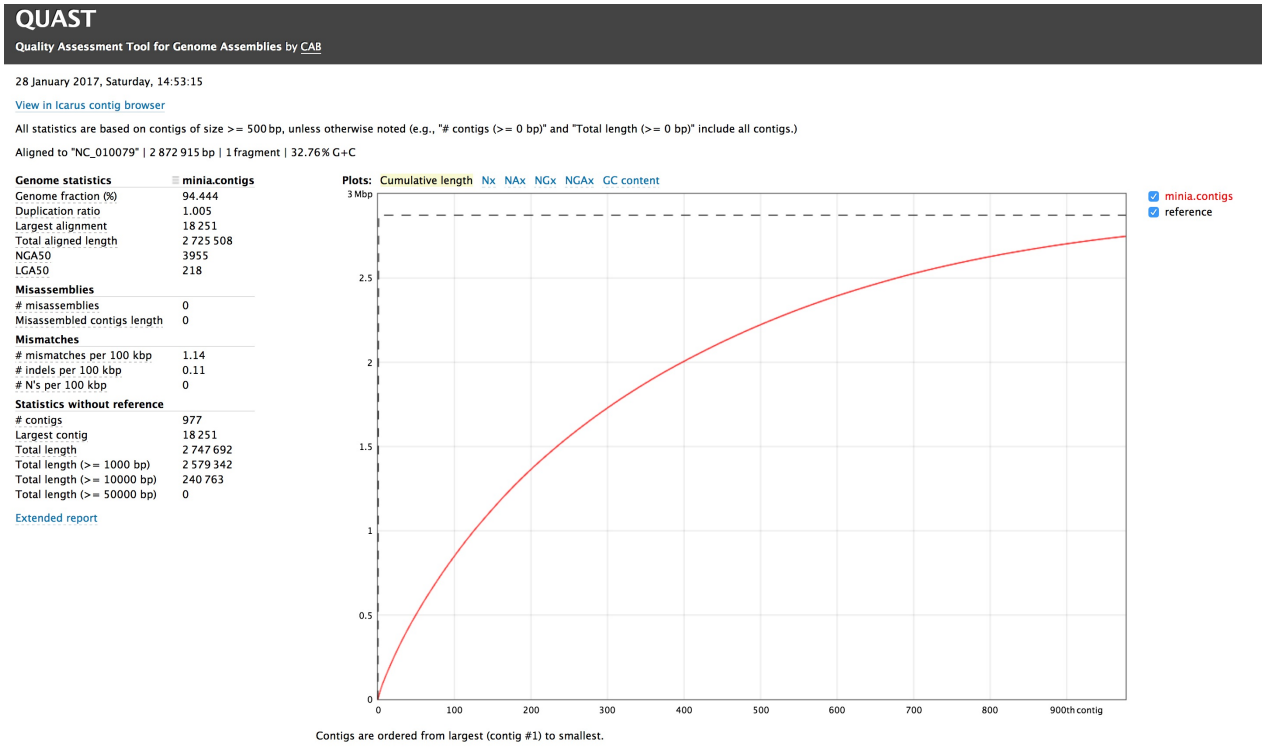
# How do I visualize a resulting assembly?

Thanks to the leadership of Pavel Pevzner The Center for Algorithmic Biotechnology St. Petersburg State University in Russia has become a powerhouse of assembly software. Among the many tools they have created, QUAST, Quality Assessment Tool for Genome Assemblies stands out as one that produces one of the simplest reports that characterize an assembly.

Using `quast` is quite straightforward, it requires a reference and a file with contings:

```
quast -R $REF minia.contigs.fa
```

The result is an interactive web page that visualizes the assembly that you have created.




# Can I improve the assembly?

Let's give a different assembler a chance. A perennial favorite, `velvet` is run in two steps with two subcommands. First, you will need to compute subsequences of a given k-mer (31) size and as show below store them in a folder, called here `velvet31`:

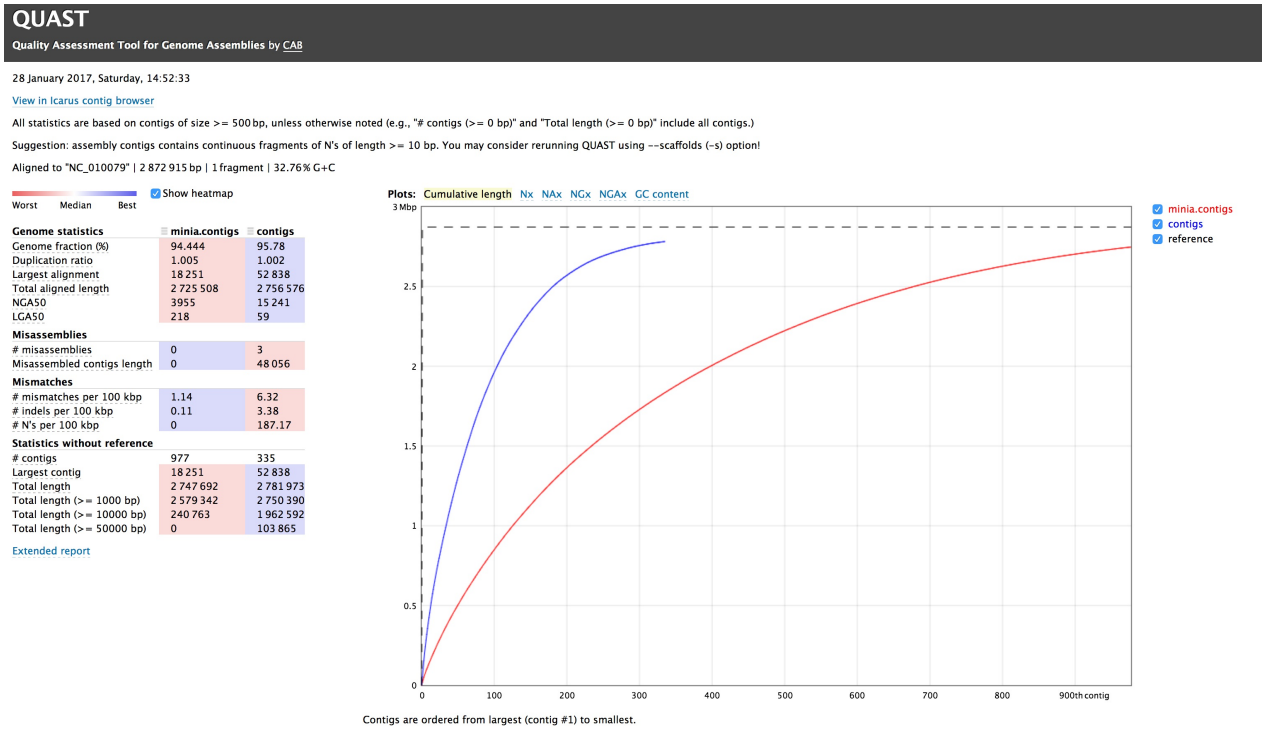
```
velveth velvet31 31 -fastq -separate -shortPaired $READ1 $READ2
```

You can use the `velvetg` subcommand to assemble the genome from these precomputed data:

```
velvetg velvet31 -exp_cov auto -ins_length 190
```

Again the process is very speedy, taking no more than about 2 minutes. The resulting assembly will be located in the file out31 called `contigs.fa`. We can now plot both assemblies on the same QUAST report.

```
quast -R $REF minia.contigs.fa velvet31/contigs.fa
```



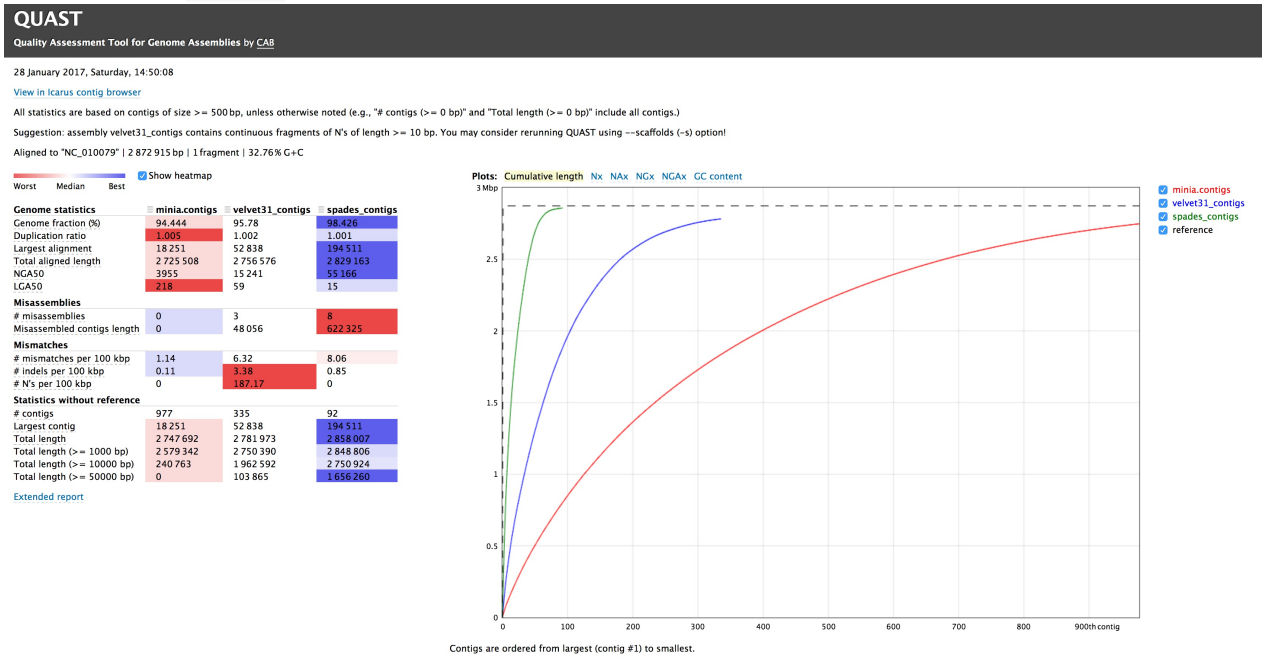

The resulting image shows the new assembly is shown as a blue line. Note how it rises much quicker to 100%, hence it contains longer segments.

# What assembler should I use?

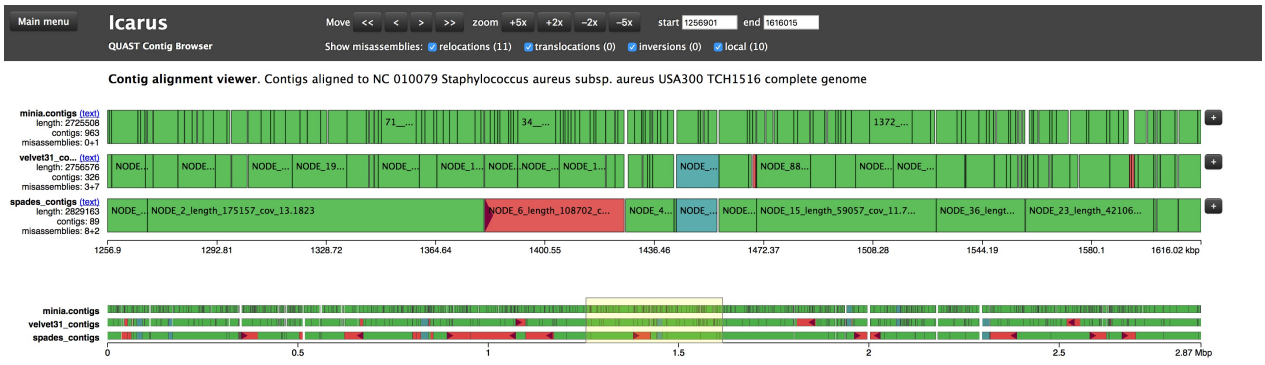
There are many choices you can choose from. See the GAGE paper for more options and inspiration if you need to. We like `velvet` because it is easy to run and allows you to set a baseline for your assembly. One of our other favorite assemblers is `spades`, another tool made at The Center for Algorithmic Biotechnology.

```
spades.py -1 $READ1 -2 $READ2 -o spades
```

Over the years `spades` hs




The assembly with `spades` appears the best by large, yet that information alone may not tell you the whole story. QUAST comes with the so called Icarus contig browser, another visual aid that helps assess how well the assembly performs:




Better than any other explanation this browser shows you the tradeoffs. The Minia assembly is built out of many more pieces, and none are incorrect or in the wrong orientation, and it does cover the genome correctly. On the other end, the spades assembly contains the largest of the pieces - but it also includes misjoins, large misassembled regions. The velvet assembler produces output that is in between them both.

# What do I do next?

At this point is where the "white whale" chase starts in earnest. It is easy to get excited and start tuning and running different methods. Progress is initially quick and fun, but then it can drag on. You can error correct, filter the data, make use of extra information, continuously evaluate the effects of each.

You may be able to use the reference genome to provide additional information to the assembler, Other data, preferably from a different sequencing platform helps even more.

Slowly you can build out better and better assemblies, stumble upon parameter setting, read up on other experiences, find and tune your results. It won't be pretty, and it won't be quick. There won't be a recipe that you could make use of right away the next time around - but you will do the second assembly much better as you have learned what works and what does not for your data.

The GAGE paper lists many other choices and strategies.

# Installing assembly tools

This book assumes that you have followed the instructions in the Biostar Handbook and your system is set up with `homebrew` and/or `conda`

Most tools can be installed directly:

```
brew install velvet
```

or

```
conda install velvet
```

# Linux specifics

There is no `conda` installation for Minia yet. Visit the website and download the linux binary and place it in your execution path (say `~/bin` ).

Then when it comes to `spades` and `quast` you may need to create separate `conda` environments to run them. This is because there are two version of Python available and only one version may be active at a given time. An conda environment allows you to switch between the various setips..

# How do I use conda environments?

The "environment" means an isolated setup that is independent of other tools that you may have installed. Here is how to set that up.

First create a conda environment. You may name it any way you want it, here we call it `assembly` .

```
conda create --name assembly python=2.7
```

now activate this environment:

```
source activate assembly
```

Now you can install `spades` and `quast` into it.

```
conda install spades
conda install quast
```

# ChIP-Seq Analysis

## What is ChIP-Seq?

ChIP-Seq stands for chromatin immunoprecipitation followed by sequencing. In a nutshell, the process consists of a laboratory protocol (abbreviated as ChIP) by the end of which the full DNA content of a cell is reduced to a much smaller subset of it. This subset of DNA is then sequenced (abbreviated as Seq) and is mapped against a known reference genome. Since you are now obtaining data from a potentially variable subset of the entire genome, it poses unique challenges to data interpretation, hence a whole subfield of data analytics has grown out around it.

Whereas the name appears to imply that the analysis methods "require" that the data was obtained via chromatin immunoprecipitation (ChIP) this is not necessarily so. Many other laboratory protocols where the net effect is that only a subset of the total DNA is selected for can be processed with ChIP-Seq methods.

## What are the challenges of ChIP-Seq?

Perhaps one of the greatest problems of this entire subfield is the insufficiently well-defined terminology. See for example the section "**What the heck is a peak**" later on. Also, more so than in other analysis methods, there a "subjectivity creep." Seemingly simple decisions are made all the time - that in the end seem to accumulate and slowly grow out of control as the analysis proceeds.

Examples of these insufficiently defined terms are for example the concepts of "upstream" and "downstream" regions which typically refer to the genomic intervals that lie ahead of and after a particular genomic coordinate.

When referring to these ranges, you will need to associate a finite length to them: how far out do you still call a region to be "upstream." The choice of this distance may, in turn, significantly alter the outcome of the analysis. You will likely obtain different results when the upstream region is defined to be 500bp, 1000bp or 5000 kb long.

Further complicating the picture is that different genomes have varying complexities, in some genome the genes are densely packed in other vast tracts of seemingly nonfunctional regions lie between functional regions. Hence there is no "objective" measure by which a "reliable" length for an upstream or downstream region could be determined from.

## What the heck is a peak?

You will see how ChIP-Seq analysis is all about peaks - a term, that in our opinion is adopted widely yet lacks any reliable definition. Somewhat ironically speaking, perhaps the best definition of a peak is the following:

> A "peak" is what a "peak-finder" finds.

What makes the problem complicated is that we all think that we know what a "peak" is (or should be). A peak is a "shape" that rises sharply to a maximum then quickly drops back to zero. In that ideal case then the peak consist of the **coordinate** of the maxima on the genome (an integer number) and a **peak height**.

As it happens, in reality, most of your signals may not follow such well-behaved patterns - the peaks may be a lot broader,

Now if the peak is broad you may need to compute the **peak width** the range where there is a signal. Then once you have a wider peak representing it with its center may be counterproductive as it is not clear that the peak will be symmetrical around it. Instead, you'll have to keep track of the **right** and **left** borders of it.

Just having a signal, of course, is insufficient, we may want to ensure that the signal rises **above the background** of the noise levels. Also, most studies are interested in how cells respond to stimuli that in turn will alter the signal. So the changes may occur by the peak rising in a **different location** or via **changed peak height** often denoted with the term **differential peak expression**.

You can see how a seemingly innocuous word: peak may require managing more information than you'd expect.

# How are RNA-Seq studies different from ChIP-Seq studies?

Whereas both approaches appear to operate under similar constraints and count reads over intervals, there are several important differences:

1. The DNA fragments coming from a ChIP-Seq study are much shorter than a transcript studied in RNA-Seq analysis. Instead of counting measurements distributed over transcripts a single read may cover most of the fragment (or may even be longer than the original DNA fragment)
2. The DNA fragments of a ChIP-Seq experiment are less localized than transcripts. The cell machinery accurately locates and transcribes DNA always starting from the same start locations. The ChIP-Seq protocol is a whole lot more inaccurate. You need apply peak-calling to establish the origin of the fragments from incorrect data.
3. The ChIP-Seq data produces a larger number of false positives. In a typical RNA-Seq experiment, only transcripts are isolated, and we can be fairly sure that a DNA sequence that we observe will match a transcribed region of the DNA. In contrast, ChIP-Seq protocols strongly depend on the properties of the reagents and active ingredients used and these properties will vary across different proteins.

As a matter of fact, we need to compare to control (baseline) to verify that the protocol worked at all. Whereas in RNA-Seq we can immediately see the success of it by having reads cover transcript in ChIP-Seq study no such certainty exists. Also when comparing peaks across conditions each peak has to first pass the comparison to a background, and then only those peaks that pass the first stage are compared to one another in a second stage. Since each statistical test is fraught with uncertainties differential peak expression studies will often exhibit compounding errors.

# What will the data look like?

For the record let's enumerate the possible observations for a given genomic location.

1. No signal of any kind.
2. A signal that shows no peak structure and appears to be noise.
3. A "peak like" signal that is reported to be under a background level.
4. A "peak like" signal that is reported to be over a background level.
5. A "peak like" signal that exhibits differential expression across samples.

Many (most) peak finders will start reporting data only from level 4 on - leaving you with many head-scratching moments of the form: "Why is this perfectly looking peak not reported?". It all makes sense once you recognize that "peak finding" as the field currently defines it combines finding the peaks and comparing them to a background. In addition the statistical methods employed to determine these background levels have very high levels of uncertainty.

# What does a "peak" represent?

Most ChIP-Seq guides start with explaining how ChIP-seq works, and how it allows us to identify where proteins where bound to DNA. We believe that is far more important to understand what your data stands for in a simpler way first, then extend that to the laboratory process.

Imagine that we have DNA of from a cell that looks like this:

```
AGTGATACCGGTCTAAGCCTCGCCTCTTCCACAGGGGAAACTAGGTGGCC
```

Now let's say some locations are "covered," say "protected" from an interaction. Let's visualize those locations via the `=` symbols.

```
         =========               ==========
AGTGATACCGGTCTAAGCCTCGCCTCTTCCACAGGGGAAACTAGGTGGCC
         =========               ==========
```

Suppose that can wash away all unprotected regions. If we did that then sequenced the remaining DNA we'd be left with:

```
1 GGTCTAAGC
1 GGGGAAACTA
```

The first number indicates how many of those sequences were found. So far so good. Now let's take another cell from the same population. Suppose that this is also now covered, but in only one location:

```
         =========
AGTGATACCGGTCTAAGCCTCGCCTCTTCCACAGGGGAAACTAGGTGGCC
         =========
```

After washing away the uncovered regions we add the outcome to the prior results, and we have:

```
2 GGTCTAAGC
1 GGGGAAACTA
```

Finally, let's assume we have DNA from a third cell, this one is covered in an another location.

```
                                    ==========
AGTGATACCGGTCTAAGCCTCGCCTCTTCCACAGGGGAAACTAGGTGGCC
                                    ==========
```

So our cumulative data that we obtain after washing away unprotected regions from three cells would be.

```
2 GGTCTAAGC
2 GGGGAAACTA
```

In a ChIP-Seq experiment, instead of sequencing the entire genomes from three different cells we only sequence relatively short pieces of DNA (in reality the sequences are longer than these example sequences above but still radically shorter than say a transrcript). The alignments that we generate will pass through a "peak finding" program that will attempt to reconstruct which locations had sequences mapping to and how many from each of the small sequences we have observed. If everything goes well, the ideal outcome of our analysis for the data above will produce:

- A peak of height 2 that starts at coordinate 10 and ends at coordinate 19
- A peak of height 2 that starts at coordinate 34 and ends at coordinate 43

Do note however that the DNA from one cell had two regions covered, whereas the two other cells had only one region each. Our peaks above do not indicate this. They represent a superposition of configurations across all cells. Other configurations leading to the same picture may be possible; there is no way to know that the covered regions appear all together or separately in the original DNA molecule.

Essential to remember:

> The peaks are cumulative, superimposed measures of the observed configurations across millions of different cells.

# Do peaks represent binding?

Now, up to this point, we avoided the word "binding" altogether - yet the purpose of ChIP-Seq is to measure exactly that. A protein bound to a DNA molecule can "protect" it from being washed away - then in a second "pulldown" stage, only the DNA bound to a certain protein is kept. Hence under the ideal condition, the result of a ChIP-Seq experiment contains DNA that was protected by one particular type of protein.

The reality is a bit different. Remember your ChIP-Seq data does not detect binding. It detects the abundance of DNA fragments in a pool of DNA molecules. Each measure indicated that a given DNA fragment was present when all the DNA was combined. As it turns out in practice, there may be several other reasons for DNA to be present in your library. Control samples are almost always required (see next question)

# How do we determine peaks from sequencing data?

There is a tendency to glossed over this step, relying on the "peak finder" then using the results that it produces. Once you think about the problem, you'll see how peak finding process is more counterintuitive than you'd imagine.

Remember how sequencing instrument produces a linear sub- measurement of a DNA fragments? This measured "read" has a length associated with it, but this length has nothing to do with the original fragment length.

If we mark the original fragment as `=` then we can see that depending on the DNA fragment lengths and our measurement (read) lengths the measured reads could fall into three different orientations.

```
         ========= DNA FRAGMENT =========

         Reads much shorter than fragment.
         ------->                 <------

         Reads as long as the fragment.
         ------------------>
                      <------------------

         Reads longer than the fragment
         ------------------------------------>
    <------------------------------------
```

Three different situations occur.

1. The read length is much smaller than the fragment.
2. The read length is about the same size as the DNA fragment
3. The read length is larger than the DNA fragment and runs past the fragment ends.

As a result, the analysis methods fall into two categories: 1 and 3 as the second case occurs very rarely. Often these are called "narrow" peak and "broad" peak methods.

Be advised that a different mindset and approach is required for each situation - methods that fit broad peaks are likely to work sub-optimally in for narrow ones and vice versa.

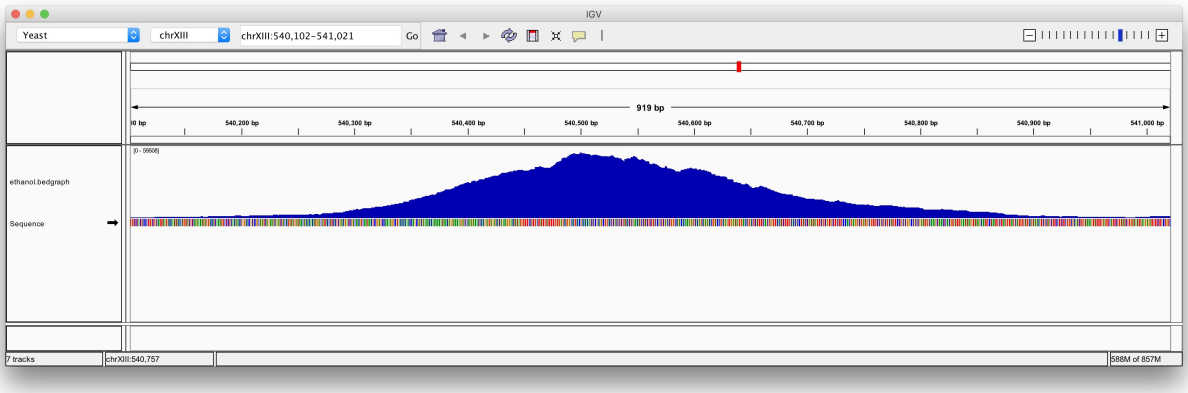
# What does a ChIP-Seq data measure?

If you refer to the sketch above only one thing is common across all data sets. The start of the reads that align on the forward strand indicates the left border of the DNA fragment. The start of the reads that align on the reverse strand all indicates the right border of the DNA fragment.

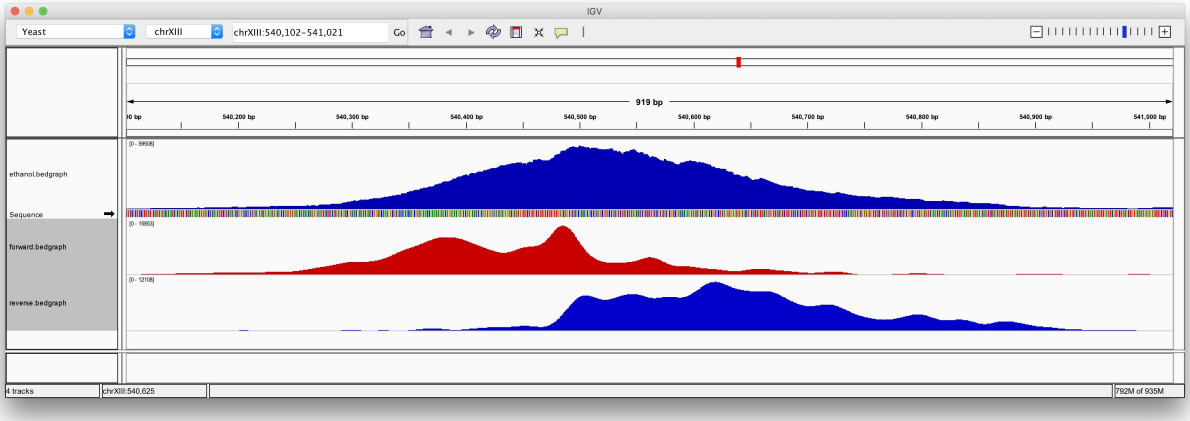
Hence we can state the fundamental "law" of ChIP-Seq data:

> ChIP-Seq data measure the outer borders, the "edges" of the DNA fragments.

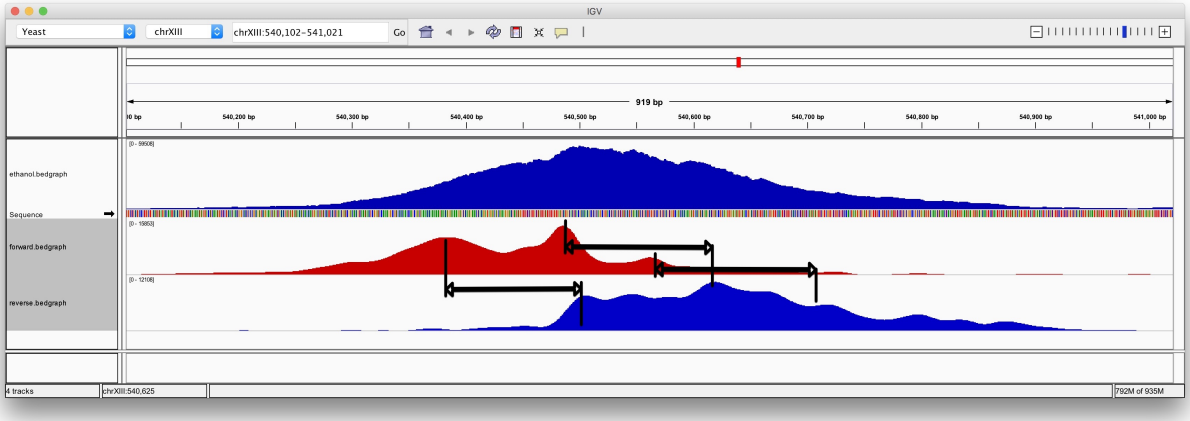
Of course this is not all that great of a news. It makes the picture a whole lot more complicated. Let me show you what this means. Imagine that you see a coverage that looks like this image below. it seems so clear cut, there is one binding event, in the middle, indicated by the highest point of the peak.




Sadly this is misleading. Where the fragments appear to overlap to most may be a superposition of many different, alternative positions. In this particular case if you were to break down the data by strand then plot only the coverages for the 5' ends for each strand separately you would see the following coverages:

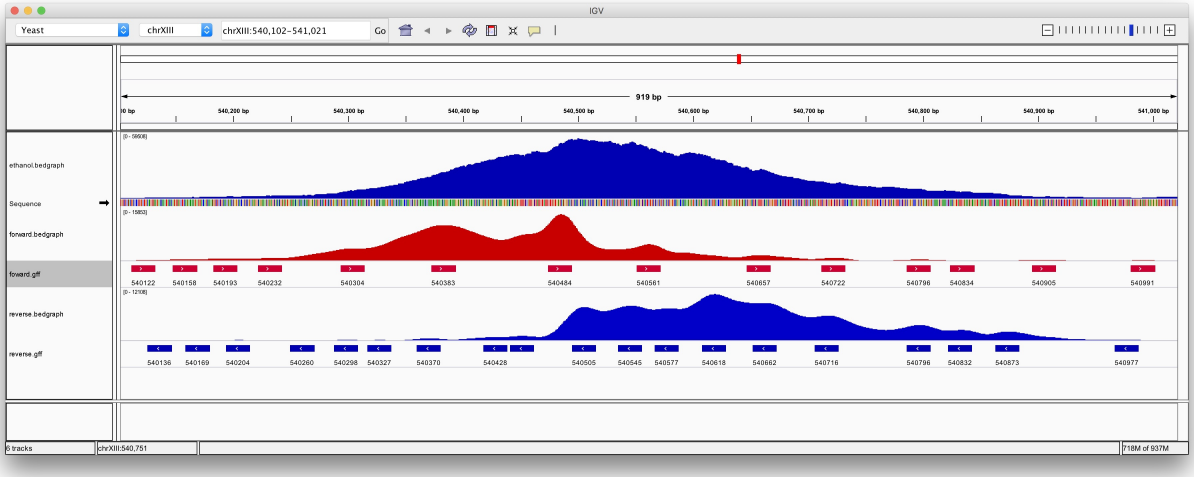



It is the superposition of these two lower tracks that gives the top coverage. But when plotted this way you can clearly see at least two distinct peaks. Remember these now track what the instrument measures: the ends of the DNA fragments. It appears that there are a few distinct fragments (position) on the DNA. Let me mark these positions.




If we trust that our ChIP-Seq method measures bound locations then these are the locations occupied by our protein. But note how none of them are bound in the location that the peak maxima indicates. If location accuracy is critical the peaks must be found by pairing up the peaks called separately from the

forward and reverse strands to attempt to reconstitute the original DNA fragments, rather than the center of each peak.




If peak location is not all that important you may be able to take the mid point of the original peak.

Much algorithmic effort is spent to create software that automatically builds the "consensus" location out of data. Some work better than other - none of the tools can handle all the situations and much subjectivity remains. The type of the data and its properties, whether or not a superposition of locations are present will all substantially alter the performance of the method you choose.

# What type of ChIP-Seq studies are performed?

Speaking by the extremes, and your study will likely fall somewhere in between but closer to one of these extremes, ChIP-Seq studies fall into two major categories:

1. The **Theory of Everything**: these are large scale studies, usually studying millions of locations, then often make sweeping, large-scale generalizations on where certain positions are about others. The methods driven by the prinicple "gain-some lose-some". The data analysis tuypically processes all peaks with the same parameters producing very high rates of false positives and false negatives. The goal of these studies is to beat random chance - which is a very low bar to pass.
2. The **Theory of One Peculiar Thing**: these are extremely focused studies that use ChIP-Seq studies to attempt to explain one particular phenomenon. The data often gets reduced to just a few locations where the very high accuracy of the methods is required.

Obviously, the needs of the two approaches are radically different. As a rule, if you need to accurately determine the "peak" at just a few locations the best tool that you can use is YOUR EYE! We make use of "peak callers" primarily because we can't afford to investigate every single location.

# Does the data need to be compared to control?

Yes. The ChIP-Seq protocols have turned out to be affected by many more systematic errors than initially thought. Hence control samples are needed to identify these errors.

Typically two kinds of controls samples are used: **IgG** control and **input** control. Each control can be thought of as an incomplete ChIP-Seq protocol, where the process intentionally skips a step of the process. These controls attempt to identify regions that are enriched by other procecesses beyond the protein being bound to the DNA.

- The **IgG** control is DNA resulting from a "mock" ChIP with Immunoglobulin G (IgG) antibody, which binds to non-nuclear antigen.
- **Input** control is DNA purified from cells that are cross-linked, fragmented, but without adding any antibody for enrichment.

One problem with IgG control is that if too little DNA is recovered after immunoprecipitation(IP), sequencing library will be of low complexity (less diverse) and binding sites identified using this control could be biased. I

Input DNA control is thought to be ideal in most of the cases. It represents all the chromatin that was available for IP. More information on choosing controls can be seen in:

- Biostars post: What Control For Chip-Seq: Input, Igg Or Untagged Strain? for a more details.
- Nature, 2015: ChIP-Seq: technical considerations for obtaining high-quality data

# Should my ChIP-Seq aligner be able to detect INDELs?

Absolutely YES!

One of the most misleading advice - one that is reiterated even in peer reviewed publications like Practical Guidelines for the Comprehensive Analysis of ChIP-Seq Data PLoS Comp. Biology 2014, stems from a fundamental misunderstanding of scientists when it comes to defining how "alignment" is different from "mapping."

We described the duality before in the alignment section stating that "mapping" refers to finding the genomic location of where a read originates from, whereas "alignment" refers to finding the place and arrangement about the genome. Since ChIP-Seq methods need to identify genomic locations many wrongly assume that the alignment methods that cannot handle insertions and deletion are still fine since we don't care about the exact alignment anyway.

It is time to put this misconception out of its misery as it has already caused and continues to cause substantial wasted efforts. So let's make it abundantly clear:

> An aligner that cannot handle insertions and deletions will not be able to find the correct coordinates when a measurement contains such differences. The "mapping" produced by such aligners over the reads in question will be severely and systematically affected.

As a rule, aligners that cannot detect insertions and deletions should **NEVER** be used under **ANY** circumstances - unless no other aligner can be utilized (very rare cases where other aligners would not work at all).

# What are the processing steps for ChIP-Seq data?

The general processing steps are as following:

1. Visualize and, in necessary, correct the quality of the sequencing data. See Visualizing sequencing data quality.
2. Align sequencing reads to a reference genome. The most popular read aligners are `bwa` and `bowtie`. Read the section Short Read Aligners.
3. Calling peaks from the alignment bam files.
4. Visualizing the resulting peak files and signal files.
5. Finding a biological interpretation for the positions where the peaks are observed.

# Aligning ChIP-Seq data

As we explained before chromatin immunoprecipitation (ChIP) followed by high-throughput DNA sequencing (ChIP-Seq) is a technique that is used to identify locations in the genome, for example, the footprints of genome-wide transcription factor binding sites and histone modification enriched regions.

We will demonstrate the steps used to process ChIP-Seq data through a re-analysis of a published scientific work.

## Which paper is going to be re-analyzed?

To illustrate all the analysis steps, we are going to use data from the following publication:

- Genome-Wide Mapping of Binding Sites Reveals Multiple Biological Functions of the Transcription Factor Cst6p in Saccharomyces cerevisiae. published in MBio in 2016

The data corresponding to this paper can be found in the SRA Bioproject:

- https://www.ncbi.nlm.nih.gov/bioproject/PRJNA306490
- The script that will reproduce this paper is located in http://data.biostarhandbook.com/chipseq/

This section provides an annotated guide to the commands in the `chip-exo-redo.sh` script. This script makes use of GNU Parallel to execute multiple commands at a time (up to the number of CPU on the system).

# How do I obtain the data for project PRJNA306490?

Make a few directoried to store data in:

```
mkdir -p data
mkdir -p refs
mkdir -p bam
```

Then perform the search for the run information:

```
esearch -db sra -query PRJNA306490  | efetch -format runinfo > runinfo.csv
```

The file contains information for four SRR runs

```
runinfo.csv | cut -d , -f 1,2,3 | head
```

will produce

```
Run,ReleaseDate,LoadDate
SRR3033157,2016-04-08,2015-12-18
SRR3033156,2016-04-08,2015-12-18
SRR3033155,2016-04-08,2015-12-18
SRR3033154,2016-04-08,2015-12-18
```

You could run the following command on each SRR run:

```
fastq-dump -O data --split-files -F SRR3033154
```

Or you could automate that into a single line, by cutting the first column for SRR ids then piping the result into `fastq-dump`

```
# Isolate just the run ids for the run.
cat runinfo.csv | cut -f 1 -d , | grep SRR > runids.txt

# Download the fastq files.
cat runids.txt | parallel --eta --verbose  "fastq-dump -O data --split-files -F {}"
```

The data download may chug along for a bit; the files are a few gigabytes each. You could use the `-X` flag to limit the dataset sizes.

# How do I find out more information each sample?

While the `runinfo.csv` contains quite a bit of information, maddeningly it lacks what arguably would be the most important detail: which SRA number corresponds to the samples mentioned in the paper. We need to run the Entrez Direct tool called `esummary` to connect run ids to sample names:

```
esearch -db sra -query PRJNA306490  | esummary > summary.xml
```

The XML file returned by this query has a nested structure with a wealth of information that can be difficult to extract. Sometime the easiest is to read it by eye and manually match the information. This might work here since there are only four samples. But there are ways to automate this data extraction.

The tool called `xtract` that is also part of EDirect: xtract is meant to facilitate turning XML files into tabular files. The `xtract` tool operates by parsing an XML file, matching and extracting and printing elements from this file.

Now XML as an industry standard with methods to do just this, there is a so-called `XPath` query language and the `XSL` styling language that could be used to turn XML documents into any other format. `xtract` has nothing to do with either of these - yet it is used to achieve the same. It is, in a way a re-imagining of what `XPath` and `XSL` do - but this time filtered through a bioinformaticians mind.

For example, the following command generates a new row on each element called named `DocumentSummary` then extracts the `acc` attribute of the `Run` element and prints that with the title. Investigate the XML file to see where this information is present.

```
cat summary.xml | xtract -pattern DocumentSummary -element Run@acc Title
```

will transform the `summary.xml` file into :

```
SRR3033157 GSM1975226: Cst6p_ethanol_rep2; Saccharomyces cerevisiae; ChIP-Seq
SRR3033156 GSM1975225: Cst6p_ethanol_rep1; Saccharomyces cerevisiae; ChIP-Seq
SRR3033155 GSM1975224: Cst6p_glucose_rep2; Saccharomyces cerevisiae; ChIP-Seq
SRR3033154 GSM1975223: Cst6p_glucose_rep1; Saccharomyces cerevisiae; ChIP-Seq
```

From this output we can see here that each sample has two replicates:

- `SRR3033157` and `SRR3033156` correspond to `Cst6p_ethanol`
- `SRR3033155` and `SRR3033154` correspond to `Cst6p_glucose`

# What other information do I need to start a ChIP-Seq analysis?

You will now need a genome sequence and an annotation file that lists the genes.

Sometimes (often?) it is surprisingly complicated to get the latest information even from websites dedicated to the distribution of data. As you will note for yourself, the data always seems to be either in the wrong format or combined in ways that are inappropriate to your current needs.

The data sources claim that it is a matter of "just downloading it" yet soon you'll be knee-deep in elbow-grease as you wrestle in the mud with uncooperative file formats. For example, SGD claims that the latest genome releases can be downloaded from

http://downloads.yeastgenome.org/sequence/S288C_reference/genome_releases/. As you happily and cheerfully download that data, you realize that the chromosomal naming coordinates are different across the sequence and the annotations. For example, the FASTA file will have the sequence names listed as:

```
> ref|NC_001133| [org=Saccharomyces cerevisiae] [strain=S288C] [moltype=genomic] [chromosome=I]
```

whereas the features the GFF files in the *very same* downloaded data will list the chromosomal coordinates as:

```
> chrI
```

Suffice to say no bioinformatics tool will be able to match up these two. To use this data you would need postprocess the file to rewrite `chromosome=I` to be `chrI`. A data janitor job if there ever was one.

# How do I get the "standard" yeast genome data and annotation?

The paper that we are reanalyzing makes use of the most up-to-date UCSC yeast genome release code named `sacCer3` from the UCSC download site some might balk at knowing that this release dates back to 2011. Quite a bit of knowledge could have been accumulated since then. One advantage is though that many tools, like IGV, already know of this genomic build and no custom genome building is necessary.

Let's get the genome:

```
# Reference genome.
REF=refs/saccer3.fa
```

Download the chromosomes and build a genome. The sequences are stored by chromosome:

```
URL=http://hgdownload.cse.ucsc.edu/goldenPath/sacCer3/bigZips/chromFa.tar.gz
curl $URL | tar zxv

# Get the chromosome sizes. Will be used later.
curl http://hgdownload.cse.ucsc.edu/goldenPath/sacCer3/bigZips/sacCer3.chrom.sizes > refs/sacC
er3.chrom.sizes

# Move the files
mv *.fa refs

# Create genome.
cat refs/chr*.fa > $REF
```

Remember to index the reference.

```
bwa index $REF
samtools faidx $REF
```

We now have the saccer3 genome on our local filesystem.

# How do I align the ChIP-Seq data?

We assume that by this time your `fastq-dump` programs above have finished. The alignment commands will all be of the form:

```
bwa mem $REF data/SRR3033154_1.fastq | samtools sort > SRR3033154.bam
```

Note how this data is aligned in single end mode and only file 1 is used. We too were surprised to learn this from the authors, their rationale is that they are using a ChIP-Exo method that sequences only a short segment of the fragment. We don't quite believe this rationale to be correct - suffice to say only half of the data was used and half of the data was tossed out.

We can automate the alignments as just one single line.

```
cat runids.txt | parallel --eta --verbose "bwa mem -t 4 $REF data/{}_1.fastq | samtools sort -
@ 8 > bam/{}.bam"
```

Once the step above completes the computationally most demanding process of ChIP-Seq has now been completed - but the analysis itself is only starting in its earnest.

# Do I need to trim data to borders?

As we mentioned this before ChIP-Seq data in general and ChIP-Exo in particular measures the left-right borders of the binding sites rather than the center of them. Even though tools should be able to reconstruct these borders algorithmically from the entire alignments, sometimes (often) it is best to not even "show" this spurious data to the algorithm, and to keep only the essential sections of the data, in this case, the borders only.

That is what the authors chose here, trimming 70 bp from the ends of each alignment using the bamUtil tool:

```
# Trim each bam file.
cat runids.txt | parallel --eta --verbose "bam trimBam bam/{}.bam bam/temp-{}.bam -R 70 --clip"


# We also need to re-sort the alignments.
cat runids.txt | parallel --eta --verbose "samtools sort -@ 8 bam/temp-{}.bam > bam/trimmed-{}
.bam"

# Get rid of temporary BAM files.
rm -f bam/temp*

# Reindex trimmed bam files.
cat runids.txt | parallel --eta --verbose "samtools index bam/trimmed-{}.bam"
```

The same process may work without this step.

# How do I visualize the alignments?

ChIP-Seq data is all about coverages and locations. As such the details of alignments are usually less important. Visualizing `BAM` files directly puts too much strain on visualizers as often many samples need to be viewed simultaneously.

A bed graph file is a variant of a BED file that is meant to display coverages over intervals. It is a simple format consisting of a tab separated format of this form:

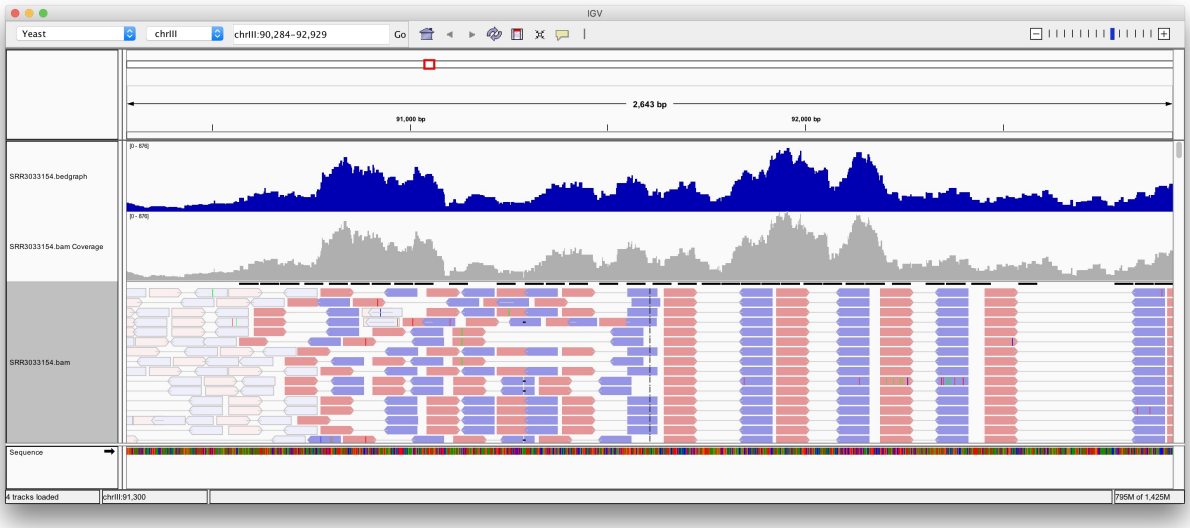
```
chrom start end value
```

Where `value` is the coverage over the interval between `start` and `end`. To create a coverage bedgraph file, we can use `bedtools`:

```
# Create an genome file that bedtools will require.
samtools faidx $REF

# Create a bedgraph file out of the BAM file.
bedtools genomecov -ibam bam/SRR3033154.bam  -g $REF.fai -bg > bam/SRR3033154.bedgraph
```

The bed graph file are smaller than the BAM file and may be visualized and processed with many other tools

Below is a figure of visualizing bedgraph and BAM files simultaneously. The upper track is bedgraph, the lower two panels are the coverage and alignment sections of a BAM file.




As it happens the bedgraph is not quite efficient either, IGV has to load the entire file into memory to visualize it. Yet another file format had to be invented (and we'll spare your the gory details) one that is called bigwig. A bigwig is a binary format that contains the same information as a bedgraph but allows for efficient access, hence allows software tools to jump to various location in the file more quickly. This format, by the way, was invented by Jim Kent, the "benevolent dictator" of the UCSC genome browser.

The format itself is not documented or described properly - a single program code, also written by Jim Kent was the sole source of understanding how the format works. We invite the readers to reflect on this a bit.

The tool called `bedGraphToBigWig` in Jim Kent's utilities can perform this transformation from bedgraph to bigwig:

```
bedGraphToBigWig bam/SRR3033154.bedgraph $REF.fai bam/SRR3033154.bw
```

We can automate these steps as well:

```
# Create the coverage files for all BAM files.
ls bam/*.bam | parallel --eta --verbose "bedtools genomecov -ibam {} -g $REF.fai -bg | sort -k
1,1 -k2,2n > {.}.bedgraph"

# Generate all bigwig coverages from bedgraphs.
ls bam/*.bedgraph | parallel --eta --verbose "bedGraphToBigWig {} $REF.fai {.}.bw"
```

Let's recapitulate what each file contains and what their sizes are:

- `SRR3033154.sam` (if generated) would be 5.3Gb, plain text, has alignment details, no random access.
- `SRR3033154.bam` is 423Mb, binary compressed format, has alignment details, allows random access.
- `SRR3033154.bedgraph` is 34Mb, plain text, coverage info only, loads fully into memory.
- `SRR3033154.bw` is 8.1Mb, binary format, coverage info only, allows random access.

We went from 5.3Gb to 8.1Mb, an approximately 1000 fold data size reduction. Hence, in general, when we intend to visualize these ChIP-Seq alignment files we transform them from BAM files into a **bigwig** files and subsequently, we only display these. Note that this same approach should be used when visualizing RNA-Seq coverage data as well.

# Are there other ways to generate bigwig files?

There are other tools that can generate bigwig files directly from BAM files. The package that contains them is called `deeptools` and may be installed with:

```
# Create a new environment.
conda create --name bioinfo

# Activate the environment.
source activate bioinfo

# Install the deeptools.
conda install deeptools
```

One specific program of this package called `bamCoverage` offers a large number of functions to generate coverages. It is also able to produce outputs with various smoothing, averaging and normalizing parameters:

```
# Counts within a window of 50bp.
bamCoverage -b bam/SRR3033154.bam -o bam/SRR3033154-digitized.bw

# Smooth the signal in a window.
bamCoverage -b bam/SRR3033154.bam --smoothLength 300 -o bam/SRR3033154-smooth.bw
```

We recommend a lot of caution when applying any of these data altering, methods. Use them only when there are good reasons and a well-defined rationale for using them. Never "borrow" settings from another analysis or publication. In general, the best bet is to use the original coverage and resort to smoothing and averaging methods only if these appear to be necessary.

As an example here is the comparison between the original coverage, on the top and a smoothed data based on the coverage at the bottom. As you can see for yourself, the data characteristics change quite a bit - sometimes this is desirable but not always.

# What is the next step of analyzing the data?

Once you have the alignments and coverages you can move onto the peak calling processes.

# Chip-Seq peak calling

This chapter continues with the data processing started in

- ChIP-Seq alignments.

We assume that you have the BAM alignment and coverage files.

# How do I reanalyze a ChIP-Seq experiment?

As you recall we are trying to reanalyze data from the following publication:

- Genome-Wide Mapping of Binding Sites Reveals Multiple Biological Functions of the Transcription Factor Cst6p in Saccharomyces cerevisiae. published in MBio in 2016

Whereas we can download the data from SRA the publication offers substantially less information on how the analysis itself was performed. It briefly states that:

> To identify peaks and compare biological duplicates, the program GEM (39) was used.

Now if you were to visit the GEM homepage you'll see that the tool takes a wide variety of parameters - each of which has consequences. Another line states:

> The noise level was calculated from averaged noise throughout each replicate.

You may not fully understand what this means in practice- we don't either. Finally, you'll find the following statement:

> "Binding events were manually curated in the Integrative Genomics Viewer o remove redundant events representing the same binding and false-positive events with poor peak shape or poor duplication"

I think we can all agree that this is not an objective process that would allow you to repeat the selection for yourself. What you will find that a typical ChIP-Seq data analysis is characterized by surprisingly subjective decision making. You will have quite a hard time following the analysis as the processes used to generate the results will often rely on seemingly subjective decisions.

# Can/should I contact the author of a study?

Yes. We too have contacted the authors of the study and they, in turn, have been very responsive to my questions. I would like to thank both Dr. Jens Nielsen and David Bergenholm for being most speedy and accommodating with their time.

Yet even so, as you yourself will experience it, if you ever undertake a reanalysis of this type, the process of moving forward with reproducing the results may still proceed at a surprisingly slow pace.

Somehow every single question/response pair missed an important detail, an essential yet perhaps tacitly assumed decision, one that required yet another followup email.

After twenty (!) or so email exchanges with the author I was finally able to simplify the process to the following script chip-exo-redo.sh that in turn, I believe to contain everything one needs to redo the analysis.

Notably, while the script contains most reproducible commands that I believe were used in the paper, the results do not exactly match (but are similar to) the published ones.

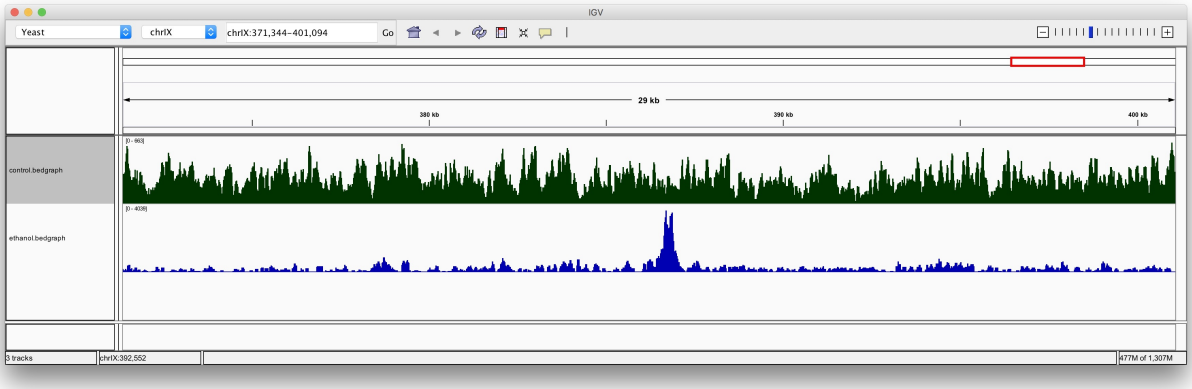
# Should I first summarize my data ?

Some like to simplify and summarize their data by merging all replicates into a single file and comparing them. By doing this merging, you will, of course, lose the independent measurements. On the other hand, it may allow you to gain a higher level understanding of all of its properties.

```
samtools merge -r bam/glucose.bam bam/SRR3033154.bam bam/SRR3033155.bam
samtools merge -r bam/ethanol.bam bam/SRR3033156.bam bam/SRR3033157.bam
samtools index bam/glucose.bam
samtools index bam/ethanol.bam
```

Generate the coverages for each, here you may skip making bigwig files since these files are not as large and IGV will handle them as is:

```
bedtools genomecov -ibam bam/glucose.bam  -g $REF.fai -bg > bam/glucose.bedgraph
bedtools genomecov -ibam bam/ethanol.bam  -g $REF.fai -bg > bam/ethanol.bedgraph
```




We can clearly see that in the region that we have visualized, the glucose sample has a more uniform coverage throughout whereas the ethanol affected sample shows a differential binding event. Auto-scaling, in this case, may be misleading, it appears as if the upper track has more data yet both have a similar number of reads 48 million versus 39 million.

The main difference is that the scale on the upper track goes to 1436 whereas the scale of the lower track goes to 5642.

# What tools can I use to predict peaks?

There is no shortage of tools that one can invoke to predict peaks. The most well-known is called MACS: Model-based Analysis for ChIP-Seq. Even for this tool there are two versions of it `macs` and `macs2` and each can be installed with conda or home brew.

Since these are python based tools they are best installed into a separate environment:

```
# Create a namespace for the tool
conda create --name macs python=2.7

# Activate the new environment.
source activate macs

# Install the tools.
conda install numpy
conda install macs2
```

You can also install the GEM peak predictor designed specifically for ChIP-Exo data - this is the tool used in the paper, and we'll use it as well later. Let's call peaks with macs2:

```
# Glucose samples.
GLU1=bam/trimmed-SRR3033154.bam
GLU2=bam/trimmed-SRR3033155.bam

# Ethanol samples.
ETH1=bam/trimmed-SRR3033156.bam
ETH2=bam/trimmed-SRR3033157.bam

macs2 callpeak -t $ETH1 $ETH2 -c $GLU1 $GLU2 --gsize 1E7  --name ethanol --outdir ethanol/
```

The results of this peak calling process will go into the directory `ethanol` where a number of files are created. The file called `ethanol_peaks.bed` contains the regions that form peaks.

Visually inspecting the results show the regions that MACS believes to form "peaks". As a comparison we also plot in a track the final results of the paper that would indicate the binding locations. As we can see `macs2` does reasonably good job in finding the locations, but at the same time the peaks don't quite seem to be well localized, they spread over larger regions.

# How do I refine the peaks?

The `ethanol_summits.bed` file contains the most likely maxima of each peak. These are single base pair wide location computed at the same time as the rest of the peaks:

Subsequent analysis typically consists of processing these summits into one place (if there are multiple of them overlapping in one region) and filtering for the most likely binding location. Alas neither process is entirely objective, much effort and custom coding goes into refining the final positions

As we mentioned before the parameter settings for filtering are almost always "reverse engineered." There are expectations, and then in turn the parameters are tuned in the direction that strengthens these expectations. For example, we could keep only peaks with a score above 10 with:

```
cat ethanol/ethanol_summits.bed | awk ' $5> 10 { print $0 }' > highscore.bed
```

Of course, here we treading into the dangerous territory of data manipulation - that is artificially selecting data with some properties. Unfortunately, this is a risk we must take in this case - as objective measures are few and far between.

# What else can I do to refine the peaks?

Due to the biased nature of the sequencing protocols ChIP-Seq data is typically affected by a very higher number of false positives. Thus the most common necessary step is that of removing some (or many) of the resulting peaks.

One such example is that of filtering out repetitive or low complexity regions of the genome. Data from these areas almost always indicate false signals as, if there is a measurement that matches equally

The `sdust` program of the [minimap][https://github.com/lh3/minimap]can be used to create an interval file with the regions of low complexity:

```
sdust refs/sc.fa > lowcomplexity.bed
```

then you could keep the high scoring values that do not fall on low complexity regions with:

```
bedtools intersect -v -a highscore.bed -b lowcomplexity.bed > highcomplexity.bed
```

You could also filter to keep only peaks that fall upstream of gene starts. Again bedtools could help:

```
bedtools flank -g refs/sc.fa.fai -l 1000 -r 0 -s -i refs/genes.gff > flanked.gff
```

Then intersect the peaks with that

```
bedtools intersect -u -a ethanol_summits.bed -b flanked.gff > upstream-peaks.bed
```

These types of tweaking and filtering are perhaps the most time-consuming aspects of a ChIP-Seq analysis - they are part of the quest to refine the very high false positive rates to a more reliable subset.

In the end the success of the ChIP-Seq analysis gets decided right here at peak filtering. Every single subsequent process operates on the position and sequence information of these peaks hence these peak filtering have the most effect on the results.

At the same time, and we reiterate, there is the danger that you may create an artificial, non-representative smaller subset of data.

# How did the paper call peaks?

The paper used the GEM peak caller in the following way:

```
# Shortcuts.
GEM=~/src/gem/gem.jar
DIST=~/src/gem/Read_Distribution_ChIP-exo.txt

# Run GEM on ethanol samples.
java -Xmx3G -jar $GEM --d $DIST --g refs/sacCer3.chrom.sizes --genome refs --exptCond1 $ETH1
--exptCond1 $ETH2 --f SAM --out ethanol --outBED --k 5  --smooth 3 --mrc 20 --k_win 30
```

and these will generate the peaks that most closely reproduce the published results.

# Chip-Seq motifs

This chapter continues with the data processing starting in both

- ChIP-Seq alignments.
- ChIP-Seq peak-calling
- ChIP-Seq data

Recall the section on Sequence Patterns, see Pattern matching where a sequence pattern (a motif) was defined as a sequence of bases described by certain rules. These rules could vary from requiring exact matches to being more elaborate requirements such as:

> All locations where `ATA` is followed by one or more `GC`s and no more than five `T`s then ending with `GTA`

## Why do scientists consider sequence motifs to be important?

Sequence motifs are of particular interest in ChIP-Seq data as it is believed that these motifs control (or at least influence) many biological processes: for example the binding of transcriptions factors to DNA.

## What makes motifs difficult to evaluate?

The most common challenge when dealing with motifs stems from not having a proper picture of just how often a given motif might be present in the genome, to begin with. Finding a motif in a binding location would be interesting only if seeing that motif has predictive value. Since the content of genome itself is not random the only way to properly assess the motif's abundance is to match it back to the genome.

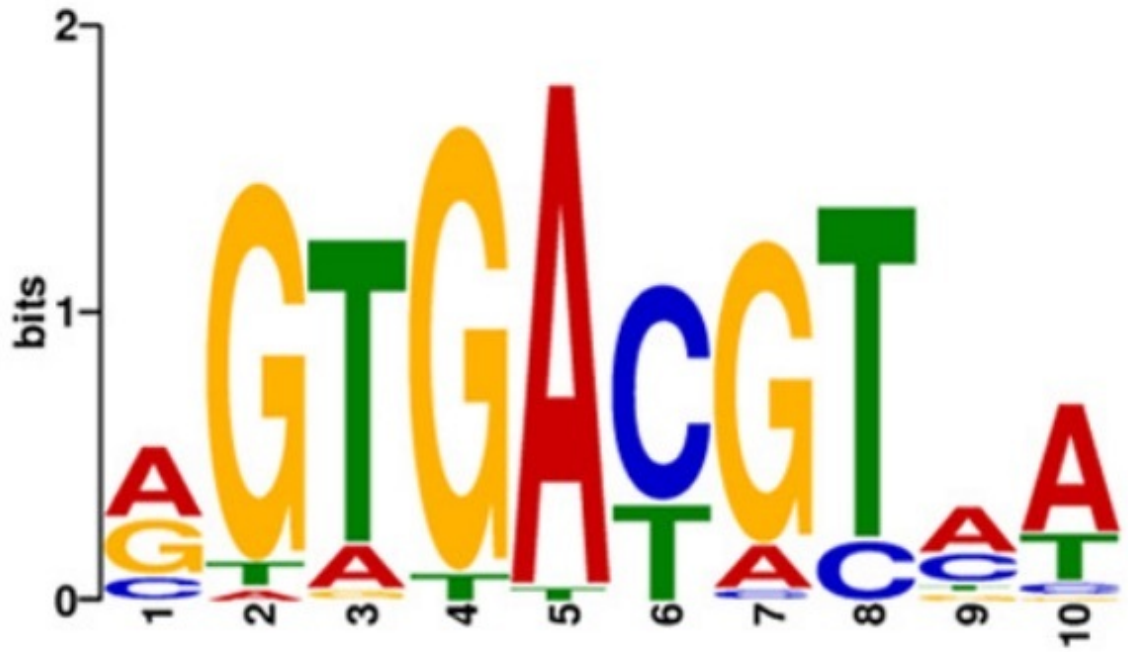
Then, of course, a motif might turn out to be "statistically significant" in that it may be observed at a rate that is different from random chance. That alone is a weak observation. At the same time, you should also be aware of and take into account the "effect size": just how big this deviation from random chance is.

## How do I find known motifs?

The paper that we are re-analyzing indicates that the motif `GTGACGT` was found in the location of the peaks. This pattern is then represented by a so-called sequence logo and described as

> A binding site motif of 5′ GTGACGT 3′ was enriched from the bound region sequences (discovered in 32 of the 40 regions)

A sequence (consensus) logo consist of a stack of letters where the size of each letter corresponds to the frequency of observation of each letter.




How do you find this motif in the genome? In the chapters Pattern matching we describe a few tools, below we'll use `seqkit` as an example:

```
REF=refs/sc.fa

# Turn the output into a GFF file:
cat $REF | seqkit locate -p GTGACGT | head
```

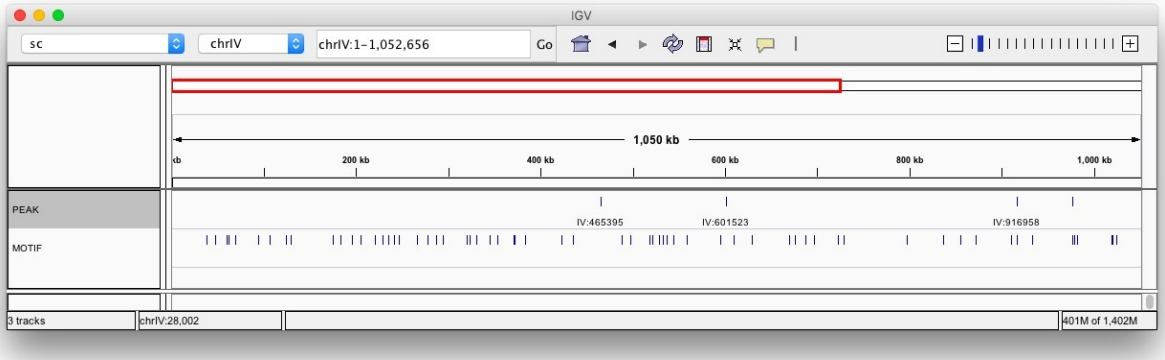
will produce the output:

```
seqID patternName pattern strand start end matched
chrI GTGACGT GTGACGT + 10705 10711 GTGACGT
chrI GTGACGT GTGACGT + 19820 19826 GTGACGT
chrI GTGACGT GTGACGT + 84191 84197 GTGACGT
chrI GTGACGT GTGACGT + 101042 101048 GTGACGT
chrI GTGACGT GTGACGT + 120400 120406 GTGACGT
...
```

To visualize this result in IGV you can turn it into a 9 column GFF file with an `awk` incanation (add `.` for unknown columns):

```
cat $REF | seqkit locate -p GTGACGT | grep -v seqID | awk ' { OFS="\t"; print $1, ".", ".", $5
, $6, ".", $4, ".", "." }' > motifs.gff
```

Visualizing the file you can see that there are `716` occurrences of the `GTGACGT` motif throughout the genome. The tracks below show the peaks found in the paper and the locations of the exact motif matches.




The published data finds `40` binding locations. Since there are `716` motif locations, it seems that even if all peaks fell on motifs only 5% of motif locations were bound by factors. Clearly, the presence of motif does not indicate binding. But as it turns out even that is not quite right. Let's find out how many of the published peaks fall on motifs:

```
curl -O http://data.biostarhandbook.com/chipseq/peaks-published.bed

bedtools intersect -u -a peaks-published.bed -b motifs.gff | wc -l
# 4
```

It turns out only 4 of the reported peaks intersect exactly with this motif locations. So the wording of 32 of 40 regions containing the motif is probably interpreted differently - though again we don't know what it means. Most likely the word "region" means a wider interval, not just the reported peak coordinate. Let's extend these peaks:

```
bedtools slop -b 50 -g refs/saccer3.fa.fai -i peaks-published.bed > wider.bed
bedtools intersect -u -a wider.bed -b motifs.gff | wc -l
# 11
```

Extending peaks by 50 bases left and right produced 11 more hits. We're getting there but not quite there yet.

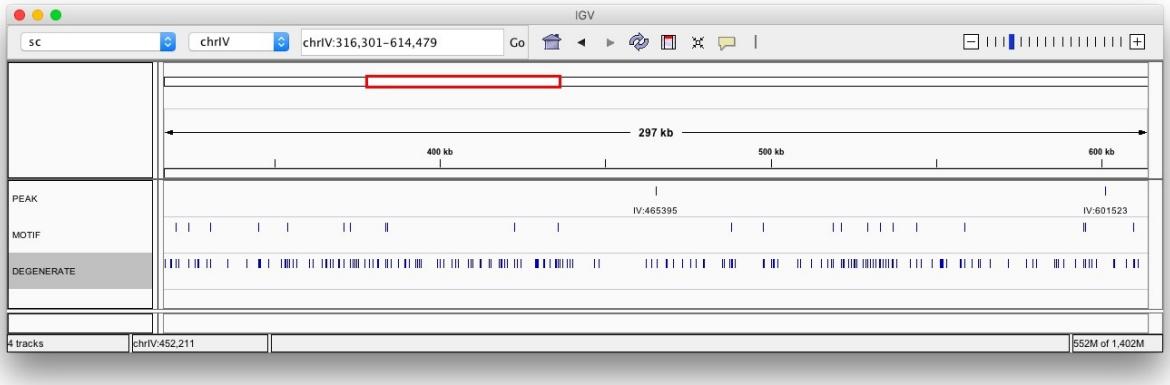
Of course, as you may have already noticed the motif above does not need to be exact - it allows for mismatches. On the other hand of course when we allow a motif to be inexact the count for its locations rises, often dramatically.

# How many "degenerate" motifs in the genome?

The sequence logo as drawn in the paper allows for some variations, but just how many it covers is difficult to visually assess. For all their visual flair and seeming "logic" sequence logos can often be misleading. They show the most common letter, but we don't see the individual motifs - only a superposition of all states. Let's run our pattern locator by allowing two alternatives at three locations:

```
cat $REF | seqkit locate -p 'G(T|A)GA(C|T)(G|A)T' | wc -l| grep -v seqID | awk ' { OFS="\t"; p
rint $1, ".", ".", $5, $6, ".", $4, ".", "." }' > motifs-generic.gff
```

It now produces `11349` possible matches! That is a tenfold increase over the exact motif! Note just how dense the motif coverage over the genome has become.




Even so, this pattern has picked up just two new peaks, while raising the total non-relevant locations for the motif 15 fold. Not a good tradeoff! That still does not reproduce all the motifs as described in the paper.

# How do I call motifs?

The GEM peak predictor that besides calling peaks also calls motifs, though the authors appear to have not used those results, instead chose to use the MEME suite.

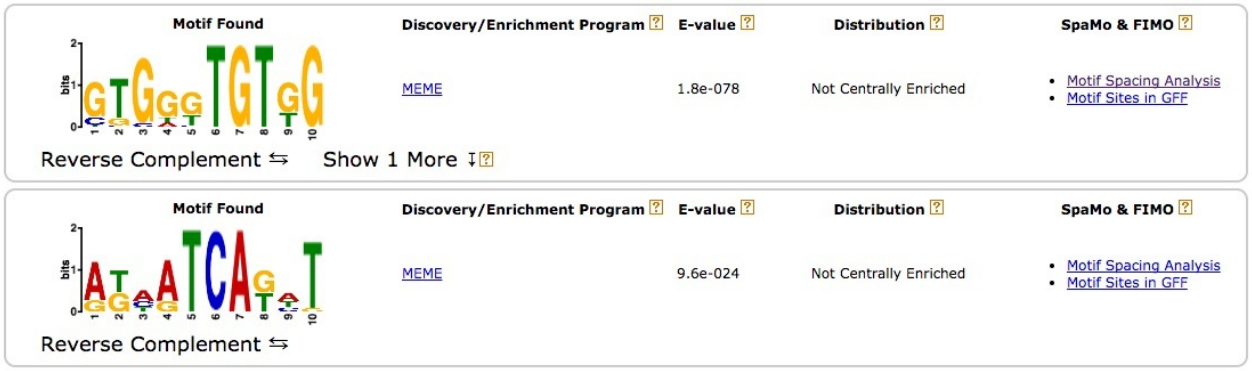
For example, if you were to attempt to find motifs for the peaks called with methods shown in the previous chapter you would need to do the following:

1. take the peak locations,
2. extend them left and right by a fixed amount,
3. extract the sequences that these intervals cover then
4. run a motif search program on these sequences

The script might look like this:

```
BED=ethanol/ethanol_summits.bed
REF=refs/saccer3.fa
N=50

# Extend the peak locations by 50 basepairs left and right.
bedtools slop -g $REF.fai -i $BED -b $N > extended.bed

# Extract the sequences corresponding to these locations.
bedtools getfasta -bed extended.bed -fi $REF -fo sequences.fa

# Run meme-chip on the sequences.
rm -rf results
meme-chip -meme-maxw 10 -meme-minw 5 -o results sequences.fa
```

These commands will generate a directory called `results` that among other information will contain the logo. In this case the logo looks like:




Obviously, this motif is not similar to the published motif at all. It appears to have a low complexity to it, indicating that our sequences represent too many repetitive regions. A good reminder that many/most peak prediction methods still produce a high number of false positives and that repetitive regions can influence results. Replace the `BED` file above with the different filtered files that we also suggested before: the `highscoring.bed` then the `highcomplexity.bed` file and rerun the script to see how the motif evolves. In the latter case we get the following:

This motif now matches the published result, though we can recognize the relative frequencies of the letters is different. If anything this motif may look a little cleaner than the published one. In addition we have filtered the data no by eye but with an objective measure of taking into account the genome complexity.

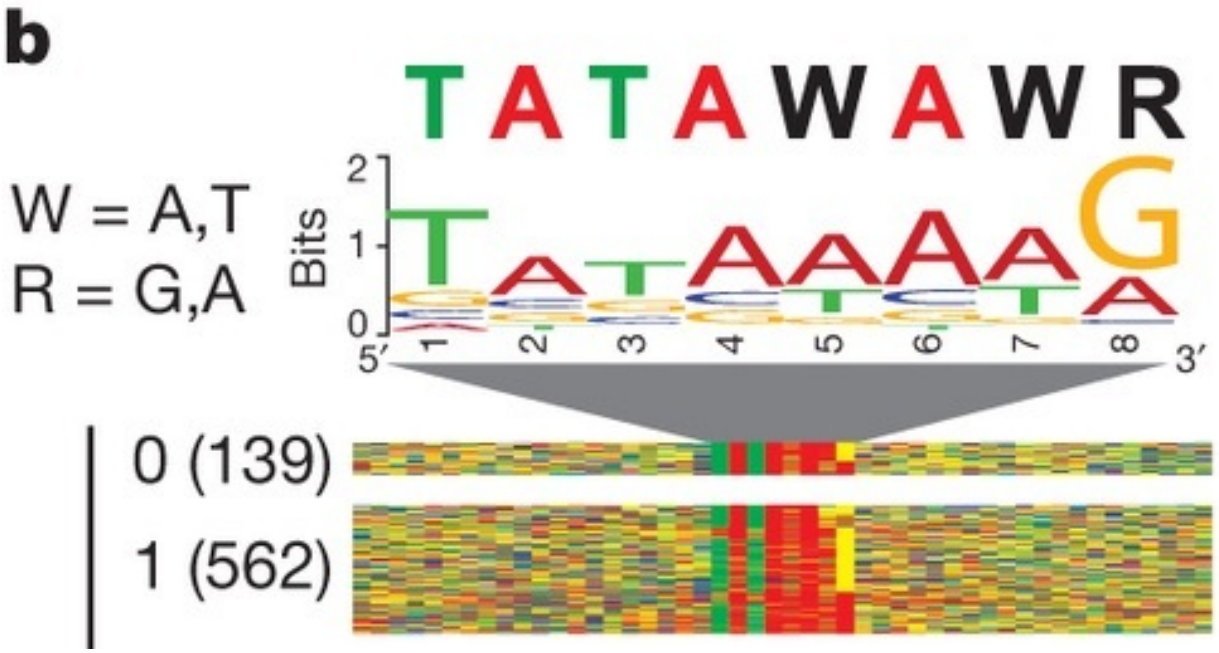
# What is the conclusion on finding motifs?

When it comes to motifs the results appear to be more subjective than the scientific field currently cares to admit. As much as we would all like to find a simple explanation of why a factor binds to a certain location, why a phenomenon is localized in the genome the evidence shows that the influence of the sequences and motifs are less well defined than we hoped for.

In our opinion motifs are pursued primarily because they appear to bring objectivity and determinism as to the why a factor binds to a given location - in reality, this effect is much weaker than many assume.

# Have motifs ever misled scientists?

The most educational cautionary tale of motif finding comes from the now retracted paper titled Genomic organization of human transcription initiation complexes published in Nature in 2013 (retracted in 2014 by the authors). In it the authors report sequence patterns of the form:




As it turns, it has been found that due to the degeneracy of the patterns the resulting motifs could have been produced just by chance in any preselected region of the genome. While the authors have attempted to correct and verify that the motif appears at rates above this chance they performed the comparison incorrectly and hence believed their results as being novel.

It does not help that the motif itself is so "obviously" clear cut. It is quite hard to believe that this pattern could be due to random chance. The goal here not to criticize the authors but to learn from them - you'll have to protect yourself from errors like with ongoing vigilance.

The lesson here is to understand and remember to treat motifs with a healthy dose of skepticism and to recognize their valuable yet also limited information content. It is very unlikely that whatever mechanisms you are after will be explained by the presence/absence of motifs alone. The DNA is composed of only four bases A,T,G and C. This low variety of choices coupled to a large number of measurements and samples can easily produce results that seem impossible to occur by chance.

# ChIP-Seq analysis

The author of this guide is Ming Tang. The material was developed for the Biostar Handbook.

- GitHub profile
- Diving into Genetics and Genomics

## What is ChIP-seq?

Chromatin immunoprecipitation (ChIP) followed by high-throughput DNA sequencing (ChIP-seq) is a technique to map genome-wide transcription factor binding sites and histone-modification enriched regions.

Briefly, DNA bounding proteins and DNA (Chromatin) are cross-linked by formaldehyde and the chromatin is sheared by sonication into small fragments (typically 200 ~ 600 bp). The protein-DNA complex is immnuoprecipitated by an antibody specific to the protein of interest. Then the DNA is purified and made to a library for sequencing. After aligning the sequencing reads to a reference genome, the genomic regions with many reads enriched are where the protein of interest bounds. ChIP-seq is a critical method for dissecting the regulatory mechanisms of gene expression.

## What are the processing steps for ChIP-seq data?

The general processing steps are as following:

1. Quality control of your `fastq` read files using tools such as FASTQC. Read our previous section: Visualizing sequencing data quality.
2. Aligning fastq reads to reference genome. The most popular read aligners are `bwa` and `bowtie`. Read section Short Read Aligners. `bowtie` has two versions: `bowtie1` and `bowtie2`. `bowtie2` is better for reads length greater than 50bp. It is still very common to use 36bp single-end sequencing library for ChIP-seq, so `bowtie1` is preferred for ChIP-seq with short reads.
3. Calling peaks from the alignment bam files.
4. Visualizing the resulting peak files and raw signal files.

## What are IgG control and input control for ChIP-seq?

Like any other experiments, a control is needed for ChIP-seq. There are two kinds of controls for ChIP-seq: IgG control and input control. IgG control is DNA resulting from a "mock" ChIP with Immunoglobulin G (IgG) antibody, which binds to non-nuclear antigen; input control is DNA purified from cells that are cross-linked, fragmented, but without adding any antibody for enrichment.

One problem with IgG control is that if too little DNA is recovered after immunoprecipitation(IP), sequencing library will be of low complexity and binding sites identified using this control could be biased. Input DNA control is ideal in most of the cases. It represents all the chromatin that was available for IP. Read this biostars post for discussion.

# Data sets

To illustrate all the analysis steps, we are going to use a public data set from two papers:

- Genome-wide association between YAP/TAZ/TEAD and AP-1 at enhancers drives oncogenic growth. Francesca Zanconato et.al. 2014. *Nature Cell Biology*. We will use transcription factor YAP1 ChIP-seq data in MDA-MB-231 breast cancer cells from this paper.
- Nucleosome positioning and histone modifications define relationships between regulatory elements and nearby gene expression in breast epithelial cells. Suhn Kyong Rhie et.al. 2014. *BMC Genomics*. We will use H3K27ac ChIP-seq data in MDA-MB-231 cells from this paper.
- The data sets can be found with accession number GSE66081 and GSE49651.

# How to obtain the data?

Read our previous section: Accessing the Short Read Archive (SRA).

Prepare folders and download the data.

```
# go to your home directory
cd ~
# make folders for this example
mkdir -p ChIP-seq/{data,results,scripts,software}
mkdir -p ChIP-seq/results/{bams,peaks,fastqc,motifs}
cd ChIP-seq/data

# for YAP1. It takes time, alternatively download fastq directly from EGA https://www.ebi.ac.u
k/ega/home
fastq-dump SRR1810900

# for H3K27ac
fastq-dump SRR949140

# for input DNA
fastq-dump SRR949142
```

In general, do not change the names of files. This can be error prone, especially when you have many samples. Instead, prepare a name mapping file, and interpret the final results with the mapped names. For demonstrating purpose, I will re-name the files to more meaningful names.

```
mv SRR1810900.fastq YAP1.fastq
mv SRR949140.fastq H3K27ac.fastq
mv SRR949142.fastq Input.fastq
```

Now, run `FASTQC` on all the samples. Read previous section on looping over files. or use GNU parallel:

```
# always specify -j to not abuse the computing cluster
find *fastq | parallel -j 4 fastqc {} -o ../results/fastqc
```

Overall, the sequencing qualities are good for all three samples, I will skip trmming low quality bases, and go ahead with the original fastq files for aligning.

# How to align ChIP-seq reads?

once you have finished QC confirming the sequencing reads are of high quality and with no adaptor contamination, you can do:

```
# use 10 threads to speed up. should finish within 10 mins
# change the path to the bowtie1 index in your own machine.
bowtie -p 10 --best --chunkmbs 320 /risapps/reference/bowtie1/hg19 -q YAP1.fastq -S ../results
/bams/YAP1.sam

# reads processed: 24549590
# reads with at least one reported alignment: 19664247 (80.10%)
# reads that failed to align: 4885343 (19.90%)
Reported 19664247 alignments to 1 output stream(s)

bowtie -p 10 --best --chunkmbs 320 /risapps/reference/bowtie1/hg19 -q H3K27ac.fastq -S ../resu
lts/bams/H3K27ac.sam

# reads processed: 29759754
# reads with at least one reported alignment: 29100444 (97.78%)
# reads that failed to align: 659310 (2.22%)
Reported 29100444 alignments to 1 output stream(s)

bowtie -p 10 --best --chunkmbs 320 /risapps/reference/bowtie1/hg19 -q Input.fastq -S ../result
s/bams/Input.sam

# reads processed: 18811563
# reads with at least one reported alignment: 18127743 (96.36%)
# reads that failed to align: 683820 (3.64%)
Reported 18127743 alignments to 1 output stream(s)
```

For demonstrating purpose, I will write down explicitly all the commands, but the GNU parallel trick can be used to save you some typing. In the end, I will introduce a shell script to chain all the steps together and show you how to make reusable codes.

Convert `sam` to sorted `bam` :

```
# avoid writing unsorted bam to disk
# note that samtools sort command changed invoke pattern after version 1.3 https://www.biostar
s.org/p/169745/
# samtools view and index will be multi-thread soon https://github.com/samtools/htslib/pull/397

cd ../results/bams
samtools view -bS YAP1.sam | samtools sort -@ 4 - -T YAP1 -o YAP1.sorted.bam
# index the sorted bam
samtools index YAP1.sorted.bam

samtools view -bS H3K27ac.sam | samtools sort -@ 4 - -T H3K27ac -o H3K27ac.sorted.bam
samtools index H3K27ac.sorted.bam

samtools view -bS Input.sam | samtools sort -@ 4 - -T Input -o Input.sorted.bam
samtools index Input.sorted.bam

# remove sam files to save space , rm can be dangerous, use rm -i if necessary
rm *sam
```

# How do I call peaks?

MACS is short for Model Based Analysis for ChIP-Seq data, which was developed in Sheirly Liu's lab at Harvard by Tao Liu. It is one of the most popular peak-calling algorithums used in literatures. First, you need to install MACS.

```
## install through pip
pip install macs

# peak calling for human samples, with default p value 1e-5
macs -t YAP1.sorted.bam -c Input.sorted.bam -n YAP1 -g hs -p 1e-5 -o ../peaks/

## peak calling for H3K27ac
macs -t H3K27ac.sorted.bam -c Input.sorted.bam -n H3K27ac -g hs -p 1e-5 -o ../peaks/
```

In addition to `bam` files, `macs` can take in other input file formats such as `bed` and `sam`. type `macs` on your terminal and press `Enter` to see full list of helps.

Parameter settings can be different depending on your own data sets. There is no one-fit-all settings. You may want to load the called peaks in to `IGV` and visually inspect them.

Note: Tao Liu is developing MACS2. there is a `-broad` option to call broad peaks for histone modification ChIP-seq(e.g. H3K36me3). However, in my experience, `macs14` is still widely used and performs very well.

Take a look at How to set MACS2 peak calling parameters.

# How do I call super-enhancers?

The fancy "super-enhancer" term was first introduced by Richard Young's lab in Whitehead Institute. Basically, super-enhancers are clusters of enhancers (most commonly defined by H3K27ac peaks) stitched together if they are within 12.5kb apart. The concept of super-enhancer is NOT new. One of the most famous example is the Locus Control Region (LCR) that controls the globin gene expression, and this has been known for decades. If you are interested in the controversy, please read:

A review in Nature Genetics What are super-enhancers? Another comment in Nature Genetics Is a super-enhancer greater than the sum of its parts?

From the HOMER page **How finding super enhancers works:**

> Super enhancer discovery in HOMER emulates the original strategy used by the Young lab. First, peaks are found just like any other ChIP-Seq data set. Then, peaks found within a given distance are 'stitched' together into larger regions (by default this is set at 12.5 kb). The super enhancer signal of each of these regions is then determined by the total normalized number reads minus the number of normalized reads in the input. These regions are then sorted by their score, normalized to the highest score and the number of putative enhancer regions, and then super enhancers are identified as regions past the point where the slope is greater than 1.

`HOMER` is a very nice tool for finding enriched peaks, doing motif analysis and many more. It requires making a tag directory first. `ROSE` is the tool from Richard Young's lab. Since it works directly with `bam` files, we will use it for demonstration.

Download `ROSE` to the `software` folder.

```
cd ../../software/
wget https://bitbucket.org/young_computation/rose/get/1a9bb86b5464.zip
unzip 1a9bb86b5464.zip
cd young_computation-rose-1a9bb86b5464/

# there are multiple python scripts in the folder.
# one has to run ROSE inside the ROSE folder.
python ROSE_main.py -g hg19 -i ../../results/peaks/H3K27ac_peaks.bed -r ../../results/bams/H3K
27ac.sorted.bam -c ../../results/bams/Input.sorted.bam -o ../../results/peaks
```

# How do I get a normalized `bigWig` track for visualizing the raw signal?

After you get the peak calls, we want to inspect the quality of the peaks by visulizing them in a browser such as `IGV` along with the raw signal track. We will use `bigWig` track for this purpose.

From the UCSC help:

> The bigWig format is for display of dense, continuous data that will be displayed in the Genome Browser as a graph. BigWig files are created initially from wiggle (wig) type files, using the program wigToBigWig. The resulting bigWig files are in an indexed binary format. The main advantage of the bigWig files is that only the portions of the files needed to display a particular region are transferred to UCSC, so for large data sets bigWig is considerably faster than regular wiggle files

Make sure you understand the other two closely related file formats: bedgraph and wig file.

`macs14` can generate a `bedgraph` file by specifying `--bdg`, but the resulting `bedgraph` is **NOT** normalized to sequencing depth. Instead, we are going to use another nice tool `deeptools` for this task. It is a very versatile tool and can do many other things.

Make a bigWig file:

```
# install deeptools
conda install -c bioconda deeptools

# normalized bigWig with Reads Per Kilobase per Million mapped reads (RPKM)
bamCoverage -b YAP1.sorted.bam --normalizeUsingRPKM --binSize 30 --smoothLength 300 -p 10 --ex
tendReads 200 -o YAP1.bw

bamCoverage -b H3K27ac.sorted.bam --normalizeUsingRPKM --binSize 30 --smoothLength 300 -p 10 -
-extendReads 200 -o H3K27ac.bw

bamCoverage -b Input.sorted.bam --normalizeUsingRPKM --binSize 30 --smoothLength 300 -p 10 --e
xtendReads 200 -o Input.bw
```

I set `--extendReads` to 200bp which is the fragment length. Why should we extend the reads? Because in a real ChIP-seq experiment, we fragment the genome into small fragments of ~200bp, and pull down the protein bound DNA with antibodies. However, we only sequence the first 36bp(50bp, or 100bp depending on your library) of the pull-down DNA. To recapitulate the real experiment, we need to extend it to the fragment size.

You can read more details Why do we need to extend the reads to fragment length/200bp?

# How do I put all the steps together?

## Shell script comes to rescue

Up until now, we have finished the basic analyzing steps for ChIP-seq. We aligned the reads to get `bam` files; we called peaks using `macs`; we generated `bigWig` tracks as well. This is great, but one of the stumbling blocks for beginners is learning how to reuse the commands we have typed. We do not want to type the same command time and time again changing the file names only when new data come in.

The strategy is to write a shell script to chain all the steps together. see the section writing scripts in the handbook.

First, you need to think about how the workflow should be. We will need to download the `fastq` files and convert them to `bam` files; next, we will call peaks for a pair of IP and Input. Sometimes, you may get sorted `bam` files, which can be used directly to call peaks. It is reasonable that we have two shell scripts for each task, respectively.

```
#! /bin/bash

set -euo pipefail

# the SRA number to work with
SRA=$1

# the reference file path, change to where you put the reference
REF="/risapps/reference/bowtie1/hg19"

##extract the fastq files
fastq-dump $SRA

## step1, quality control of the fastq files
fastqc "${SRA}".fastq

## step2, algin the reads with bowtie
bowtie -p 10 --best --chunkmbs 320 $REF -q "${SRA}".fastq -S "${SRA}".sam

## step3, convert sam to bam, and index the bam
samtools view -bS "${SRA}".sam | samtools sort -@ 4 - -T "${SRA}" -o "${SRA}".sorted.bam
samtools index "${SRA}".sorted.bam

# remove sam to save space
rm "${SRA}".sam
```

For more on `set -euo pipefail`, read this post Use the Unofficial Bash Strict Mode (Unless You Looove Debugging).

Save it to `sra2bam.sh`. Make it executable `chmod u+x sra2bam.sh`.

execute:

```
# YAP1
./sra2bam.sh SRR1810900

# H3K27ac
./sra2bam.sh SRR949140

# Input
./sra2bam.sh SRR949142
```

Now, with the `sra2bam.sh` script, it saves you from typing different file names for aligning `fastqs` and converting `sam` files. Moreover, it can be used to process any number of `sra` files.

If you have a `sra_id.txt` file with one `sra` id on each line, you can:

```
## run 6 jobs in parallel
cat sra_ids.txt | parallel -j 6 ./sra2bam {}
```

to process all of them.

second shell script `bam2peaks.sh`:

```
#! /bin/bash

set -euo pipefail

IP_bam=$1
Input_bam=$2
oprefix=$(basename "${IP_bam}" .sorted.bam)

macs -t "${IP_bam}" -c "${Input_bam}" -n "${oprefix}" -g hs -p 1e-5 -o ${oprefix}_peaks
```

The `sra2bam.sh` script will generate the indexed `bam` files, those `bam` files can be fed into `bam2peaks.sh` to call peaks. In our example data set, we only have two IPs and one Input. We can call peaks by:

```
# call peaks for H3K27ac
./bam2peaks.sh SRR949140.sorted.bam SRR949142.sorted.bam

# call peaks for YAP1
./bam2peaks.sh SRR1810900.sorted.bam SRR949142.sorted.bam
```

Imagine we have a lot of bam files generated by `sra2bam.sh`, and we want to call peaks for them, how should we stream the workflow?

Because it invovles a pair of files (IP and Input control) for calling peaks, we need to make a tab delimited `txt` file with pairs of file names on each line.

```
cat bam_names.txt
SRR949140.sorted.bam    SRR949142.sorted.bam
SRR1810900.sorted.bam    SRR949142.sorted.bam
```

Now, we can loop over the `bam_names.txt` file one by one and call peaks:

```
cat bam_names.txt | while read -r IP Input
do
    ./bam2peaks.sh "${IP}" "${Input}"
done
```

## Arguments handling for shell script

What we have so far is all good, but what if you want to make your script more flexiable? e.g. you want to specify mouse genome for mapping reads. You can add arguments for your script.

```
#! /bin/bash

set -euo pipefail

# show help
show_help(){
cat << EOF
  This is a wrapper to align fastq reads to bam files for ChIP-seq experiments.
  usage: ${0##*/} -d < a directory path containing the fastq.gz files > -r < h or m>
        -h display this help and exit
```

```
        -f the full path name of the fastq file
        -r reference genome to be used. m for mouse; h for human
EOF
}

## if there are no arguments provided, show help
if [[ $# == 0 ]]; then show_help; exit 1; fi

## parsing arguments
while getopts ":hf:r:" opt; do
  case "$opt" in
    h) show_help;exit 0;;
    f) fq=$OPTARG;;
    r) REF=$OPTARG;;
    '?') echo "Invalid option $OPTARG"; show_help >&2; exit 1;;
  esac
done

## set up some defaults
REF=${REF:-"h"}

## check if the fastq file exist
if [ ! -f "$fq" ]; then
  echo "file ${fq} does not exit"
  exit 1
fi

## reference genome path for mouse and human
human_ref="/risapps/reference/bowtie1/hg19"
mouse_ref="/risapps/reference/bowtie1/mm9"

## which species? human or mouse?
if [[ $REF == "m" ]]; then
  ref_genome="${mouse_ref}"
elif [[ $REF == "h" ]]; then
  ref_genome="${human_ref}"
else
  echo "please only specify m or h for the reference genome"
  exit 1
fi

## extract output name
oprefix=$(basename "${fq}" .fastq)

## mapping
bowtie -p 10 --best --chunkmbs 320 ${ref_genome} -q "${fq}" -S "${oprefix}".sam

## convert sam to sorted bam
samtools view -bS "${oprefix}".sam | samtools sort -@ 4 - -T "${oprefix}" -o "${oprefix}".sort
ed.bam

# index sorted bam
samtools index "${oprefix}".sorted.bam

rm "${oprefix}".sam
```

With this script, you can map not only ChIP-seq data for human but also for mouse. You can surely add more arguments to set `bowtie` mapping thread numbers and `samtools sort` thread numbers (the above script uses 10 and 4 respectively). You can read more on arguments handling for shell scripts: small

getopts tutorial.

Congratulations! You have come this far with me. If you want to take the next level of making your analysis reproducible and flexiable, read our Makefile section.

# What are the black-listed regions?

For ChIP-seq, it is important to filter artifact regions that has abnomral high signals. From the website of Anshul Kundaje in Stanford University:

> These regions are often found at specific types of repeats such as centromeres, telomeres and satellite repeats. It is especially important to remove these regions that computing measures of similarity such as Pearson correlation between genome-wide tracks that are especially affected by outliers

One can download the `hg19` blacklist:

```
wget https://www.encodeproject.org/files/ENCFF001TDO/@@download/ENCFF001TDO.bed.gz

# 411 regions are black-listed
zless ENCFF001TDO.bed.gz | wc -l
411
```

Note that more and more sequencing projects are moving their reference genome to the latest version `GRCh38`. `GRCh38` blacklist was uploaded by Anshul Kundaje on 2016-10-16. The file can be downloaded by:

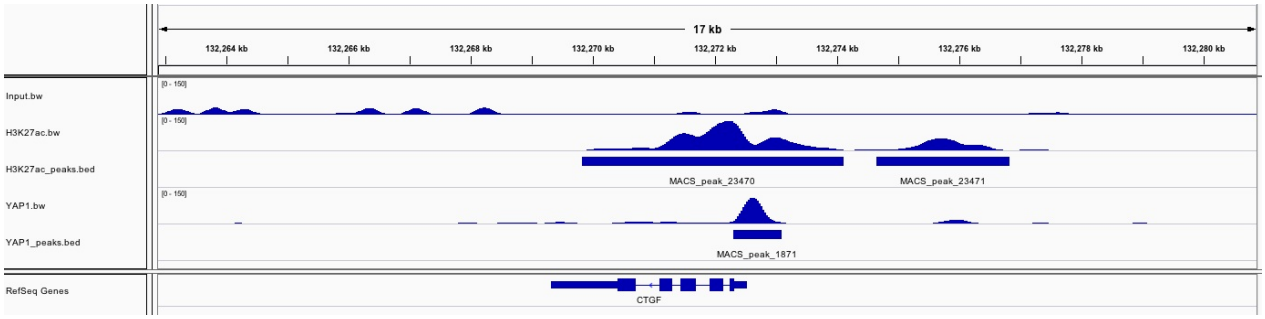
```
wget https://www.encodeproject.org/files/ENCFF419RSJ/@@download/ENCFF419RSJ.bed.gz
```

# How do I visualize the peaks?

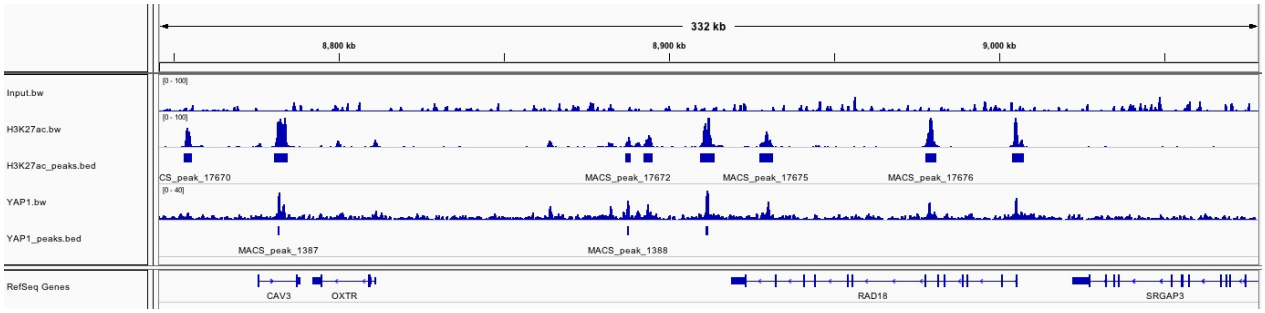
We will use `IGV` to visualize the peaks and raw signal. check our previous section on using IGV.

Open `IGV`, click `File`, then `Load From File`, choose the peak files end with `bed` and the raw signal files end with `bw`. you can change the scale of the `bigWig` tracks by right click the tracks and choose `Set Data Range`. Now, you can browser through the genome or go to your favoriate genes.

YAP1 binds to known target gene *CTGF*:



one other example from the original *Nature Cell Biology* paper at the *RAD18* locus:




We see that `macs` did a pretty good job in identifying enriched regions for histone modifications and transcription factor (TF) binding peaks. However, we do also see some potential YAP1 binding sites are missed. Fine tuning the peak calling parameters may improve the results.

# ChIP-Seq Downstream Analysis 1

The author of this guide is Ming Tang. The material was developed for the Biostar Handbook.

- GitHub profile
- Diving into Genetics and Genomics

## How do I compare the peak sets?

Now, we have the peaks called for H3K27ac and YAP1 and we want to exclude the peaks that overlap with the blacklisted regions. How should you do it? We will use a very popular tool `bedtools` developed by Aaron Quinlan lab for this purpose. Documentation of `bedtools` is of top niche. Also read Interval analysis tools section in the handbook.

There are many sub-commands for `bedtools`, but the most commonly used one is `bedtools intersect`:

```
# make sure you are inside the folder containing the peak files and the black-listed region fi
le.

# unzip the file
gunzip ENCFF001TDO.bed.gz

# How many H3K27ac peaks overlap with black-listed regions?
bedtools intersect -a H3K27ac_peaks.bed -b ENCFF001TDO.bed -wa | wc -l
#14

# exclude those peaks
bedtools intersect -a H3K27ac_peaks.bed -b ENCFF001TDO.bed -v > H3K27ac_filtered_peaks.bed

# do the same for YAP1
bedtools intersect -a YAP1_peaks.bed -b ENCFF001TDO.bed -v > YAP1_filtered_peaks.bed
```

How many YAP1 peaks overlap with H3K27ac peaks?

```
bedtools intersect -a YAP1_filtered_peaks.bed -b H3K27ac_filtered_peaks.bed -wa | wc -l
#1882

bedtools intersect -a YAP1_filtered_peaks.bed -b H3K27ac_filtered_peaks.bed -wa | sort | uniq |
 wc -l
#1882

bedtools intersect -a H3K27ac_filtered_peaks.bed -b YAP1_filtered_peaks.bed -wa | wc -l
#1882

bedtools intersect -a H3K27ac_filtered_peaks.bed -b YAP1_filtered_peaks.bed -wa | sort | uniq |
 wc -l
#1772
```

what's happening here? why there are only 1772 unique H3K27ac peaks overlap with YAP1 peaks?

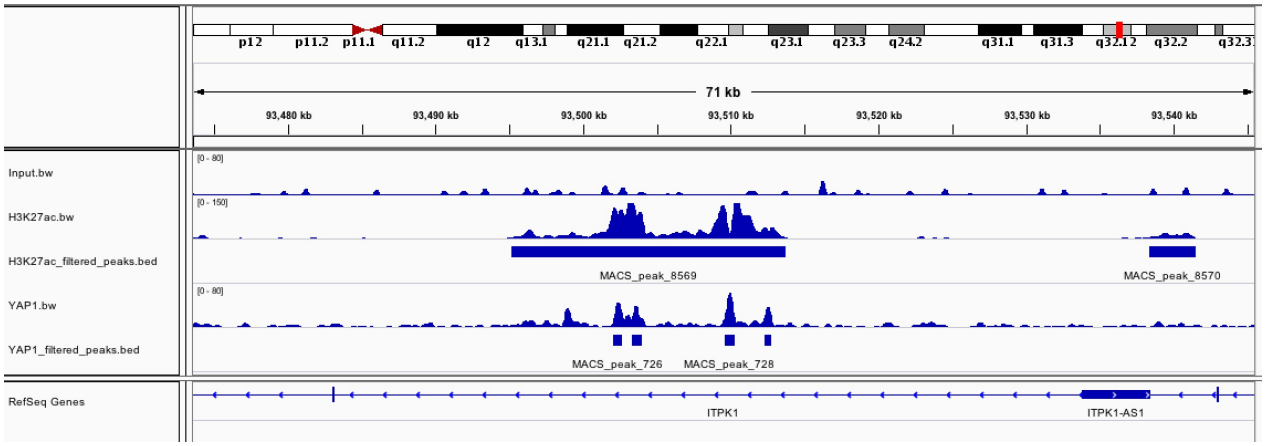
It turns out that bedtools will output the overlapping peaks of H3K27ac whenever there is an overlapping with YAP1 peaks, and it is possible that the same H3K27ac peak (tends to be really broad peaks) overlaps with multiple YAP1 peaks. With that in mind, I always do `sort | uniq` following `bedtools` command.

You can identify those H3K27ac peaks by:

```
bedtools intersect -a H3K27ac_filtered_peaks.bed -b YAP1_filtered_peaks.bed -wa | sort | uniq
-c | sort -k1,1nr | head
      4 chr14    93495168    93513756    MACS_peak_8569    935.17
      4 chr20    30276949    30312747    MACS_peak_16217    3100.00
      3 chr10    95216260    95244053    MACS_peak_3871    3100.00
      3 chr11    65654426    65689186    MACS_peak_5045    3100.00
      3 chr14    23441596    23453313    MACS_peak_7876    3100.00
      3 chr1    86040764    86052311    MACS_peak_1241    3100.00
      3 chr1    86070388    86080738    MACS_peak_1243    2714.64
      3 chr19    13943280    13955945    MACS_peak_13069    3100.00
      3 chr19    13956264    13965827    MACS_peak_13070    3100.00
      3 chr1    95080161    95091780    MACS_peak_1347    3038.66
```

We see some H3K27ac peaks can overlap with up to 4 peaks of YAP1.

If you choose to believe your eyes, let's visualize it in `IGV` :




One of the under-appreciated tools is `BEDOPS` . It has many nice features as well and the documentaion is also very good. Read this biostar post: Question: Bedtools Compare Multiple Bed Files?

# How do I annotate my peaks?

The next obvious question is where are the peaks in terms of their genomic context. e.g. What genes are the peaks associated with? Are some of the peaks located in intergenic region? You need to **annotate** your peaks. There are many tools you can use. You can find a list here.

I will show you how to do peak annotation using an R bioconductor package `ChIPseeker` developed by Guangchuan Yu.

```
> # load the packages, install them following the instructions in the links above
> library(ChIPseeker)
> library(TxDb.Hsapiens.UCSC.hg19.knownGene)
> library(rtracklayer)
> library("org.Hs.eg.db")
```

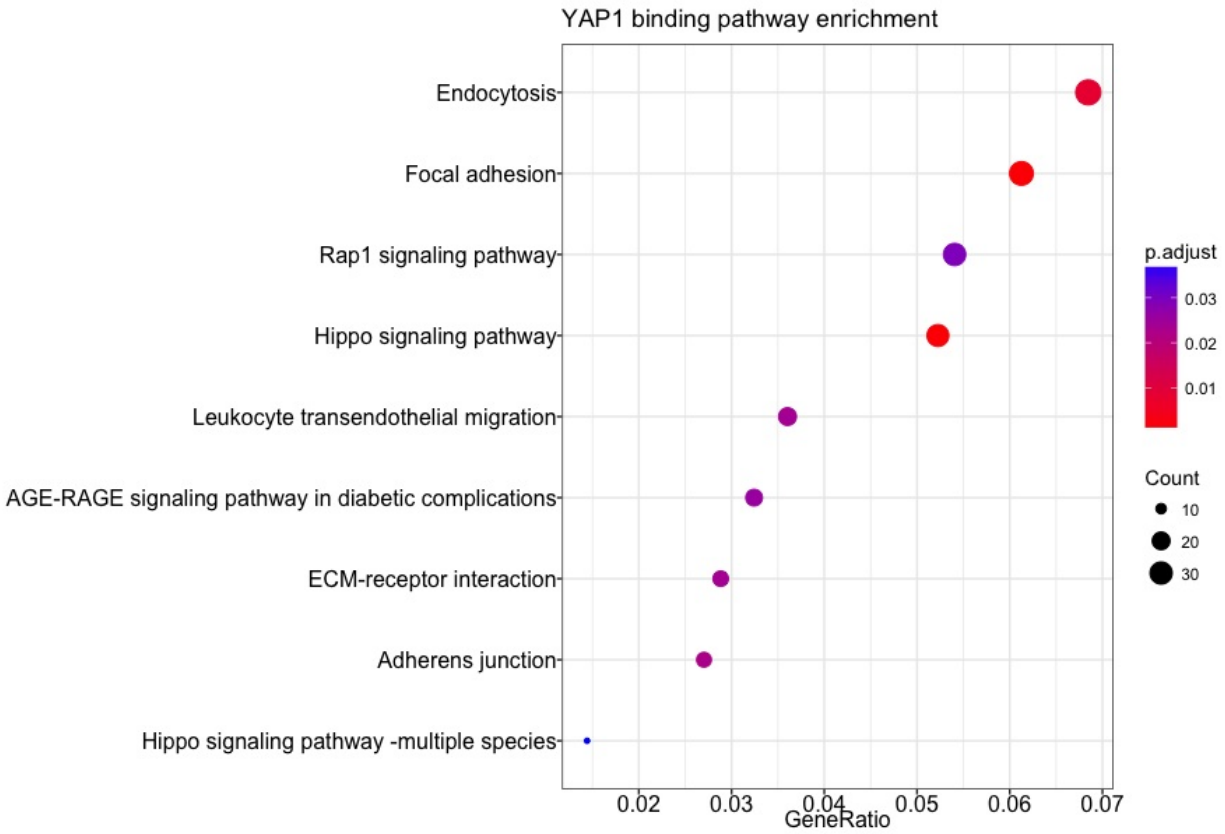
```
>
> txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene

> # read in the peaks for YAP1
> YAP1<- import("YAP1_peaks.bed", format = "BED")
> YAP1
GRanges object with 2150 ranges and 2 metadata columns:
         seqnames               ranges strand   |           name     score
            <Rle>            <IRanges>  <Rle>   |    <character> <numeric>
     [1]     chr1   [  959961,   960483]      *   |    MACS_peak_1    132.10
     [2]     chr1   [ 1295901,  1296269]      *   |    MACS_peak_2     59.44
     [3]     chr1   [ 1992556,  1993176]      *   |    MACS_peak_3     98.23
     [4]     chr1   [ 8963802,  8964418]      *   |    MACS_peak_4    126.02
     [5]     chr1   [16499970, 16500516]      *   |    MACS_peak_5    142.50
     ...      ...                   ...    ... ...            ...       ...
  [2146]     chrX [102911475, 102912040]      *   | MACS_peak_2146     73.47
  [2147]     chrX [103168254, 103168801]      *   | MACS_peak_2147     58.64
  [2148]     chrX [115403519, 115404015]      *   | MACS_peak_2148     50.08
  [2149]     chrX [149252969, 149253470]      *   | MACS_peak_2149     61.44
  [2150]     chrX [153147396, 153148074]      *   | MACS_peak_2150     61.75
  -------
  seqinfo: 23 sequences from an unspecified genome; no seqlengths

> YAP1_anno<- annotatePeak(YAP1, tssRegion=c(-3000, 3000),
                         TxDb=txdb, level = "gene", annoDb="org.Hs.eg.db",
                         sameStrand = FALSE, ignoreOverlap = FALSE,
                         overlap = "TSS")
> # some nice visualization you can do
> plotAnnoPie(YAP1_anno)
> upsetplot(YAP1_anno, vennpie=TRUE)

> # check the annotation
>  head(as.data.frame(YAP1_anno))
  seqnames    start      end width strand        name  score
1     chr1   959961   960483   523      * MACS_peak_1 132.10
2     chr1  1295901  1296269   369      * MACS_peak_2  59.44
3     chr1  1992556  1993176   621      * MACS_peak_3  98.23
4     chr1  8963802  8964418   617      * MACS_peak_4 126.02
5     chr1 16499970 16500516   547      * MACS_peak_5 142.50
6     chr1 17454948 17455379   432      * MACS_peak_6  59.60
                                annotation geneChr geneStart  geneEnd
1 Intron (uc001ack.2/375790, intron 2 of 35)       1    955503   991499
2                          Promoter (<=1kb)       1   1288071  1297157
3   Intron (uc001aiq.3/5590, intron 4 of 17)       1   1981909  2116834
4                         Distal Intergenic       1   8921059  8939151
5                         Distal Intergenic       1  16450832 16482582
6                         Distal Intergenic       1  17393256 17445948
  geneLength geneStrand geneId distanceToTSS         ENSEMBL SYMBOL
1      35997          1 375790          4458 ENSG00000188157   AGRN
2       9087          2  54587           888 ENSG00000162576  MXRA8
3     134926          1   5590         10647 ENSG00000067606  PRKCZ
4      18093          2   2023        -24651 ENSG00000074800   ENO1
5      31751          2   1969        -17388 ENSG00000142627  EPHA2
6      52693          2  11240         -9000 ENSG00000117115  PADI2
                             GENENAME
1                               agrin
2      matrix-remodelling associated 8
3                 protein kinase C zeta
4                    enolase 1, (alpha)
5                       EPH receptor A2
```

```
6 peptidyl arginine deiminase, type II

> # you can save it to a txt file to your computer
> write.table(as.data.frame(YAP1_anno), "YAP1_peaks_anno.txt", row.names =F, col.names=T, sep =
"\t", quote = F)
```

# How do I do pathway enrichment analysis for the peaks?

Now, you have the genes that are associated with the peaks, you can do pathway enrichment analysis using the genes. I will use the `clusterProfiler` package developed by Guangchuan Yu as well:

```
library(clusterProfiler)

## GO term enrichment
ego <- enrichGO(gene          = as.data.frame(YAP1_anno)$SYMBOL,
                OrgDb         = org.Hs.eg.db,
                keytype       = "SYMBOL",
                ont           = "BP",
                pAdjustMethod = "BH",
                pvalueCutoff  = 0.01,
                qvalueCutoff  = 0.05)
# visualization
dotplot(ego, showCategory = 20)


## Kegg pathway enrichment, need the Entrez ID
kk<- enrichKEGG(gene    = as.data.frame(YAP1_anno)$geneId,
           organism     = 'hsa',
           pvalueCutoff = 0.05)

dotplot(kk, showCategory = 20, title = "YAP1 binding pathway enrichment")

## you can write the result to a tsv file
write.table(kk@result, "YAP_KEGG_pathway_genes.txt", sep = "\t", col.names = T, row.names = F,
 quote =F)
```




Note that we used all the genes that associated with a peak for the pathway analysis. This may not be optimal, especially when you have a lot of peaks. One alternative is that you can filter the genes/peaks by some rules first and then do the enrichment analysis.

e.g.

```
library(dplyr)
## only consider genes within 5000 bp of the peaks
as.data.frame(YAP1_anno) %>% dplyr::filter(abs(distanceToTSS) < 5000)

## or you can rank the peaks by the Pvalue and choose the top 1000 peaks (this number is arbit
rary..)
# score column is the -10*log10 Pvalue
as.data.frame(YAP1_anno) %>% dplyr::arrange(desc(score)) %>% head(n =1000)
```

The underlying mechanism is to compare the peaks associated gene set with the vairous annotated pathway gene sets to see whether the peak assoicated genes are overpresented in any of the known gene sets using hypergeometric test. Of course, many other complex algorithums have been developed for this type of analysis. Further reading: A Comparison of Gene Set Analysis Methods in Terms of Sensitivity, Prioritization and Specificity

### GREAT predicts functions of cis-regulatory regions

## How do I do motif analysis with the peaks?

One of the most popular tools is MEME-suite for motif enrichment analysis. I will demonstrate how to use MEME-ChIP for YAP1 peaks.

`MEME-ChIP` needs 500bp DNA sequences around the YAP1 binding summits (with the highest signal of binding), which is output by `macs14` when you call peaks.

Fetching the 500 bp DNA sequence around the YAP1 summits:

```
# get the coordinates of 500bp centered on the summit of the YAP1 peaks
cat YAP1_summits.bed | awk '$2=$2-249, $3=$3+250' OFS="\t" > YAP1_500bp_summits.bed

# you need a fasta file of the whole genome, and a bed file containing the cooridnates
# the whole genome fasta file can be gotten by:
rsync -avzP rsync://hgdownload.cse.ucsc.edu/goldenPath/hg19/chromosomes/ .
cat *fa.gz > UCSC_hg19_genome.fa.gz
gunzip UCSC_hg19_genome.fa.gz

# Use betools get fasta http://bedtools.readthedocs.org/en/latest/content/tools/getfasta.html
bedtools getfasta -fi UCSC_hg19_genome.fa -bed YAP1_500bp_summits.bed -fo YAP1_500bp.fa
```

Now, upload it to `MEME-ChIP` :

# MEME-ChIP

Motif Analysis of Large Nucleotide Datasets

Version 4.11.2

**MEME-ChIP** performs **comprehensive motif analysis** (including motif discovery) on LARGE (50MB maximum) sets of sequences (typically **nucleotide**) such as those identified by ChIP-seq or CLIP-seq experiments (sample output from sequences). See this Manual for more information.

**Data Submission Form**

Perform motif discovery, motif enrichment analysis and clustering on large nucleotide datasets.

**Select the motif discovery and enrichment mode**

(•) Normal mode ( ) Discriminative mode

**Select the sequence alphabet**

Use sequences with a standard alphabet or specify a custom alphabet.

(•) DNA, RNA or Protein ( ) Custom [Choose File] No file chosen

**Input the primary sequences**

Enter the (equal-length) nucleotide sequences to be analyzed.

Upload sequences [Choose File] YAP1_500bp.fa DNA

**Input the motifs**

Select, upload or enter a set of known motifs.

Eukaryote DNA DNA

Vertebrates (In vivo and in silico)

**Input job details**

(Optional) Enter your email address.

(Optional) Enter a job description.

▶ Universal options

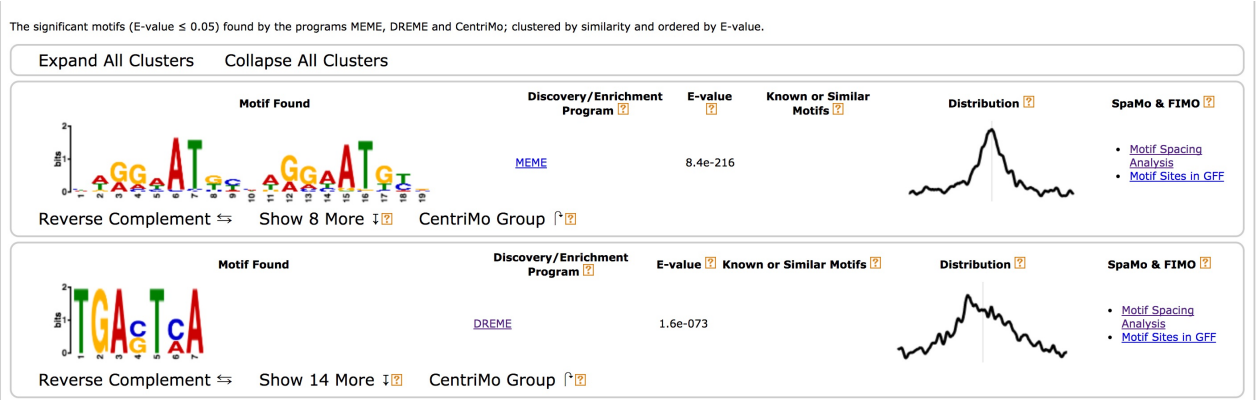
▶ MEME options

▶ DREME options

▶ CentriMo options

I will just keep the defaults for all the options. It takes a while to finish.

Output from `MEME-ChIP`:



YAP1 is not a DNA binding transcription factor itself, rather it is in a complex with TEAD, which is a DNA binding transcription factor. The first motif is a TEAD binding motif in tandem, and the second motif is AP-1 binding motif.

One downside of `MEME-ChIP` is that you can only input the DNA sequences with the same length. Instead, you can use Homer `findMotifsGenome.pl` with the length of the peaks for finding motifs:

```
findMotifsGenome.pl YAP1_filtered_peaks.bed hg19 YAP1_motifs -size given
```

| Rank | Motif | Name | P-value | log P-pvalue | q-value (Benjamini) | # Target Sequences with Motif | % of Targets Sequences with Motif |
|---|---|---|---|---|---|---|---|
| 1 | | TEAD4(TEA)/Tropoblast-Tead4-ChIP-Seq(GSE37350)/Homer | 1e-401 | -9.255e+02 | 0.0000 | 1430.0 | 66.54% |
| 2 | | TEAD(TEA)/Fibroblast-PU.1-ChIP-Seq(Unpublished)/Homer | 1e-350 | -8.066e+02 | 0.0000 | 1242.0 | 57.79% |
| 3 | | TEAD2(TEA)/Py2T-Tead2-ChIP-Seq(GSE55709)/Homer | 1e-348 | -8.035e+02 | 0.0000 | 1116.0 | 51.93% |
| 4 | | Fra1(bZIP)/BT549-Fra1-ChIP-Seq(GSE46166)/Homer | 1e-297 | -6.841e+02 | 0.0000 | 957.0 | 44.53% |
| 5 | | Atf3(bZIP)/GBM-ATF3-ChIP-Seq(GSE33912)/Homer | 1e-287 | -6.631e+02 | 0.0000 | 1037.0 | 48.26% |
| 6 | | BATF(bZIP)/Th17-BATF-ChIP-Seq(GSE39756)/Homer | 1e-284 | -6.552e+02 | 0.0000 | 1022.0 | 47.56% |
| 7 | | AP-1(bZIP)/ThioMac-PU.1-ChIP-Seq(GSE21512)/Homer | 1e-269 | -6.212e+02 | 0.0000 | 1083.0 | 50.40% |

The `Homer` results are in consistence with the `MEME-ChIP` results.

# How to call differential peaks?

One of the other frequent questions biologists ask is how the peaks change in different conditions. e.g. different treatment of the cells, different subtypes of the same cancer. To answer this question, you'd better have biological replicates of each condition.

There is a review paper on this topic: A comprehensive comparison of tools for differential ChIP-seq analysis. You can choose tools based on various rules:

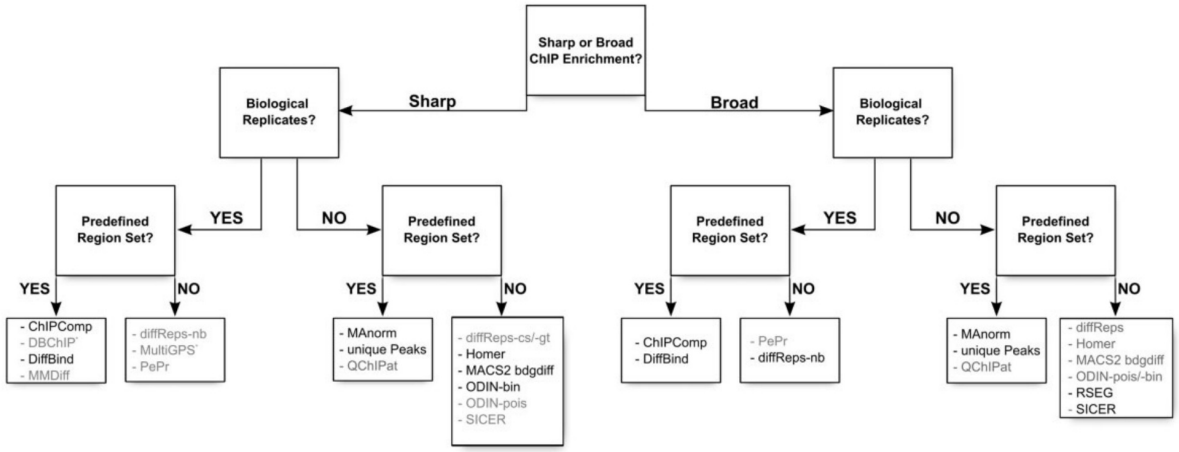



**Figure 7.** Decision tree indicating the proper choice of tool depending on the data set: shape of the signal (sharp peaks or broad enrichments), presence of replicates and presence of an external set of regions of interest. We have indicated in dark the name of the tools that give good results using default settings, and in gray the tools that would require parameter tuning to achieve optimal results: some tools suffer from an excessive number of DR (PePr, ODIN-pois), an insufficient number of DR (QChIPat, MMDiff, DBChIP) or from an imprecise definition of the DR for sharp signal (SICER, diffReps-nb). *MultiGPS has been explicitly developed for transcription factor ChIP-seq.

# ChIP-Seq Downstream Analysis 2

The author of this guide is Ming Tang. The material was developed for the Biostar Handbook.

- GitHub profile
- Diving into Genetics and Genomics

# How do I generate a heatmap with ChIP-seq data?

Heatmap is of no mystery. It is a way to visualize the data a.k.a. using colors to represent values. However, one really needs to understand the details of heatmaps. I recommend you to read Mapping quantitative data to color and Heat maps from a series of articles from *Nature Methods*.

Usually one has a matrix and then plot the matrix using R functions such as `heatmap.2`, `pheatmap` or `Heatmap`. With those R packages, it is very easy to make heatmaps, but you do need to read the documentations of the packages carefully to understand the details of the arguments. Read a tale of two heatmap functions.

For ChIP-seq data, one wants to plot ChIP-seq signal around the genomic features(promoters, CpG islands, enhancers, and gene bodies). To do this, you need to first take the regions of say 5kb upstream and downstream of all (around 30000) the transcription start sites (TSS), and then bin each 10kb region to 100 bins (100 bp per bin). Count the ChIP-seq signal (reads number) in each bin. Now, you have a data matrix of 30000 (promoters) x 100 (bin), and you can plot the matrix using a heatmap function mentioned above.

you can of course do it from scratch, but one of the taboos of bioinformatics is re-inventing the wheel. Most likely, someone has written a package for this type of analysis. Indeed, check this biostar post for all the available tools. Choose the right tool for yourself, if you are not that familiar with R, you can use some GUI tools. `EaSeq` is a pretty powerful tool for windows users.

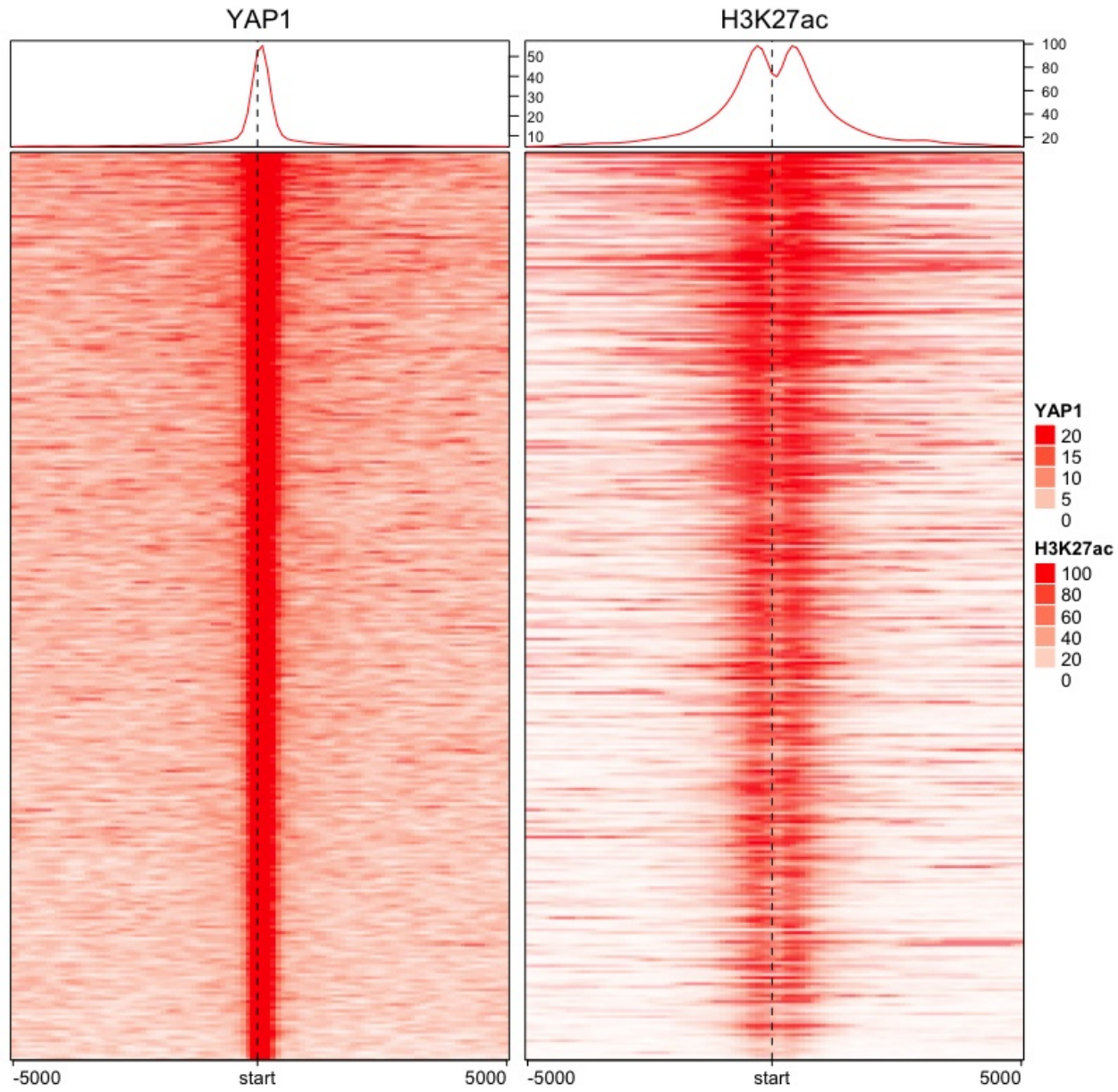
I personally like `EnrichmentHeatmap` by Zuguang Gu the most because it gives me the most flexiability to control the looks of my heatmap. It is based on `ComplexHeatmap`, which I highly recommend you to learn how to use. Below, I will walk you through how to make a heatmap with the `EnrichmentHeatmap` package.

First, let's read in the data:

```
library(EnrichedHeatmap) # for making heatmap
library(rtracklayer)  # for reading bigwig and bed files
library(GenomicRanges)
# read in the bed peak files to GRanges object
H3K27ac.bed<- import("~/ChIP-seq/results/peaks/H3K27ac_peaks.bed", format = "BED")
YAP1.bed<- import("~/ChIP-seq/results/peaks/YAP1_peaks.bed", format = "BED")


# read in bigwig files to GRanges object, it can be slow. bigwig is several hundred MB
# you can use which argument to restrict the data in certain regions.
H3K27ac.bw<- import("~/ChIP-seq/results/bams/H3K27ac.bw", format = "BigWig")
YAP1.bw<- import("~/ChIP-seq/results/YAP1.bw", format = "BigWig")
```

We want to plot the H3K27ac signal and YAP1 signal 5kb flanking the center of YAP1 peaks.

```
YAP1.10kb<- resize(YAP1.bed, width = 10000, fix = "center")

# take only the center of the peaks
YAP1.10kb.center<- resize(YAP1.10kb, width =1, fix = "center")

YAP1.mat<- normalizeToMatrix(YAP1.bw, YAP1.10kb.center, value_column = "score",
                                mean_mode="w0", w=100, extend = 5000)

H3K27ac.mat<- normalizeToMatrix(H3K27ac.bw, YAP1.10kb.center, value_column = "score",
                                mean_mode="w0", w=100, extend = 5000)

## check data range
quantile(YAP1.mat, probs = c(0.005, 0.5,0.90))
quantile(H3K27ac.mat, probs = c(0.005, 0.5,0.90))

## mapping colors
library(circlize)

## from the quantile, I choose the color mapping range
col_fun_YAP<- circlize::colorRamp2(c(0, 20), c("white", "red"))
col_fun_H3K27ac<- circlize::colorRamp2(c(0, 100), c("white", "red"))

EnrichedHeatmap(YAP1.mat, axis_name_rot = 0, name = "YAP1",
                column_title = "YAP1", use_raster = TRUE, col = col_fun_YAP,
                top_annotation = HeatmapAnnotation(lines = anno_enriched()),
                top_annotation_height = unit(2, "cm")) +
EnrichedHeatmap(H3K27ac.mat, axis_name_rot = 0, name = "H3K27ac",
                column_title = "H3K27ac", use_raster = TRUE, col = col_fun_H3K27ac,
                top_annotation = HeatmapAnnotation(lines = anno_enriched()),
                top_annotation_height = unit(2, "cm"))
```

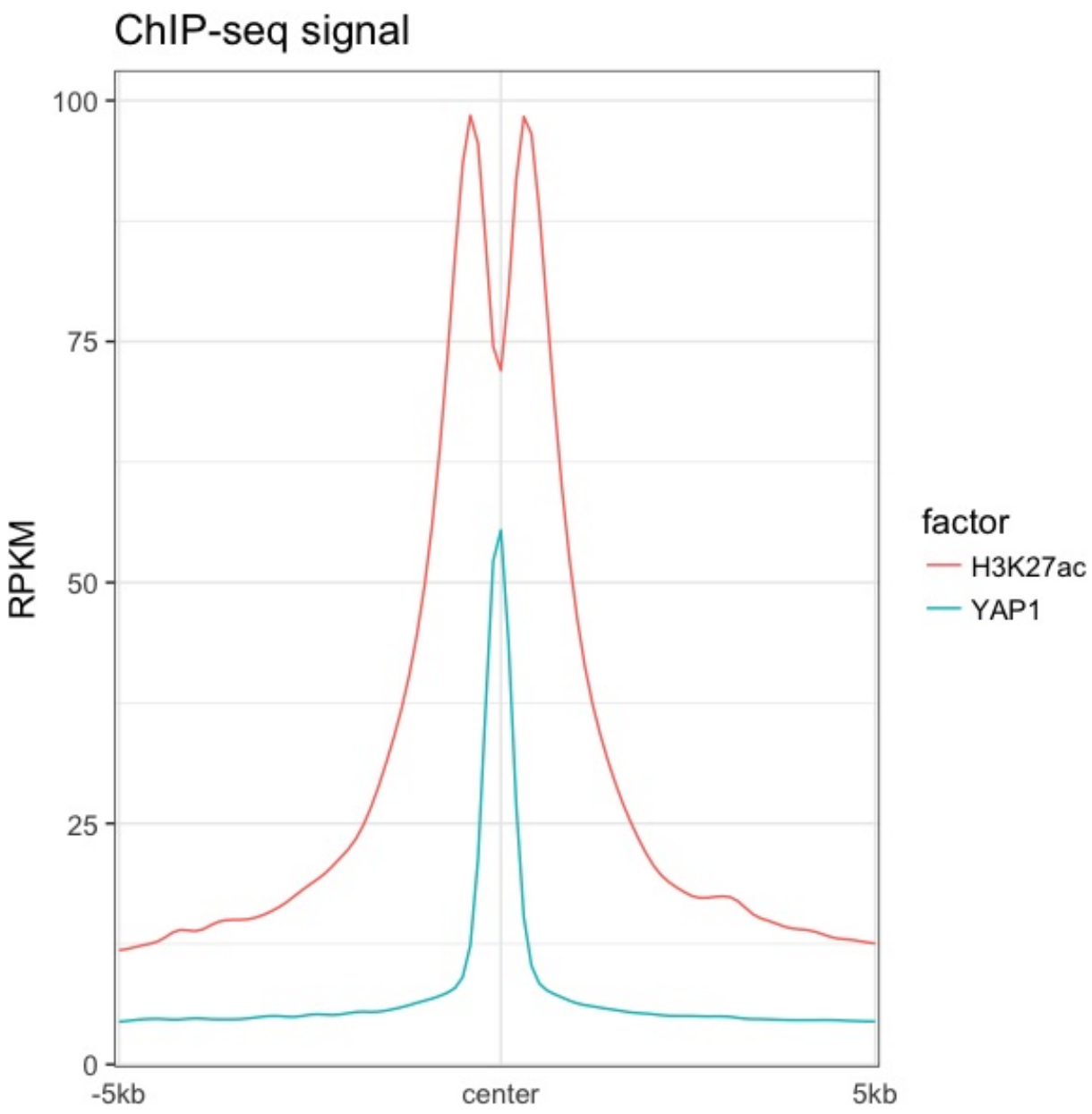


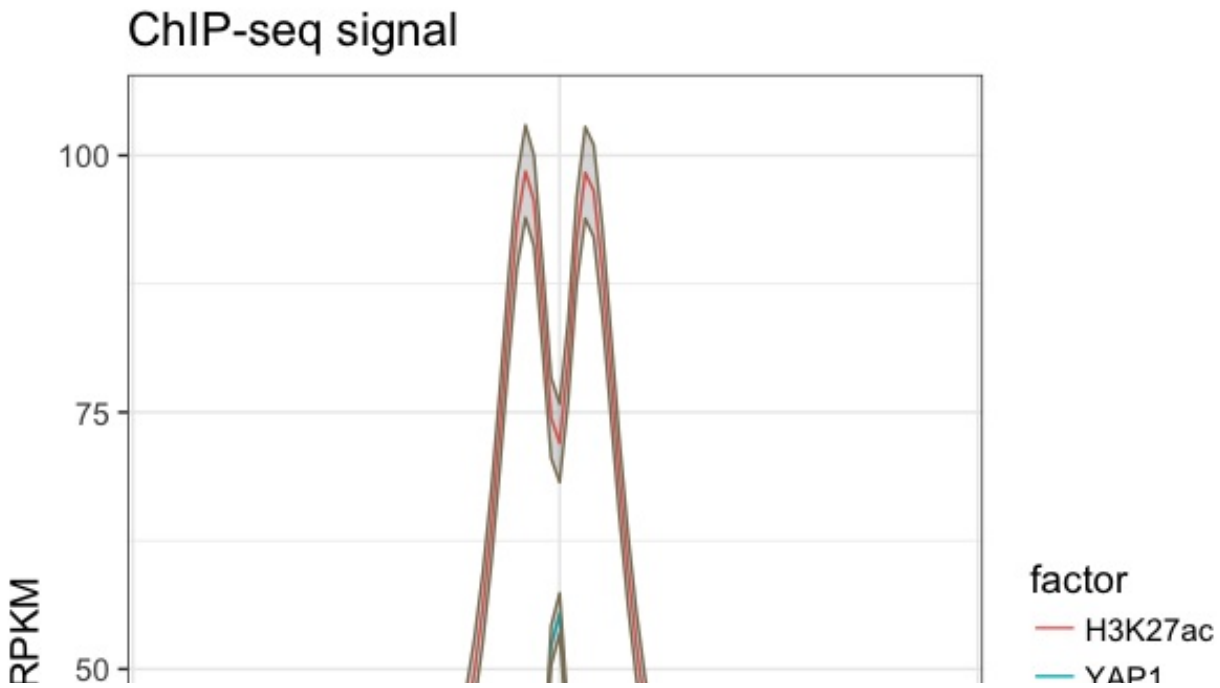

# How do I generate a meta profile plot with ChIP-seq data?

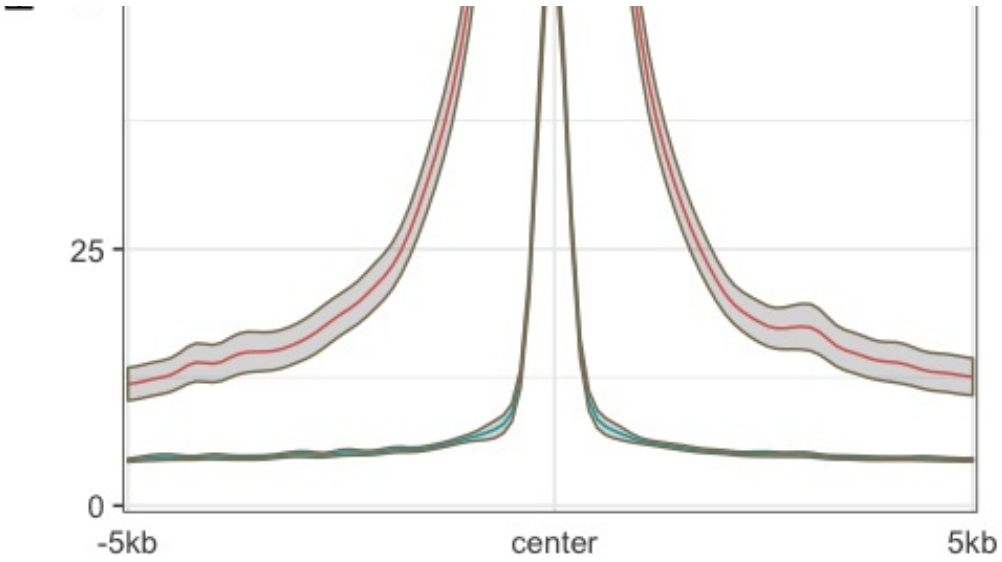
Although we used `HeatmapAnnotation(lines = anno_enriched())` to add a line plot on top of the heatmap, it may not easy to put the two line graph together. It is very common to compare signal strength for the same ChIPed factor in different conditions (treatment vs control, tumor vs normal etc). How can we do it?

The `YAP1_mat` and `H3K27ac_mat` are just like regular matrix, you can use `colMeans` to get the average and plot the average in the same figure with `ggplot2`. We can even add 95% confidence intervals onto the graph.

```
YAP1_mean<- data.frame(avg = colMeans(YAP1.mat),
                          CI_lower = apply(YAP1.mat, 2, Hmisc::smean.cl.normal)[2,],
                          CI_upper = apply(YAP1.mat, 2, Hmisc::smean.cl.normal)[3,]) %>%
        mutate(factor = "YAP1", pos = colnames(YAP1.mat))

H3K27ac_mean<- data.frame(avg = colMeans(H3K27ac.mat),
                          CI_lower = apply(H3K27ac.mat, 2, Hmisc::smean.cl.normal)[2,],
                          CI_upper = apply(H3K27ac.mat, 2, Hmisc::smean.cl.normal)[3,]) %>%
        mutate(factor = "H3K27ac", pos = colnames(H3K27ac.mat))

library(tidyverse)
combine_both<- bind_rows(YAP1_mean, H3K27ac_mean)

## change position to factor and order it
combine_both$pos<- factor(combine_both$pos, levels= YAP1_mean$pos)

## without confidence interval

ggplot(combine_both, aes(x = pos,y = avg, group = factor)) + geom_line(aes(color = factor)) +
        theme_bw(base_size = 14) +
        theme(axis.ticks.x = element_blank()) +
        scale_x_discrete(breaks = c("u1", "d1", "d50"), labels =c ("-5kb", "center", "5kb")) +
        xlab(NULL) +
        ylab("RPKM")+
        ggtitle("ChIP-seq signal")

## take some touch up to finalize the figure
ggplot(combine_both, aes(x = pos,y = avg, group = factor)) + geom_line(aes(color = factor)) +
        geom_ribbon(aes(ymin= CI_lower, ymax=CI_upper), alpha=0.2, col = "#8B7E66") +
        theme_bw(base_size = 14) +
        theme(axis.ticks.x = element_blank()) +
        scale_x_discrete(breaks = c("u1", "d1", "d50"), labels =c ("-5kb", "center", "5kb")) +
        xlab(NULL) +
        ylab("RPKM")+
        ggtitle("ChIP-seq signal")
```


ChIP-seq signal
RPKM
100
75
50
25
0
-5kb
center
5kb
factor
H3K27ac
YAP1


ChIP-seq signal
RPKM
100
75
50
factor
H3K27ac




As you can see, when you really know the power of `R` programming language, you do much more!

# Where can I get public available ChIP-seq data sets?

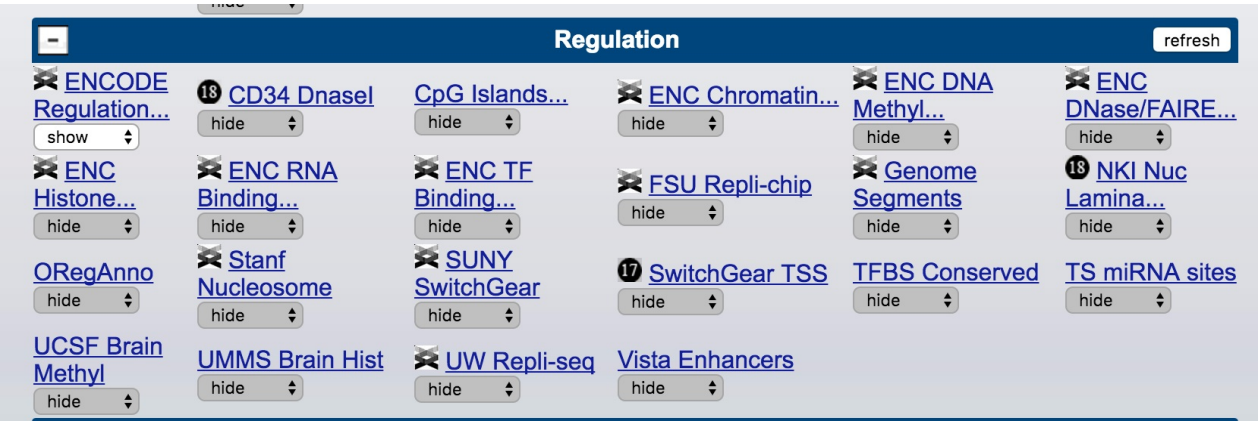
## ENCODE

It is not complete for a ChIP-seq chapter without mentioning the ENCODE project. Many ChIP-seq data sets from various cell types can be found there. In addtion, ENCODE has many other data sets such as ChIA-PET, Hi-C and DNase-seq.

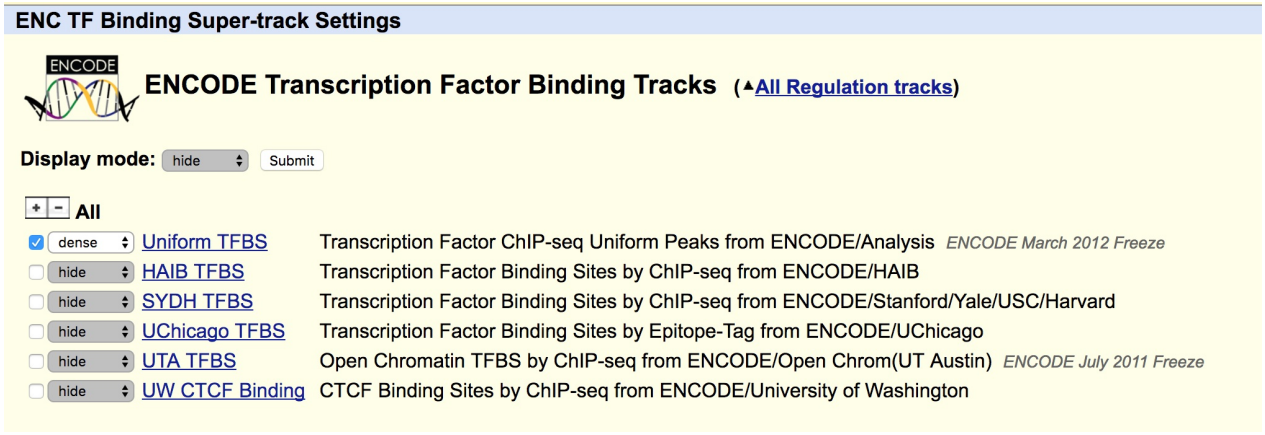
## UCSC genome browser

Every biologist need to know how to use UCSC genome browser! It is one of the most popular web genome browsers.

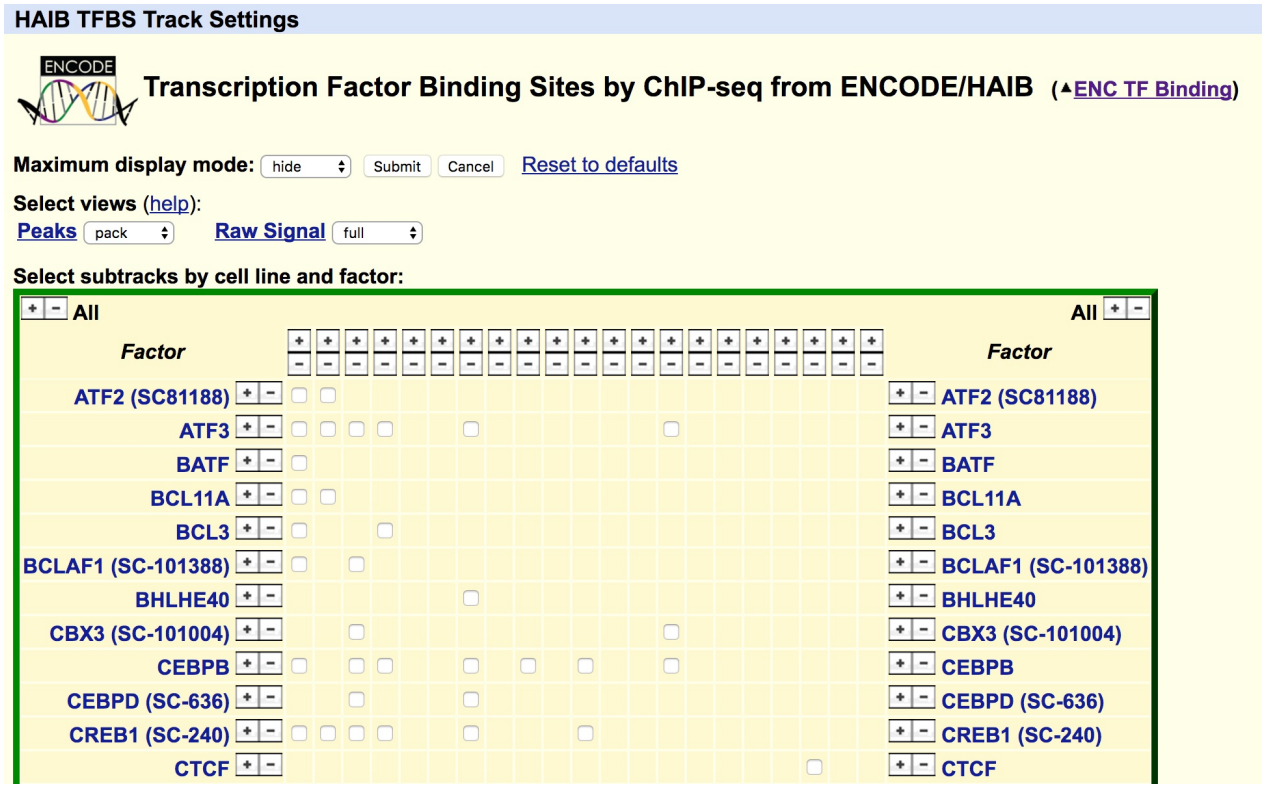
If you scroll to the `Regulation` tracks (hg19 version). There are many ChIP-seq data avaiable for you to browse.




click `ENC TF Binding`:




click `HAIB TFBS` :




Now you can check various boxes to make it show up in the browser. quite useful! Many of the data sets are from ENCODE.

I recommend you to watch the tutorials from openhelix to take full advantage of this great resource.

## Cistrome

Cistrome is a project maintained by Sheirly Liu's lab in Harvard. It has a lot more data sets including ENCODE and other public data sets from GEO. Some features are very friendly to wet biologists.

## GEO and ENA

Of course, if you want to adventure yourself with the the raw data in a publication, you can always go to GEO to get the SRA file and convert to fastq. Alternatively, you can directly download fastqs from European Nucleotide Archive (ENA).

# Introduction to Metagenomics

Microbial communities cover the entire surface of Earth, including the inside of our bodies. Life of all forms coexists with and rely on a vast variety of microorganisms. Understanding the role, function and purpose of microbial species are essential to understanding higher order organisms.

Through environmental sequencing, scientists have come to realize that the level of functional and taxonomic diversity in the earth’s microbial communities is much greater than previously anticipated.

Metagenomics analyses are more challenging than other sequencing data analysis because the skills required to succeed will swing wildly across the different domains of science even more abruptly than you might expect. Do you need to be a programmer, a biologist, a statistician, an ecologist, a microbiologist to succeed? Perhaps all of them at the same time. And while wearing many hats is not an unexpected requirement in bioinformatics where metagenomics will suprise you is that decision making is exceedingly subjective and choices made along the way have a pronounced impact on the results.

## What is metagenomics?

Taking the definition from Wikipedia:

> Metagenomics is the study of genetic material recovered directly from environmental samples. The broad field may also be referred to as environmental genomics, ecogenomics or community genomic

From a bioinformatician's point of view, metagenomics is the application of high-throughput genomic techniques to the study of communities of microbial organisms.

## What are the challenges of metagenomics?

Traditionally genetics has focused on *vertical transfer* of genetic material: from parent to offspring. There is another mechanism of evolution, the so called *horizontal gene transfer* that allows some types of organisms to share genetic material *across* different species. Among single-celled organisms, it is perhaps the dominant form of genetic transfer.

Consequently, the variety and diversity of the genomic content of microorganisms is incomparably larger than that of higher order organisms, and cannot easily be modeled in the traditional top-down, across generation models. Also whereas in higher order organisms the phenotypes are well defined within a taxonomic level, bacteria from the same genera can manifest widely different functions.

## Do humans host ten times as many bacteria as their cells?

Crack open any book on metagenomics there is a good chance that a statement of the following nature will greet you:

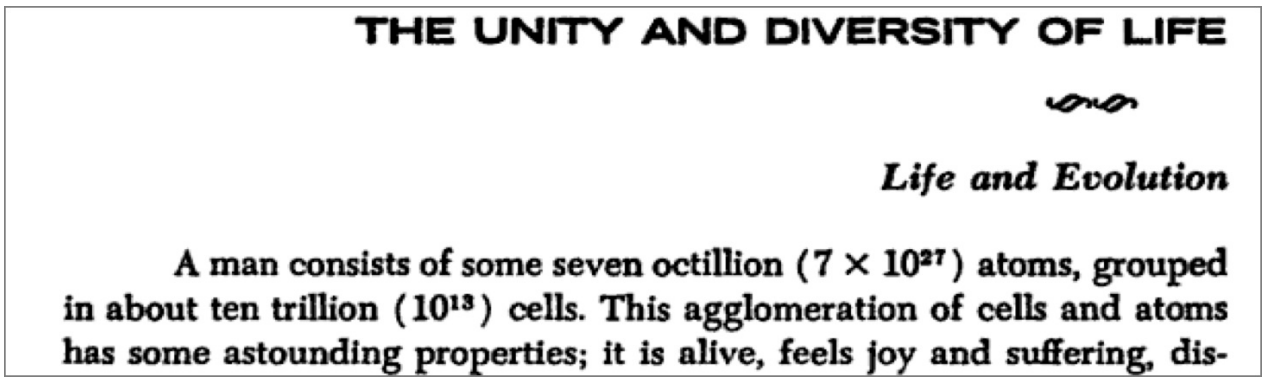
> ... humans have more bacterial cells inhabiting our body than our own cells ...

For example as seen in A Primer on Metagenomics, PLoS Comp Biology, 2010.

> profoundly affected by microbes, from the scourges of human, farm animal, and crop pandemics, to the benefits in agriculture, food industry, and medicine to name a few. We humans have more bacterial cells ($10^{14}$) inhabiting our body than our own cells ($10^{13}$) [2],[3]. It has been stated that the key to understanding the human condition lies in understanding the human genome [4],[5]. But given our intimate relationship with microbes [6], researching the

This claim often surfaces as a favorite introductory quote to presentations on metagenomics.

Amusingly, tracking down the origin of the numbers in this statement (10^13 = trillion, 10^4 = ten trillion) is a mixture of amusement and incredulity. It appears that most scientists keep borrowing this statement from one another, repeating it without making any effort to ensure that it is true. Each paper cites another paper, stretching back to seemingly the beginning of modern biology. For example, the A Primer on Metagenomics, PLoS Comp Biology, 2010 cites a paper from 1996 in "Trends Microbiol" titled "The indigenous gastrointestinal microflora" as the origin of the statement above. But this cited article is not the source of this number. Instead, it repeats the claim and cites another source for from 1977 titled "Microbial ecology of the gastrointestinal tract." But that paper is not the actual source either; it refers the reader to the origin of the number to another publication from 1972 in "American Journal of Clinical Nutrition" titled "Introduction to intestinal microecology." Unsurprisingly this paper, in turn, cites a book from 1971 titled "Genetics of the evolutionary process" by Theodosius Grigorievich Dobzhansky. It is in the introduction of that book, in the very first sentence actually where Dobzhansky states that the human body contains ten trillion cells. There is no indication as to where he got that number from, this estimate (an opinion really) has been cited as a fact ever since.



THE UNITY AND DIVERSITY OF LIFE

*Life and Evolution*

A man consists of some seven octillion ($7 \times 10^{27}$) atoms, grouped in about ten trillion ($10^{13}$) cells. This agglomeration of cells and atoms has some astounding properties; it is alive, feels joy and suffering, dis-

Jordan Anaya has series of blog posts that demonstrate the absurdity of this scientific "habit" that is unfortunately not as rare as one might hope. He calls the practice russian-doll citations that often, like above culminate in a so called "rick-roll" citation. The origin for the estimate of the bacterial numbers is similarly lengthy and equally unreliable. A more recent estimate states that Body's bacteria don't outnumber human cells so much after all

Why even mention this here? It is an example and a reminder that you will need to approach every statement and discovery that has to do with metagenomics with a very healthy dose of skepticism. Yes, the field is fascinating, yes the field has a lot of potential - but at the same time it is also a domain that is

rife with incorrect assumptions, flawed analyses, hand-waving arguments, grandiose claims, intriguingly sounding yet meaningless "discoveries."

# What type of answers does metagenomics produce?

Metagenomics is the science of answering questions that fall into the following categories:

1. Which microorganisms are present in a sample?
2. What abundance (quantity) is each microorganism present in?
3. How do the observed abundances change across conditions?
4. What are the functions of the microorganisms that are observed?

Important: It is essential to recognize from the very beginning that most microorganisms cannot exist independently and their survival requires the presence of diverse communities of other microbes. Thus we need to keep in mind that organisms need to be treated in the context of their communities rather than individual species

# What are the metagenomics approaches?

Current practices in metagenomics fall into the categories of:

1. 16S rRNA sequencing
2. Whole-metagenome sequencing

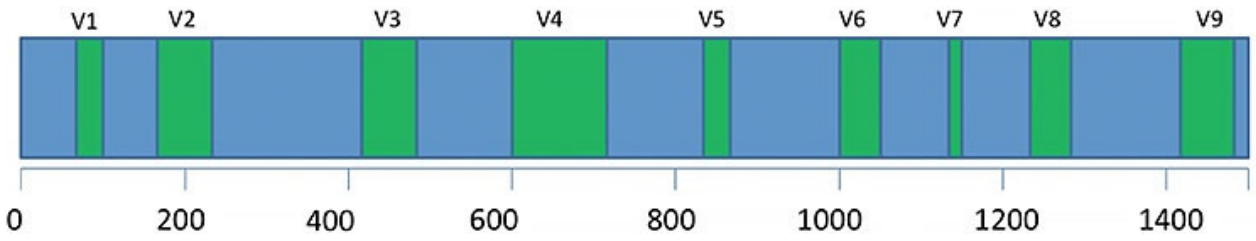
# What is 16S sequencing?

From the Wikipedia page on 16S Sequencing

> 16S ribosomal RNA (or 16S rRNA) is the component of the 30S small subunit of a prokaryotic ribosome []. The genes coding for it are referred to as 16S rRNA gene and are used in reconstructing phylogenies, due to the slow rates of evolution of this region of the gene.

In a nutshell the 16S gene forms a peculiar part of a genome that has both very highly conserved regions (identical across just about all bacteria) as well as highly variable regions (hypervariable) that may be specific to a taxonomical level such as genus or species. You can use the conserved regions to isolate the variable regions that may be used to identify the bacteria. Since only a tiny section of the entire bacteria is genome will be sequenced, the 16S method is very "economical" requires a fraction of the coverage and produces far lower quantities of data. The analysis methods are simpler, more efficient and more focused.

The 16S rRNA gene is about 1500bp long the nine hypervariable regions labeled as `V1-V9` range from about 30-100 base pairs long. Due to sequencing technology read length limitations, most 16S methods operate on just a single one of these regions at any given time. For example `V4`. Picking the "best" region is a source of lots of (somewhat pointless) discussions.




# What is whole-genome metagenomics?

This method is identical to conventional whole genome sequencing. The mixture of DNA that originates from many different organisms is randomly sheared into smaller fragments (100-600bp) that are then sequenced. Of course unlike the situations where the origin of the DNA is from a single organism a metagenomic will represent a mixture of DNA. The main complication arises from following factors:

1. Each organism may be present in wildly different abundances.
2. The sequencing represents random sample.
3. All measures are relative to the total number of DNA fragments.

# How many bacteria have been fully sequenced?

In a subsequent chapter, we will see how to get this data. For now, we'll say that NCBI stores 7318 fully sequenced bacterial genomes. There are much more sequences of course that are known to correspond to bacteria but have not yet been assembled into a single genome.

The NCBI bacterial taxonomy currently labels 65,231 species under the superkingdom of bacteria.

# How many bacteria are unknown?

Scientists believe that the vast majority of bacteria are yet unknown. Depending on who you ask large numbers potential species are considered to exist, sometimes ranging into millions. Notably when the scientist providing the estimate is involved in so called "big data analysis" their assessment tends to be very large (billions).You may even notice an interesting pattern,

The publication from 2004: Status of the Microbial Census, Microbiolgy and Molecular Biology Reviews, 2014 states that:

> Estimating the diversity of life is a persistent challenge in biology. In microbiology, the task is complicated by the fact that the subjects of the census are not visible to the naked eye or easily differentiated morphologically, and they are estimated to number over $10^{30}$ individual bacteria worldwide (**30**). The properties of microorganisms necessitate the use of indirect

and it cites a paper from 1998 titled "Prokaryotes: the unseen majority, PNAS, 1998" that in turn states that:

> The subsurface is defined here as terrestrial habitats below 8 m and marine sediments below 10 cm. Few direct enumerations of subsurface prokaryotes have been made, largely because of the difficulty in obtaining uncontaminated samples. Nevertheless, circumstantial evidence suggests that the subsurface biomass of prokaryotes is enormous (20). For instance, groundwater from deep aquifers and formation water from petroleum deposits contain $10^3$–$10^6$ prokaryotic cells/ml (21, 22).

and cites a paper from 1994 titled "The deep hot biosphere, PNAS, 2014" and so on. The pattern might be familiar now, as the citations go back in time each publication is a little less rigorous and a little more speculative. The problem here is that what was acceptable as speculation in the early 1990s has, by the virtue of being cited over and over, become "a fact".

Now, we do need to recognize that bacteria are subject to more diverse mechanisms for evolution. It makes sense to assume that there would be more variety to them. The confounding factor is that there is no definition of when two bacteria should be labeled as different species. Discussing this distinction is beyond the scope of this book. Interested readers consult chapters from: Chapter 3: Classification in Medical Microbiology. 4th edition. or the many other resources.

In summary, there are many inconsistencies when trying to place labels on organisms as diverse as bacteria.

# How to analyze metagenomics data?

## What type of analyses will I need toperform?

Metagenomics analyses span the whole gamut from well-defined tasks of the classification category i.e. "what is in my pot" to the highly speculative tasks where you may be called upon to evaluate the "metabolic potential" of metagenomes. Whereas the former job can be objectively evaluated, the success of the second types of analyses will strongly depend not just on your results but perhaps more so on your persuasion skills.

## What online tools can I use?

Metagenomics is a unique field in that most of the time its results need to characterize hundreds if not thousands of species. For that reason, visualization takes a central role. Unlike other fields of bioinformatics, there are several alternatives for online analysis and visualization can be performed. All these choices can also be confusing. It is almost never clear exactly what type of studies a given site offers - they all seem to promise "everything" yet the implementation details vary considerably. Only after you upload the data do you get to see the limitations of the tools. For that reason, we firmly prefer command line tools though these add substantial overhead and complexity to the analysis.

Another factor to consider is that most online tools are developed to operate on a single dataset. Most scientists collect dozens if not more samples for each experiment. Processing the multitude of samples efficiently poses radically different challenges that online services may not be able to handle.

We will mention a few online tools, then in next chapters, we will perform operations with command line tools.

- MGRAST: Metagenomics Analysis Server
- VIROME
- QIITA
- EBI Metagenomics

## How can I quantify and compare results?

Terms such as alpha, beta diversity were introduced to describe biodiversity. The alpha diversity is the species diversity at a local scale (same site). The beta diversity represents differences in species composition among different sites.

Do note however that the mathematical definitions behind these words are not uniquely defined. There are several ways to calculate each. You should think in terms of categories such as:

1. Community richness
2. Community evenness
3. Community diversity

4. Shared community richness
5. Similarity in community membership
6. Similarity in community structure
7. ...

Each category, in turn, can have one or more formulas that capture one dimension of the concept. See for example the [mothur calculators][calculator] that the above listing was taken from. As you can see there, for example, the **Berker Parker Index** is calculated as the ratio of the abundance of the dominant OTU divided by the total number of individuals in the sample. It is a fraction that represents how much of the total does the most abundant species correspond to.

The BP index is a single number and you should think of it as one, particular characteristic of the data. There are several more that you could (and should) compute. A Simpson index, a Jaccard similarity, etc. Picking the right choice depends a great deal on what your data looks like and what changes take place. Therefore you need to pick the right "calculator" for the job.

# What command line tools may be used to analyze metagenomics data?

The metagenomics command line analysis communities are split into two categories:

1. QIIME
2. mothur

See the Biostar question of the day Qiime Vs Mothur : Why Use One Over The Other?. Here is what I wrote as an answer a few years ago about them, this distinction still holds well:

> There is a fundamental almost philosophical difference in how the tools are developed
>
> **Mothur** is a single program that re-implements a large number of very useful algorithms into a single, high performance standalone executable program for each platform: linux, mac and windows. In my opinion, it is one of the most amazing feats of bioformatics software engineering especially considering that is being developed by only two people.
>
> **QIIME** is a python interface (glue) that connects a very large number of disparate programs and what QIIME really does is transforms the input/outputs of these and allows you to feed one program into the other. You will need to install a very large number of dependent programs to use QIIME. In fact it is not even correct to say that you used QIIME to compute something, you are using the programs that QIIME runs for you: `pynast`, `uclust` etc.
>
> Now because in QIIME each program is developed independently some by entire groups some of these may offer higher performance than what is implemented in mothur. On the other hand the unified interfaces of mothur make it far more consistent and better documented not to mention easier to run.

# What are typical steps of an analysis?

Tools like Mothur and QIIME offer detailed instuctions that you can follow. For example the Mothur SOP (Standard Operating Protocol) for MiSeq data lists and describes in great detail the following steps:

1. Reducing sequencing and PCR errors
2. Processing improved sequences
3. Assessing error rates
4. Processing OTUs
    - Calling OTUs
    - Phylotypes
    - Phylogenetic Analysis
5. Diversity
    - Alpha diversity
    - Beta diversity measurements
    - Population-level analysis
6. Phylotype-based analysis
7. Phylogeny-based analysis

We recommend following the instructions as described there.

# Taxonomies

## What is a taxonomy?

The "taxonomy" is the science of defining and naming groups of biological organisms by shared characteristics. Swedish botanist Carl Linnaeus ushered a new era of taxonomy in the 18th century when he proposed to characterize organisms into a hierarchical scheme:

```
Domain -> Kingdom -> Phylum -> Class -> Order -> Family -> Genus -> Species
```

But then what was an excellent idea in the 18th century for classifying higher order organisms is less suited to the needs of the 21st century when we need to describe microorganisms. The Linnean scheme is not appropriate for characterizing bacterial species where the spectra of differences go beyond the top-down, vertical, hierarchical representation. For that reason, the classification of a bacteria into any given taxonomical rank is far less informative when it comes to the function and phenotype.

The flaws of the taxonomical classification have led to creating several alternatives versions to them - for an outsider this only adds to the confusion.

## What the heck is an OTU?

As soon as you get into the field, you'll soon realize that the word OTU is deeply embedded into the vocabulary of meta-genomicists - yet many of them will struggle to explain what an OTU is or is not. Other words are built by using the word OTU:

- OTU calling
- OTU clustering

etc. many suffering from the same level ambiguity. Common misunderstanding includes believing that the term has a more substantial meaning than what it stands for, as well as believing that finding an OTU is the result of a well determined, unique process.

The word OTU stands for "Operational Taxonomic Unit." The term itself is almost meaningless: it was originally defined as the taxonomic unit that was being studied, "operated" with. Obviously, this is a very loose definition. The meaning got narrowed when the OTU started to represent "things" that were also "similar." Later the concept of "similarity" grew to represent similarity over a marker gene, such as the 16S gene.

Today in most parlance the OTU accounts for a grouping of sequences based on their sequence similarity. An OTU group does imply that the sequences have identical (or even similar) taxonomical assignments, though there may be a representative taxonomical classification that gets assigned to represent an OTU. Further complications arise as well when sequences could belong equally well in more than one OTU or many very different clades.

# How many taxonomies are available?

Too many. Picking and sticking to a taxonomy appears to have an almost tribal quality to it. For a more detailed discussion see SILVA, RDP, Greengenes, NCBI and OTT — how do these taxonomies compare? in BMC Genomics 2017 where the authors conclude that:

> While we find that SILVA, RDP and Greengenes map well into NCBI, and all four map well into the OTT, mapping the two larger taxonomies on to the smaller ones is problematic.

# Which taxonomy should I use?

We typically use the NCBI taxonomy as it makes it a lot easier to connect to other data sources. There are also command line tools from `edirect` to `taxonkit` that were designed to interact with the taxonomy files developed by NCBI.

Scientists (while few admit that the problem exists) can be biased when it comes to their pet projects. In those cases, it is best to go with the flow. Go ahead and read a few papers and see what taxonomy is commonly used in the field that you need to publish in and choose that.

# How to view the NCBI taxonomy?

The NCBI web site at https://www.ncbi.nlm.nih.gov/taxonomy provides access to a visual display of the taxonomy.

# What is the story with the NCBI disclaimer?

Most pages on the NCBI taxonomy will show you the following "disclaimer":

> Disclaimer: The NCBI taxonomy database is not an authoritative source for nomenclature or classification - please consult the relevant scientific literature for the most reliable information.

Thanks for clearing that up NCBI (note: sarcasm). May we humbly ask though, if you are *not authoritative* then why are you storing and distributing this data at all? And if you know so well that you are not the *authoritative source* then what is the *authoritative* resource? Why are you punting the responsibility back to us, the readers?

Mocking aside the way we read this is that making taxonomies turned out to be somewhat subjective and controversial. NCBI tries to fend off any criticism by pointing to this disclaimer.

# How to get the NCBI taxonomy file?

NCBI states that it has phased out the `gi` numbers even though some tools rely on this information. The "old-style" tools require `gi` to `taxid` type of information whereas the "new-style" tools work on `accession` to `taxid` mapping.

```
URL=https://ftp.ncbi.nlm.nih.gov/pub/taxonomy/accession2taxid/nucl_gb.accession2taxid.gz
curl -k $URL | gunzip -c > acc2taxid.txt
```

It contains a mapping of gi number to taxonomy id.

```
cat acc2taxid.txt | head
```

It prints:

```
accession accession.version taxid gi
A00002 A00002.1 9913 2
A00003 A00003.1 9913 3
X17276 X17276.1 9646 4
X60065 X60065.1 9913 5
```

The "old-style" files are still available and approximately 1.5Gb in size and it was last updated:

```
URL=https://ftp.ncbi.nlm.nih.gov/pub/taxonomy/gi_taxid_nucl.dmp.gz
curl -O $URL
```

It contains a mapping of gi number to taxonomy id.

```
gzcat gi_taxid_nucl.dmp.gz | head
```

like so:

```
2 9913
3 9913
4 9646
5 9913
7 9913
...
```

The old style files may cease to be updated at some point (we are not even sure if that has already come). Still, as of now, there may be tools that need the information in this old format. Thankfully we can "convert" a new style file to the old-style with:

```
cat acc2taxid.txt | grep -v taxid | awk ' { print $4,"\t", $3 } ' | head
```

to produce:

```
2   9913
3   9913
4   9646
5   9913
11   9913
```

# How many taxonomical ranks are there?

Whereas the Linnean taxonomy prescribed nine ranks for the taxonomy the NCBI data has a lot many more:

```
URL=https://ftp.ncbi.nlm.nih.gov/pub/taxonomy/taxdump.tar.gz
wget $URL
tar zxvf taxdump.tar.gz

cat nodes.dmp | cut -f 3 -d '|' | uniq -c | sort -rn
```

that produces:

```
1281994   species
217328    no rank
82878     genus
20883     subspecies
8594      family
6826      varietas
2775      subfamily
1990      tribe
1420      order
1314      subgenus
 807      superfamily
 480      subtribe
 474      forma
 438      species group
 331      suborder
 309      class
 220      phylum
 133      species subgroup
 132      subclass
 104      infraorder
  49      superorder
  28      subphylum
  15      infraclass
  10      parvorder
   5      superkingdom
   4      superclass
   3      kingdom
   3      cohort
   2      superphylum
   1      subkingdom
```

Quite a few many more ranks than you might expect.

# How do I search the NCBI taxonomy from the command line?

## Access taxonomy with EDirect

For some use cases, searching NCBI with Entrez Direct may be sufficient. For other more advanced functionality we recommend the `taxonkit` command line program.

# How do I search the taxonomies?

What does the taxid `9913` stand for?

```
fetch -db taxonomy -id 9913
```

it will print:

```
1. Bos taurus
    (cattle), species, even-toed ungulates
```

## Access taxonomy with TaxonKit

```
taxonkit list --show-rank --show-name --ids 9913
```

prints:

```
9913 [species] Bos taurus
```

# How do I find the taxid of a taxonomy name?

## Taxonomy name with EDirect

For this, to work you'd have to (reasonably) correctly spell the scientific name then `esearch` will work:

```
esearch -query "E. coli[txn]" -db taxonomy | efetch
```

prints:

```
562
```

## Taxonomy name with TaxonKit

You have to specify the exact name with taxonkit:

```
echo Escherichia coli | taxonkit name2taxid
```

prints:

```
Escherichia coli    562
```

As you can see above `taxonkit` is designed to work with streams. If you had a file that contains:

```
Archaea
Bacteria
Eukaryota
Viroids
Viruses
other sequences
unclassified sequences
```

you could get the taxid of each term with:

```
cat names.txt | taxonkit name2taxid
```

to obtain:

```
Archaea                  2157
Bacteria                    2
Eukaryota                2759
Viroids                 12884
Viruses                 10239
other sequences         28384
unclassified sequences  12908
```

# How do I get the taxid for an accession number?

```
esearch -db nuccore -query NC_009565,NC_002570,NC_008358 | elink -target taxonomy | efetch
# 336982
# 272558
# 228405


efetch -db taxonomy -id 336982,272558,228405
# 1. Mycobacterium tuberculosis F11
#   high GC Gram+
# 2. Bacillus halodurans C-125
#    firmicutes
# 3. Hyphomonas neptunium ATCC 15444
#    a-proteobacteria
```

# How do I get the lineage of a taxid?

## Lineage with EDirect

```
efetch -db taxonomy -id 562 --format xml > out.xml
```

will produce quite the output. You can process that XML output with `xtract` or other tools:

```
cat output.xml | xtract -pattern LineageEx/Taxon -element TaxId,ScientificName
```

prints:

```
131567 cellular organisms
2 Bacteria
1224 Proteobacteria
1236 Gammaproteobacteria
91347 Enterobacterales
543 Enterobacteriaceae
561 Escherichia
```

you can also format it in a row oriented way with:

```
cat output.xml | xtract -pattern TaxaSet/Taxon -element TaxId,ScientificName
```

that prints:

```
562    131567    2    1224    1236    91347    543    561    Escherichia coli    cellular orga
nisms    Bacteria    Proteobacteria    Gammaproteobacteria    Enterobacterales    Enterobacter
iaceae    Escherichia
```

## Lineage with TaxonKit

```
echo 9913 | taxonkit lineage
```

prints:

```
9913    cellular organisms;Eukaryota;Opisthokonta;Metazoa;Eumetazoa;Bilateria;Deuterostomia;Ch
ordata;Craniata;Vertebrata;Gnathostomata;Teleostomi;Euteleostomi;Sarcopterygii;Dipnotetrapodom
orpha;Tetrapoda;Amniota;Mammalia;Theria;Eutheria;Boreoeutheria;Laurasiatheria;Cetartiodactyla;
Ruminantia;Pecora;Bovidae;Bovinae;Bos;Bos taurus
```

# How do I list all taxids of a given rank?

This command lists all the taxonomies that correspond to the viruses (taxid 10239)

```
taxonkit list --ids 10239 --indent ""  | head
```

Will print:

```
10239
12333
12340
12347
12366
12371
...
```

If you were to store the above query into a file called `virus.taxid.txt` you would see how many taxonomical ranks do virii have:

```
wc -l virus.taxid.txt
5285 virus.taxid.txt
```

# How many species of viruses are in the current taxonomy?

```
taxonkit list --ids 10239 --indent "" --show-rank --show-name  | grep species | head
```

prints:

```
12340 [species] Enterobacteria phage 933J
12347 [species] Actinophage JHJ-1
12366 [species] Streptococcus pyogenes phage H4489A
12371 [species] Phage h30
12374 [species] Lactococcus phage
12375 [species] Lactococcus phage (ISOLATE 7-9)
12386 [species] Lactococcus phage mi7-9
12388 [species] Mycobacterium phage FRAT1
12392 [species] Lactobacillus phage mv4
12403 [species] Leuconostoc phage P32
```

or if you want to count them:

```
taxonkit list --ids 10239 --indent "" --show-rank --show-name  | grep species | wc -l
```

prints:

```
4824
```

The answer is that NCBI knows of `4824` virus species distributed over `5285` ranks.

# How do I obtain sequences based on taxonomies?

## How do I get all known bacterial genomes?

It used to be that obtaining all known bacterial genomes from NCBI was as easy as downloading a prebuilt FASTA file that contained these genomes. Alas this practice has been discontinued and replaced with a somewhat convoluted series of actions:

```
# Get the summary as a tabular text file.
curl -O ftp://ftp.ncbi.nlm.nih.gov/genomes/refseq/bacteria/assembly_summary.txt

# Filter for complete genomes.
awk -F "\t" '$12=="Complete Genome" && $11=="latest"{print $20}' assembly_summary.txt > ftpdir
paths

# Identify the FASTA files (.fna.) other files may also be downloaded here.
awk 'BEGIN{FS=OFS="/";filesuffix="genomic.fna.gz"}{ftpdir=$0;asm=$10;file=asm"_"filesuffix;pri
nt ftpdir,file}' ftpdirpaths > ftpfilepaths
```

Moreover at this point the official NCBI instructions basically punt the task right back to you the reader and tell you to "use a script" to download the data for the URLs collected into this file. Here is a way that we you can do:

```
mkdir all
cat ftpfilepaths | parallel -j 20 --verbose --progress "cd all && curl -O {}"
```

The above will put all the genomes into the `all` directory that now contains. Do note that the process may take many hours.

```
GCF_000005825.2_ASM582v2_genomic.fna.gz
GCF_000005845.2_ASM584v2_genomic.fna.gz
GCF_000006605.1_ASM660v1_genomic.fna.gz
...
```

At this point the folder contains sufficiently large number of files that shell meta-charater expansion won't work anymore and you will get errors like the following:

```
ls -l all/*.gz
-bash: /bin/ls: Argument list too long
```

This is because the shell attempts to expand the pattern and list all the matching files *before* executing the command. You can still listing all files:

```
ls -1 | wc -l
7182
```

Due to the limitation mentioend before if you wanted to sub-select files with a certain meta-character you would need to use `find` command and pipe the results into the next action:

```
find all -name '*fna.gz' | wc -l
7182
```

# Are there other ways to get all bacterial/viral genomes?

Probably, though you should not get your hopes too high. On this topic there are many tutorials and guides yet it appears that most advice becomes obsoleted at a rapid pace. NCBI appears to move around the files quite a bit while also changing the data naming and formatting guidelines. In just 4 month the advice in the Biostar Question of the Day: Extract all bacteria sequences from the nr database has become invalid.

NCBI offers advice and support services that may be of some use - though we can't help but wonder of why the processes need to be so complicated. Below is an "official" answer for the question on how to obtain all viral genomes from NCBI:

> Dear Colleague,
>
> NCBI processes submitted (GenBank/INSDC) viral genomes differently than prokaryotic/eukaryotic genomes:
>
> - In general, we do not create Assembly records for the primary (submitted) genomes. As you have already established there is /genomes/refseq/viral path on the Genomes FTP site: ftp://ftp.ncbi.nlm.nih.gov/genomes/refseq/viral/ that is populated with assemblies, while the genomes/genbank/viral path has a negligible content (that is coming from viral metagenomes, an oddball of environmental data that can't be made into RefSeq by RefSeq's standards): ftp://ftp.ncbi.nlm.nih.gov/genomes/genbank/viral/ -NCBI generates RefSeqs only for selected GenBank records, so to get coverage for each viral species: at least one assembly, sometimes more. The good thing is that part of the RefSeq processing is to calculate (find) GenBank neighbors for the RefSeq entries. The total of the neighbors approximates the number of complete GenBank/INSDC genomes.
> - You can retrieve genome neighbors on the web by using one of the two methods that are described in this viral genomes resource FAQ: https://www.ncbi.nlm.nih.gov/genome/viruses/about/HowTo/#retrieve_neighbors Note that the number of displayed records will be larger than the number of the actual genomes, because the genomes for some organisms are multi-segmented and there is an individual record for each segment. There is another exception that I have to mention: RefSeq does not calculate neighbors for the flu virus. You will have to obtain these sequences separately as described below. Since you are after all of the GenBank genomes, the download from the web may time out for those formats that generate larger files. In such case use Batch Entrez or eutils as described in this article: https://support.ncbi.nlm.nih.gov/link/portal/28045/28049/Article/1928 For the influenza sequences, go to the Influenza virus resource: https://www.ncbi.nlm.nih.gov/genomes/FLU/Database/nph-select.cgi?go=database Click on the FTP link at the top of the page, that will take you to: ftp://ftp.ncbi.nih.gov/genomes/INFLUENZA/
>
> Refer to the README file in the directory.

# How do I set up the BLAST for taxonomy operations?

To enable blast searches that recognize taxonomies you have to place the contents of the `taxdb.tar.gz` file in the location where blast looks for databases.

```
wget ftp://ftp.ncbi.nlm.nih.gov/blast/db/taxdb.tar.gz
```

Then move this file to the blast database directory. Once this is set up the `-outfmt` paramter to `blast` may contain taxonomy related placeholders `%T`, `%t` etc.

```
blastdbcmd -db 16SMicrobial -entry all -outfmt "%a %T %t" | head
```

produces:

```
NR_118889.1 36819 Amycolatopsis azurea strain NRRL 11412 16S ribosomal RNA gene, partial seque
nce
NR_118899.1 1658 Actinomyces bovis strain DSM 43014 16S ribosomal RNA gene, partial sequence
NR_074334.1 224325 Archaeoglobus fulgidus strain DSM 4304 16S ribosomal RNA gene, complete seq
uence
NR_118873.1 224325 Archaeoglobus fulgidus strain VC-16 16S ribosomal RNA gene, complete sequen
ce
NR_119237.1 224325 Archaeoglobus fulgidus strain VC-16 16S ribosomal RNA gene, complete sequen
ce
NR_118890.1 1816 Actinokineospora fastidiosa strain ATCC 31181 16S ribosomal RNA gene, partial
 sequence
NR_044838.1 1381 Atopobium minutum strain NCFB 2751 16S ribosomal RNA gene, complete sequence
NR_118908.1 1068978 Amycolatopsis methanolica strain 239 16S ribosomal RNA gene, partial seque
nce
NR_118900.1 1655 Actinomyces naeslundii strain DSM 43013 16S ribosomal RNA gene, complete sequ
ence
NR_044822.1 2075 Pseudonocardia nitrificans strain IFAM 379 16S ribosomal RNA gene, partial se
quence
```

Note how the results include the taxid `36819` and scientific name `Amycolatopsis azurea`.

# How do I extract sequences by taxonomy from a BLAST database?

Suppose we wanted all sequences deposited under the taxonomy id `1392`

```
taxonkit list --show-name --show-rank --ids 1392
```

that corresponds to `Bacillus anthracis` species:

```
1392 [species] Bacillus anthracis
  191218 [no rank] Bacillus anthracis str. A2012
  198094 [no rank] Bacillus anthracis str. Ames
  205919 [no rank] Bacillus anthracis str. Kruger B
  212045 [no rank] Bacillus anthracis str. Western North America USA6153
  260799 [no rank] Bacillus anthracis str. Sterne
  261591 [no rank] Bacillus anthracis str. Vollum
```

The following steps would achieve that

```
URL=https://ftp.ncbi.nlm.nih.gov/pub/taxonomy/accession2taxid/nucl_gb.accession2taxid.gz
curl $URL | gunzip -c > acc2taxid.txt

# Filter accession numbers by the known taxonomy 9606.
cat acc2taxid.txt | awk ' $3 == 1392 {print $1}'  > acc.txt

# Extract sequences that belong to that taxonomy.
cat acc.txt | blastdbcmd -db nt -entry_batch -  > sequences.fa
```

# What does the env_nt blast database contain?

NCBI states:

> Sequences from environmental samples, such as uncultured bacterial samples isolated from soil or marine samples. The largest single source is Sagarsso Sea project. This does NOT overlap with nucleotide nr

```
mkdir -p ~/blastdb
(cd ~/blastdb && update_blastdb.pl --decompress env_nt)
blastdbcmd -db env_nt -entry all -outfmt "%a %T %S" | head
```

prints:

```
AAFX01139340.1 256318 metagenome
AAFX01139339.1 256318 metagenome
AAFX01139338.1 256318 metagenome
AAFX01139324.1 256318 metagenome
AAFX01139323.1 256318 metagenome
AAFX01139322.1 256318 metagenome
```

most of this data consists of short, random, unlocalized and unidentified sequences obtained during large scale sequencing.

# Classifying 16D sequences

# What data will be analyzed?

For this chapter, we will use data from the Human Microbiome Project (HMP) Demonstration Projects that contain 1065 sequencing runs that attempt to connect the disease of Necrotizing enterocolitis a devastating complication of prematurity to microbial composition of the gut.

- https://www.ncbi.nlm.nih.gov/bioproject/46337

It is important to note that often data with clinical relevance has many limitations imposed due to patient privacy practices. To fully analyze this data in its full context you would need permission to access phenotype (disease) related information on each patient. In the following, we limit ourselves to publicly available information.

Get all the SRR runs for the project (1065 runs in total):

```
# Search the runs for this project.
esearch -query PRJNA46337 -db sra | efetch --format runinfo > runinfo.csv
```

Investigating the file (open in Excel as it is comma separated) we can see that the first run id is `SRR1614899`. Get the data for it:

```
# Obtain the data for the run.
fastq-dump --split-files SRR1614899
```

This produces three files, with no explanation of what these are. A little detective work is necessary. Depending on the instrumentation related metadata makes it into the repository. For example the barcode by which the instrument tagged the samples or the primer that was used to isolate the sequences.

In this case the first file `SRR1614899_1.fastq` contains the sequence `TCAGACGGCTC` and second file `SRR1614899_2.fastq` contains the 16S rRNA primer that was used to select the 16S rRNA region `CCGTCAATTCATTTGAGT`. Checking the 16S rRNA wiki page we can see how this sequence corresponds to primer `907R`.

Finally, the file called `SRR1614899_3.fastq` contains the primary dataset and will be used for data analysis.

# Can I classify 16S sequences via alignment?

The most "logical" and "straightforward" approach to classification - though as we'll see it is neither the most accurate nor efficient is to align the sequences to their closest match. Since we do expect that many of our sequences will only be partially similar but not necessarily identical to known sequences a local alignment tool, like BLAST might be the most appropriate:

```
# Get the 16S database and place it into a newly made folder called blastdb.
(mkdir -p ~/blastdb && cd ~/blastdb && update_blastdb.pl --decompress 16SMicrobial)

# Convert FASTQ file to FASTA
seqtk seq -A SRR1614899_3.fastq >  SRR1614899.fa

# Align converted sequence with blastn
blastn -db ~/blastdb/16SMicrobial -query SRR1614899.fa > SRR1614899.blastn
```

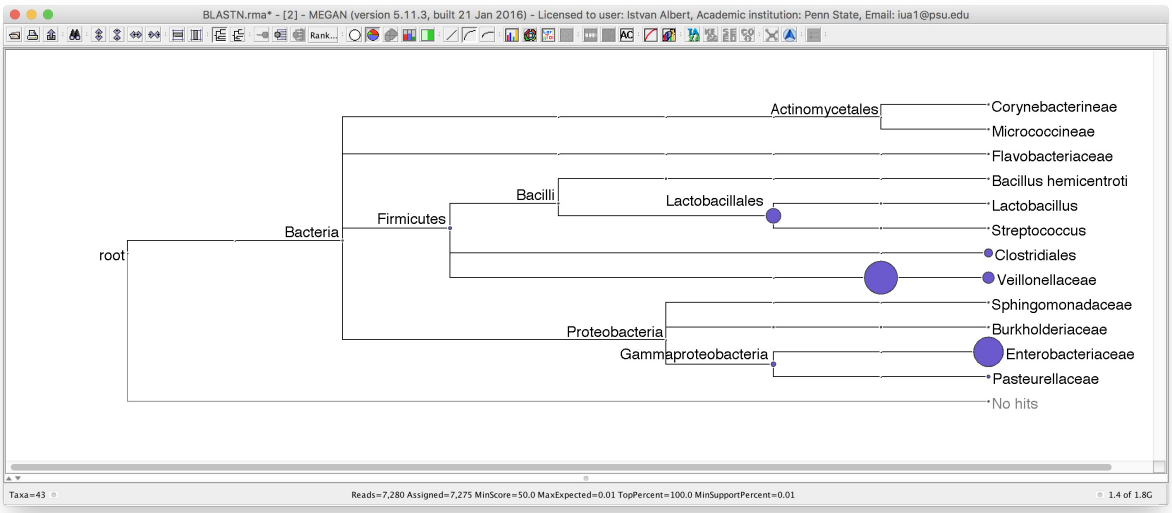
The input file has `7280` sequences. Guess how many lines does the `SRR1614899.blastn` contain? Done? There are probably more than you think, a lot more. Perhaps add a few zeroes to estimate. In the case above the total number of lines in the file was `88,073,057` , and the data was generated in about 30 minutes. Apparently, this process would not scale well as the amount of data rises.

In the resulting alignment file, each sequence may contain hundreds if not thousands of hits to all other similar 16S genes. Of course, since all 16S genes are similar to one another the resulting file is humongous. Obviously, the method is extremely redundant. You'd be surprised however just how many publications use blast to classify sequences.

# How do I visualize the 16S classification?

The Metagenome Analyzer (MEGAN) is a software can be used to load up the data. There are different versions of the tool, and as it happens the latest version, MEGAN 6, appears to have some data loading bugs. Only certain types of data can be visualized with it. In the following images use MEGAN version 5. Explaining what MEGAN does and how it works is beyond the scope of this book. The tool has a detailed manual and several publications that describe the use cases.

What we will do is demonstrate a few of the visualizations. For example, MEGAN can draw a taxonomical tree with the number of reads assigned to a rank:




Our `SRR1614899.blastn` classification will look like the image above. Note how the classification stops at the `Family` rank.

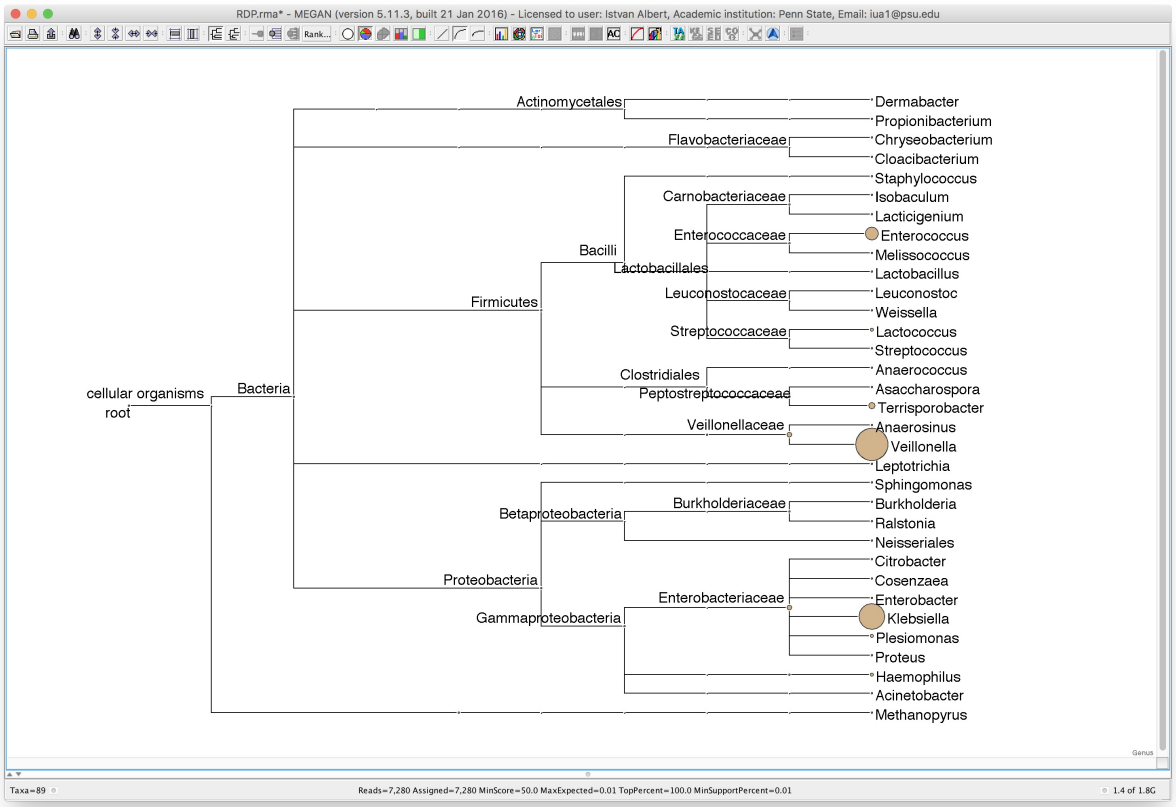
# Are there better ways to classify 16S sequences?

Oh, indeed there are. Blast alignment "kind-of" work but are an incredibly redundant and wasteful choice for this process. For the 16S gene, in particular, far more efficient classifiers have been built that rely on the patterns in the sequences rather than alignments.

We prefer the RPD classifier as a command line choice. It used to be that it was distributed as a simple command line tool that required only Java to be installed. Today is a more complicated as described in RDPTools GitHub page. Once installed you can run the classifier as:

```
java -jar ~/src/RDPTools/classifier.jar classify SRR1614899_3.fastq -o assign.txt -h ranks.txt
```

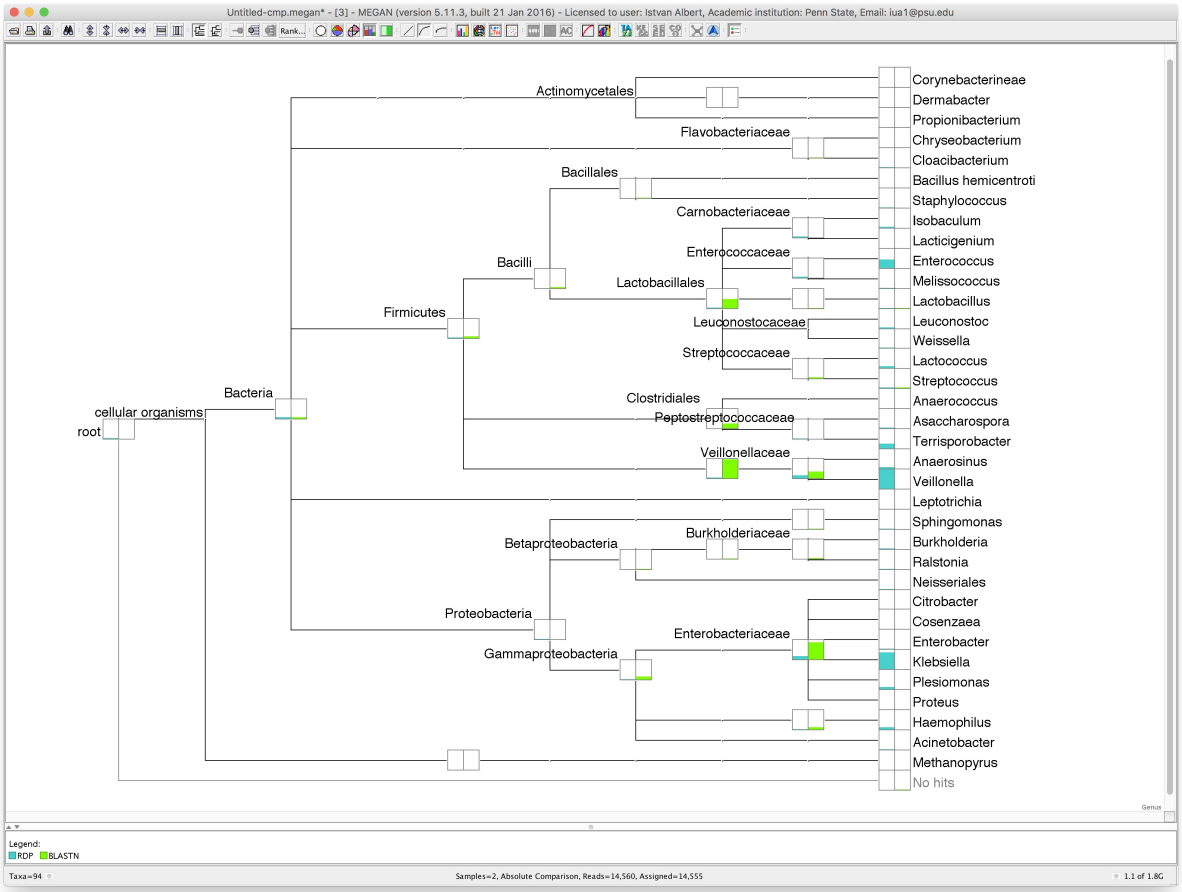
This process classifies the same number of sequences as before but does so in just 50 seconds and produces much smaller files. The `assign.txt` file contains a taxonomical assignment for each input read and can be loaded up into Megan to visualize the results. The `ranks.txt` file contains the number of reads assigned to a taxonomical rank and the number of reads (count) assigned to that rank.




Note how this classification has now assigned reads up to the `genus` level though still not at species level that many might have expected.

# How can I visually compare classifications?

MEGAN allows us to compare classifications. For example, we may compare the BLAST and RDP results to draw the image below. Note just how much more detailed the blue assignments are. RDP (blue) can classify the sequences faster and at a lower rank than BLAST (light green). Unfortunately, the limitations of BLAST for classification are not well understood and large numbers of scientific publications still make use of it.




# Are there other 16S classification methods?

Yes, there is a wide variety of tools and techniques, often integrated into automated pipelines. See the chapter on Metagenomics Analysis

# How do I classify multiple samples?

The RPD classifier can take multiple datasets and can classify them simultaneously. Let's get data for four files and classify all four:

```
# Obtain four runs.
cat runinfo.txt | cut -f 1 -d , | grep SRR | head -4 | xargs -n 1 fastq-dump --split-files

# Classify all four datasets.
java -jar ~/src/RDPTools/classifier.jar classify *_3.fastq -f allrank -o assign.txt -h ranks.t
xt
```

the resulting file contains a tabular ranking that :

```
cat ranks.txt | cut -f 1,3-10 | head
```

of the form:

```
taxid name                rank SRR1614899_3.fastq SRR1614900_3.fastq SRR1614901_3.fastq SRR161
4903_3.fastq
0     Root              rootrank 7280 3590 8891 11472
1     Bacteria            domain 7256 3572 8850 11393
841   "Proteobacteria"    phylum 2674 3055 933 8000
1501  Gammaproteobacteria  class 2653 3039 933 7993
1616  "Enterobacteriales"  order 2589 3030 933 7975
1617  Enterobacteriaceae  family 2589 3030 933 7975
1649  Plesiomonas          genus    2    2   0    3
1637  Klebsiella           genus 2100 2450 598 6413
1631  Enterobacter         genus 2 1 0 1
```

You can subselect the classifiction by different taxonomical levels:

```
cat ranks.txt | cut -f 1,3-10 | awk ' $3=="class" { print $0}' | sort -rn -k 4
```

that produces:

```
2722 Negativicutes       class 3497    3   41    2
1501 Gammaproteobacteria class 2653 3039  933 7993
2260 Bacilli             class  803  428 5831 3079
2375 Clostridia          class  220    6  492   19
1175 Betaproteobacteria  class   14    6    0    0
842  Alphaproteobacteria class    5    8    0    0
458  Flavobacteriia      class    3   13    0    0
3    Actinobacteria      class    2   18  160  143
790  Fusobacteriia       class    1    0    0    0
423  "Bacteroidia"       class    0    0 1135    0
```

A lot can be learned from a classification as simple as this. While we are at it note how widely different the counts are across samples.

# How do I determine statistically relevant differences between the classification counts?

There is a debate on whether the count data obtained via classification is conceptually similar to the counts obtained when assigning reads to transcripts. If the statistical assumptions used to model data originating from RNA-Seq experiments still apply to metagenomics data, then we could use the wide variety of methods designed for RNA-Seq to analyze classification counts as well.

One such assumption, for example, is that the total number of non-changing elements is substantially larger than the number of differentially expressed entries (basically that only a small subset changes). If this requirement is not met, then methods originally developed for RNA-Seq should not be used unless

there is a reason to believe that this requirement may be ignored.

Much has been written about the subject of comparison and normalization; a recent overview can be read in the paper Normalization and microbial differential abundance strategies depend upon data characteristics the authors summarize their findings in what we interpreted as: "Well.. it depends on a lot of factors. Good luck my friend!"

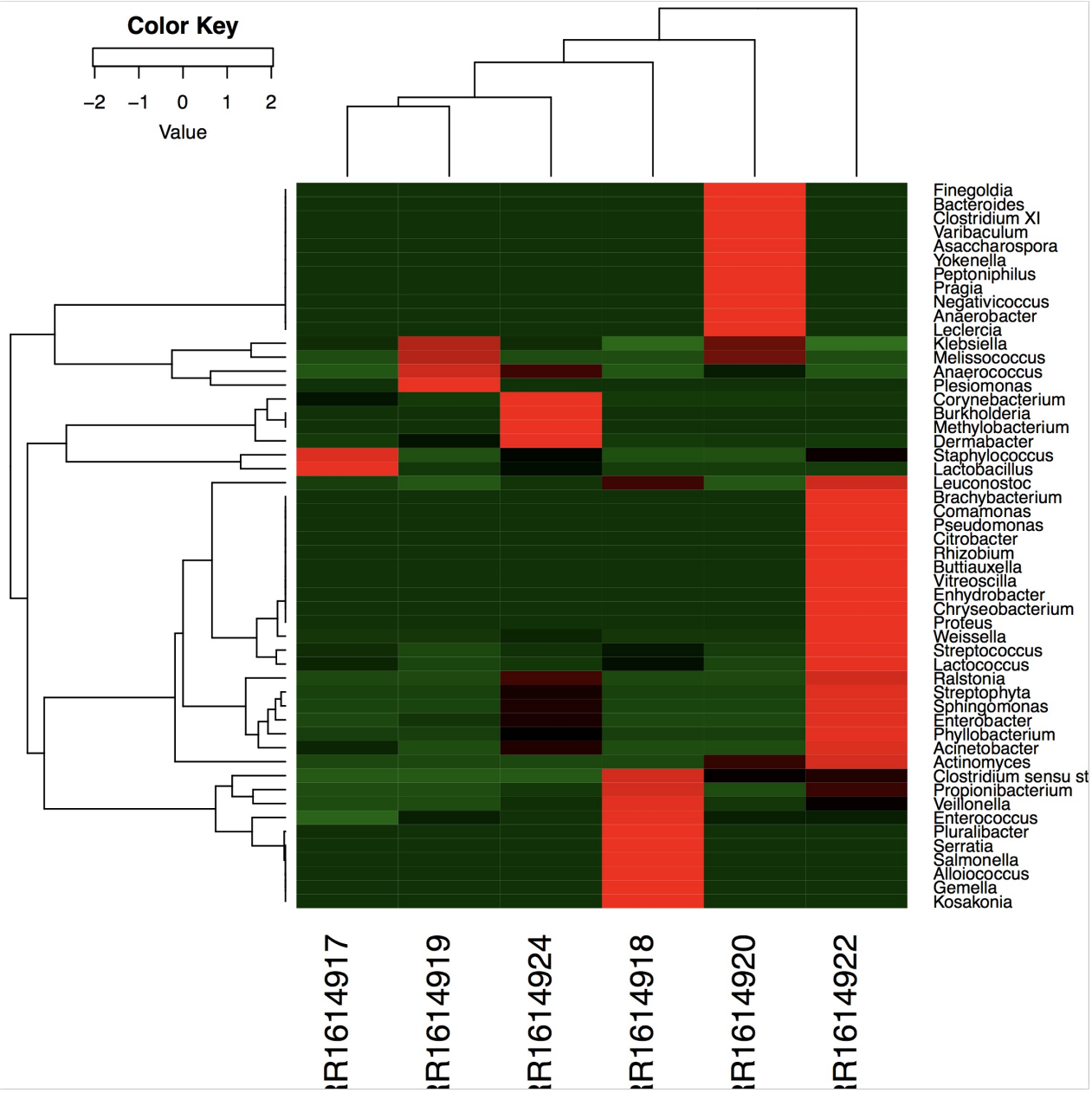
Let's give it a shot here. Suppose we have two groups. The first three runs belong to one group, the other to another group. Put these runs into a file called `groups.txt`.

```
SRR1614918
SRR1614919
SRR1614920
SRR1614917
SRR1614922
SRR1614924
```

obtain the data and classify it:

```
mkdir -p sra
cat groups.txt |  xargs -n 1 fastq-dump --outdir sra --split-files --skip-technical -F

# Ensure to list the SRR numbers by groups. First three, second three.
java -jar ~/src/RDPTools/classifier.jar classify SRR1614918_3.fastq SRR1614919_3.fastq SRR1614
920_3.fastq SRR1614917_3.fastq SRR1614922_3.fastq SRR1614924_3.fastq -f allrank -o assign.txt
-h ranks.txt

# Filter the data for only the columns of interest.
# Add a header as well.
cat ranks.txt | head -1 | cut -f 3,5-11 > counts.txt
cat ranks.txt | awk ' $4 == "genus" { print $0 }' | cut -f 3,5-11 >> counts.txt

# Obtain the deseq2 script from the Biostar Handbook.
curl -O http://data.biostarhandbook.com/rnaseq/code/deseq2.r > deseq2.r

# Run the deseg2
cat counts.txt | Rscript deseq2.r 3x3 > results_deseq2.txt

# Draw the heatmap
cat norm-matrix-deseq2.txt | Rscript draw-heatmap.r > output.pdf
```

That will, in turn, generate a heat map at genus level for the different patients and the genus level abundance of the various bacteria.




One thing is clear - there is a lot of variation across the different groups. Every individual in every group shows radical changes in their metagenomic composition. On the other hand, we don't know if the groups were built correctly - the grouping data was not available due to privacy reasons.

# Full analysis of PRJNA46337

Th following study contains analyis of full datasets. You may complete the analysis even if your computational resources by limiting the data to a smaller subset.

## How to classify all the data for the project PRJNA46337?

Suppose you'd want to classify all the data in project PRJNA46337? How much of computational resourse would it take? The answer depends on a number of factors data download speed as well as number of computer cores.

The project contains two kinds of data:

1. 1407 samples of 16S sequencing data producing on average 10000 reads per run
2. 6 runs of whole genonome sequencing (WGS) on a Illumina Hiseq sequencer producing 70 million reads per sample.

Here we demonstrate the analysis of the 1407 samples

```
# Get the runinfo. Took 40 seconds.
esearch -query PRJNA46337 -db sra | efetch --format runinfo > runinfo.csv

#  Keep the runs performed on a 454 sequencer.
cat runinfo.csv | grep "454 GS FLX Titanium" | cut -f 1 -d , > srr-16s.txt

# Store the 16S data here.
mkdir -p data

# Download the SRR files 10 at a time. On our system this took about 60 minutes.
cat srr-16s.txt | parallel -j 10 --verbose --progress --eta fastq-dump --outdir data --split-f
iles --skip-technical -F
```

Annoyingly some SRR data was split into 3 pieces other into 4, hence the files that contain our biological data are sometimes called `..._3.fastq` versus `..._4.fastq`. These type of inconsistencies are very annoying and slow down the automated processing of data. We do also note that some runs failed to download for mysterious reasons. We have only `1338` datasets whereas the file lists `1397` oh well - we just carry on.

We'd like to rename each file in such a way to split off the `_3` and `_4` and keep just the SRA number itself turning `SRR1038144_3.fastq` into `SRR1038144.fastq` and `SRR052332_4.fastq` into `SRR052332.fastq` respectively. But guess what, it is time to get in touch with that data janitor slice of your soul because there is another complication that you probably miss the first time around and create the wrong solution first (like we did). The number after the `SRR` varies too, it may be 6 or 7 digits long so it renaming won't be as simple as slicing of the string at a position (an easy operation in bash). Instead we have to find and remove either `_3.fastq` or `_4.fastq` from the end of a string. Thankfully there is an operator to do that, and you find it by googling it, the `%` operator.

The sloppines of the data modeling at SRA causes all kinds of unexpected complications right from the beginning.

So we need a renaming script. Call that `rename.sh` and have it contain:

```
for file in *.fastq; do
    name=$file

    # Chop off certin types of ends if they exists.
    name=${name%_3.fastq}
    name=${name%_4.fastq}

    # Perform the renaming.
    echo mv $file "${name}.fq"
done
```

Running this script in the data folder:

```
bash rename.sh
```

will show you the operations that it wants to perform. To have the actions perform instead of showing them to you pipe the output again into bash:

```
bash rename.sh | bash
```

We now have nicely named file. Let's classify them all:

```
# This directory will store the results of classification.
mkdir -p results

# Classify in parallel 10 at a time.
cat srr-16s.txt | parallel -j 10 --verbose --progress --eta java -jar ~/src/RDPTools/classifie
r.jar classify data/{}.fq -f allrank -o results/{}.assign.txt -h results/{}.ranks.txt

# Classify all 1381 columns.
time java -jar ~/src/RDPTools/classifier.jar classify data/*.fq -f allrank -o results/assign.t
xt -h results/ranks.txt
```

On our system this took about 7 hours and you end up with a file with 0ver 1000 columns where each sample is fully classified.

The datamash utility can help us probe the data to compute various values:

```
cat counts.txt | datamash -H sum 2-8
```

produces:

```
sum(SRR052150.fq)    sum(SRR052151.fq)    sum(SRR052152.fq)    sum(SRR052153.fq)    sum(SRR052
154.fq)    sum(SRR052155.fq)    sum(SRR052156.fq)
2509    3346    3379    3302    2988    2905    3350
```

The above shows the number of classified sequences per sample. Pretty low actually. Let us count how many bacteria of the Streptoccocus genus each column has

```
cat counts.txt | grep Streptococcus | cut -f 1-20 | head
```

it produces:

```
Streptococcus    191    290    27    2536    1    0    0    309    1    0    10    370    86
  1    392    166    37    0
```

It is pretty clear that the variability accross samples is gigantic. Some samples indicate no such bacteria is present ( `SRR052155.fq` ) whereas others (example `SRR052153.fq` ) show that over 75% of the bacteria (2536/3302) of the sequences assigned to this genus. This genus include over 50 species of bacteria and while many have quite the bad rep for causing pink-eye, meningitis, pneumonia many more are not pathogenic at all and are often present in healthy guts as well. In a manner of speaking this makes all classifictiona at higher levels of taxonomy less informative than initially assumed.

Now due to privacy considerations scientists were not allowed to distribute phenotype data of clinical relevance. For example you can't immediately access information that would connect samples to a disease phenotype. You would need to apply for permission to obtain that. The typical process here would create subgroups of our columns then attempt to discover trends in data for these.

# What is the most common misunderstanding (paradox) of metagenomics?

Look at the results above. Note how `SRR052155.fq` has no reads assigned to the `Streptococcus` genus whereas `SRR052153.fq` has 75% of its reads assigned to it. As yourself the following

> Would you conclude that the first sample contains fewer bacteria at the `Streptococcus` genus than the second?

As surprising as it sounds the answer is: NO. The only thing you may conclude is that, when compared to all the other existing bacteria, relatively speaking there are fewer `Streptococcus` bacteria in the first sample than the other.

Here is a better explanation. Imagine that both samples have the same number of `Streptococcus` bacteria in them makes that 10,000. But the first sample also has many millions of other bacteria and our random sampling of 2000 never had the chance to pick them up, they always hit something else. In the second sample there were very few other bacteria, our sampling mostly hit the `Streptococcus` variety. Hence turned out at 75%.

See how the counts reflect the relative abundances and not absolute ones. And of course that also provides us with very valuable information. We only wish to caution against interpreting the classification rates incorrectly.

# How reliable are these results?

You may wonder, just how reliable is the observation that 75% of the bacteria in a certain gut sample belongs to the Streptococcus genus? The short answer is that probably not that reliable.

The entire field of metagenomics is at its infancy. It is not entirely clear how to sample the same gut correctly after all there are many imhomogenities in different areas of the gut. There is variability and errors in sample extraction, handling and of the lenghty process of analysis.

A reliable method would be one that replicated each measurement multiple times and had the means to validate the accuracy and precision of each obtained value. The field is extremely far from doing that.

Instead of proper experimental design and validation scientists perfer to rely on "fancy" statistical methods that few (usually not even the people that use them) understand. There is a long way to go yet in obtaining reliabke results.

Metagenomics must still be considered as an early, exploratory science where we still bumbling around.

# Classifying whole genome data

Shotgun metagenomic sequencing allows scientists to study the full DNA of organisms present in a given "biological sample." Whereas 16S sequencing allows you to group bacteria into OTUs and different taxonomical ranks, whole shotgun sequencing allows for a more detailed study of organisms from classifying into subspecies to strains to investigating functional roles as well.

Of course, there is the added complexity of having to deal with massively increased amounts of data that often are of many orders of magnitude larger than a 16S type study.

# How do I evaluate how well a given method works?

Since most bacteria are currently unknown, it is somewhat difficult to assess methods as we have to evaluate success relative to a small known subset of reality. Also, as we found out, there is a bit of wild west mentality in the field, as you will see it is surprisingly difficult to evaluate methods as we continuously struggle with data janitorial tasks: the naming of bacteria can vary wildly, taxonomies not used consistently, etc.

Several publications have attempted to objectively quantify the quality of metagenomic techniques, among them the paper titled Assessment of Metagenomic Assembly Using Simulated Next Generation Sequencing Data, PLoS One, 2012. We will use a simulated data set from this paper to get a practical feel for the strengths and weaknesses of the various methods.

The data is distributed from:

- http://www.bork.embl.de/~mende/simulated_data
- http://www.bork.embl.de/~mende/simulated_data/README.txt

- Supporting Table 2: https://doi.org/10.1371/journal.pone.0031386.s002

# How do I obtain the data?

If your computer can handle the computational load you could download the original data from the site above with

```
curl http://www.bork.embl.de/~mende/simulated_data/illumina_100species.1.fq.gz | gunzip -c > r
ead1.fq
curl http://www.bork.embl.de/~mende/simulated_data/illumina_100species.2.fq.gz | gunzip -c > r
ead2.fq
```

And run your analyses on `read1.fq` and `read2.fq` respectively. That being said these datasets are a wee bit too large for exploratory analyses. We have down-sampled the first-pair file into 1 million random reads with:

```
seqtk sample -2 -s 2 read1.fq 1000000 > subset.fq
```

and made it available from the Biostar Handbook data site. You can download this data with:

```
curl http://data.biostarhandbook.com/meta/data/subset.fq.gz | gunzip -c > subset.fq

# See what the file contains.
seqkit stat subset
#file       format  type    num_seqs     sum_len  min_len  avg_len  max_len
#subset.fq  FASTQ   DNA   1,000,000  75,000,000       75       75       75
```

This subsampled data can help us in evaluating classification methods but would not be sufficient to say assemble new genomes.

# What are the expected abundances of the data?

The paper comes with some supplementary files among them:

```
# This will download the supplemental info in XLS format.
curl -OJL https://doi.org/10.1371/journal.pone.0031386.s002
```

In this case before we could evaluate a classification method we ought to understand what we expect to see - what kind of properties do the artificially generated data have? Look at the Excel file that we downloaded above.

| A | B | C | D | E | F | G | H | I | J | K | L | M | N |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Supplemental Table 2 - Genomes Used in the Medium Complexity Metagenome and Estimated Coverage (100 genomes)** | | | | | | | | | | | | | |
| **Rank** | **Chomosome** | **Chromosome** | **Name** | **phylogenetic group** | **genome size (Mb)** [1] | **GC %** | **abundance** [2] | **Percent Relative Abundance** | **Est_proportion of total sequence** | | **Estimated coverage for 454** | **Estimated coverage for Sanger** | **Estimated coverage for Illumina** |
| 1 | NC_007969 | 0 | Psychrobacter cryohalolentis K5 | Gammaproteobacteria | 3.1 | 42.2 | 1000 | 2.227683886 | 1.862781273 | | 1.50224296 | 1.20179 | 17.84665 |
| 2 | NC_009077 | 0 | Mycobacterium sp. JLS | Actinobacteria | 6 | 68.4 | 794.3108671 | 1.769473519 | 2.863794984 | | 1.19324791 | 0.9546 | 14.17579 |
| 3 | NC_007604 | 0 | Synechococcus elongatus PCC 7942 | Cyanobacteria | 2.75 | 55.4 | 707.1067812 | 1.575210382 | 1.168470804 | | 1.06224619 | 0.8498 | 12.61948 |
| 4 | NC_004722 | 0 | Bacillus cereus ATCC 14579 | Firmicutes | 5.42 | 35.3 | 656.2595204 | 1.461938759 | 2.137347181 | | 0.98586125 | 0.78869 | 11.71203 |
| 5 | NC_002662 | 0 | Lactococcus lactis subsp. lactis Il1403 | Firmicutes | 2.4 | 35.3 | 621.9749249 | 1.385563518 | 0.896983155 | | 0.93435745 | 0.74749 | 11.10017 |
| 6 | NC_004344 | 0 | Wigglesworthia glossinidia endosymbiont of Glossina brevipalp | Gammaproteobacteria | 0.7 | 22.5 | 596.8309535 | 1.329550698 | 0.251043828 | | 0.8965851 | 0.71727 | 10.65143 |
| 7 | AC_000091 | 0 | Escherichia coli str. K12 substr. W3110 | Gammaproteobacteria | 4.65 | 50.8 | 577.3502692 | 1.286153891 | 1.613215904 | | 0.86732038 | 0.69386 | 10.30377 |

Take for example the first entry (table sorted by coverage) states that `Psychrobacter cryohalolentis K5` corresponds to accession numbers `NC_007969` and will be present at an expected coverage of `17.84`. Let's see how the well the data supports this information.

```
# This took
cat subset.fq | egrep '^@NC_007969|^@NC_007968' | wc -l
# 15902

# Get the size of NC_007969 + NC_007968
efetch -db=nuccore -id=NC_007969,NC_007968 -format=fasta | seqkit stat
# file  format  type  num_seqs    sum_len  min_len      avg_len    max_len
# -     FASTA   DNA          2  3,101,097   41,221  1,550,548.5  3,059,876
```

Above we see that a total of `15902` reads were found as being generated from `Psychrobacter cryohalolentis K5`. The original data contained about 53 million reads and we used a subset of 1 million of it, we can extrapolate (estimate) the total number of reads in the original data as being about `15902 * 53 ~ 842,000`. Since we did download the full data it is worth checking out how well the estimated number matches reality:

```
cat read1.fq read2.fq | egrep '^@NC_007969|^@NC_007968' | wc -l
# 849442
```

The result is very close indeed. The accuracy of our estimate gives us confidence that we'll be able to estimate other quantities of interest just as well even from a sample of 1 million reads.

# Are there different kinds of abundances?

The word abundance can carry different meanings. Whereas during 16S sequencing each read corresponds to a bacterial cell, in the case of whole genome sequencing the length of the genome also matters. Longer bacteria will produce more DNA and will be seen in more sequencing reads. Another way to say this is that ten read counts of one bacteria vs. one read count of another bacteria does not mean that there were more of the first than the second.

# What percent of a measurements originate from a given species?

This measure depends on the length of the DNA of the organism multiplied by the number of copies of that organism. It can be found by dividing the number of reads that are classified at that rank with the total number of reads in the sample:

```
15902/1000000
```

This value does not vary when we subsample the data. In both cases `1.5%` of the reads in our file were quantified as belonging to `Psychrobacter cryohalolentis K5`. This number does not appear in the supporting materials of this research paper. Some classifiers will only report this value.

Again it is important to remember that these numbers do not indicate the counts of the bacteria in the sample. They indicate what percent of measurements were "spent" identifying one particular member of the community.

# What is the coverage?

As you recall the coverage represents the number of times each base of genome is expected to be sequenced. The genome size for `NC_007969` can be found with:

```
efetch -db=nuccore -id=NC_007969,NC_007968 -format=fasta | seqkit stat
#file  format  type  num_seqs    sum_len  min_len      avg_len    max_len
#-     FASTA   DNA       2  3,101,097   41,221  1,550,548.5  3,059,876
```

Since each read is `75` bp long and you have sequenced 849442 reads the coverage of of this bacteria will be `849442 * 75 / 3101097 = 20.5x` . Notably, this does not match the coverage in the Excel sheet that is reported to be `17.4x` . The difference is not substantial, but it is a bit discomforting that we are unable to reproduce it from the data. It is not clear why this discrepancy exists - unfortunately it is not uncommon to run into such issues.

# How can I tell how many bacteria are in the sample?

First, you need to recognize that sequencing does not produce absolute numbers. From sequencing data alone we cannot tell if a bacteria was present in one, ten, one million or one billion copies. What we can identify are the so called "relative abundances" how many more of one bacteria are there compared to another.

When a coverage tells you that an organism was covered at 10x versus another one covered at 1x it directly means that the first must have had ten times as many more copies present as the latter. To figure out the relative abundance percentages sum up all coverages then express each coverage as a percentage of the total.

For example, if you had bacteria A, B, and C covered at 10x, 20x and 30x coverages (10+20+30=60) then their relative abundances will be `10/60=17` , `20/60=33` and `30/60=50` . The fractions help to express the composition as percentages. Your sample is then composed of 17% bacteria A, 33% bacteria B and 50% bacteria C. These numbers represent abundances based on the number of cells.

# How do I match the results of a classifier?

As we prepared this chapter, we realized that it is surprisingly difficult to verify and validate the results produces by classifiers. Several inconsistencies may exist in both the expected data and the results generated by our software. Sometimes the taxonomy id is missing, in other cases, the names may be spelled slightly differently, for each organism, or perhaps the accession numbers are not annotated with a taxonomy etc.. This makes it surprisingly difficult to match results to expectations.

# How does Kraken classifier work?

The classifier has quite high performance and can be run with:

```
kraken -db ~/refs/kraken/minikraken_20141208 -fastq-input subset.fq > results.out
```

that produces the output:

```
26667004 sequences (2000.03 Mbp) processed in 401.844s (3981.7 Kseq/m, 298.63 Mbp/m).
  16202163 sequences classified (60.76%)
  10464841 sequences unclassified (39.24%)
```

The above was generated with the "prebuilt" mini-kraken database distributed by the authors. The `results.out` file tracks each input read and lists the classification for each measurement. The information typically needs to be consolidated into a more informative report with:

```
 kraken-report -db ~/refs/kraken/minikraken_20141208 results.out > results.txt
```

Where the `results.txt` generated above is a file that contains the sample abundances at each taxonomical rank (check the Kraken Manual for details on what each column means):

```
 39.38  393820  393820  U       0       unclassified
 60.62  606180  3126    -       1       root
 60.31  603053  9       -       131567  cellular organisms
 59.29  592937  930     D       2       Bacteria
 25.12  251162  491     P       1224    Proteobacteria
 13.69  136877  323     C       1236    Gammaproteobacteria
  6.69  66892   0       O       91347   Enterobacteriales
  6.69  66892   24149   F       543     Enterobacteriaceae
  1.00  10042   426     G       561     Escherichia
  0.96  9611    8721    S       562     Escherichia coli
  0.06  602     602     -       331111  Escherichia coli E24377A
  0.02  243     243     -       405955  Escherichia coli APEC O1
  0.00  21      19      -       83333   Escherichia coli K-12
  0.00  2       2       -       316407  Escherichia coli str. K-12 substr.
  ...
```

Each row indicates the taxonomical rank (for example `S` means Species). To select the rows at species rank with at least 100 reads supporting that rank

```
cat results.txt | awk ' $2>100 && $4=="S" { print $0}' | wc -l
# 83
```

Hence the classifier predicts the presence of `83` species. It remains to be seen if these are indeed matching the 100 species that we know were present. It is clear though that we have missed out on some 20% of the species. To compare to the expected data reformat the `results.txt` file to select as species level and keep the taxid and name of the bacteria (we do also sort it so that we can later join this file to another):

```
# Put the results with their taxid into a file
cat results.txt | awk ' $2>100 && $4=="S" { OFS="\t"; print $5 }' | sort  > kraken.taxid
```

This file called `kraken.taxid` now contains just the taxonomic ids:

```
101534
106590
1076
1100
...
```

We will now attempt match this to the expected data. Because the tax ids were not specified for the data in the paper it is surprisingly challenging to compare the expected results to the obtained ones.

# How do I match back the bacteria?

As you'll experience yourself often the greatest of challenges occur when trying to do something really simple, that has seemingly nothing to do with bioinformatincs all. In this case we want to figure out how well the classification worked, we want to match the file `kraken.txt` to the contents of the XLS sheet distributed as Supplementary information. It is a surprisingly tough job to get right.

The taxid is present in one file but os not there in the second, hence we first need to generate a taxid for each entry of the XLS sheet. Lets give it a go. First create a `tmp` file by pasting the bacteria names from the D column of the Excel sheet into it. Then cut the first two words and place them into the file called `names.txt`:

```
cat tmp | cut -f 1,2 -d ' '
```

So that stream will give you this:

```
Psychrobacter cryohalolentis
Mycobacterium sp.
Synechococcus elongatus
Bacillus cereus
...
```

Now pipe this output into the taxonkit:

```
cat tmp | cut -f 1,2 -d ' ' | taxonkit name2taxid
```

that will give you:

```
Psychrobacter cryohalolentis  330922
Mycobacterium sp.             1785
Synechococcus elongatus       32046
Bacillus cereus               1396
```

We notice that some bacterial names did not produce a taxid and there are duplicates there as well. Two species may correspond to same taxid.

```
cat tmp | cut -f 1,2 -d ' ' | taxonkit name2taxid | cut -f 2 | sort | uniq | egrep "[0-9]+" >
expected.taxid
```

When all said and done the `expected.taxids` file contains `76` distinct taxonomic ids. That is quite the drop from the original 100.

The `comm` tool allows us to select lines that are common (or different in files).

```
# Taxids in both file.
comm -1 -2 expected.taxid kraken.taxid  | wc -l
# 70

# Taxids missed in classification.
comm -2 -3 expected.taxid kraken.taxid
# 6

# Taxids present in the kraken classification only.
comm -1 -3 expected.taxid kraken.taxid  | wc -l# 78
#13
```

So what was missed

```
comm -2 -3 expected.taxid kraken.taxid | taxonkit lineage
```

Expected species not found:

```
1765    cellular organisms;Bacteria;Terrabacteria group;Actinobacteria;Actinobacteria;Coryneba
cteriales;Mycobacteriaceae;Mycobacterium;Mycobacterium tuberculosis complex;Mycobacterium bovi
s
1773    cellular organisms;Bacteria;Terrabacteria group;Actinobacteria;Actinobacteria;Coryneba
cteriales;Mycobacteriaceae;Mycobacterium;Mycobacterium tuberculosis complex;Mycobacterium tube
rculosis
1785    cellular organisms;Bacteria;Terrabacteria group;Actinobacteria;Actinobacteria;Coryneba
cteriales;Mycobacteriaceae;Mycobacterium;Mycobacterium sp.
1831    cellular organisms;Bacteria;Terrabacteria group;Actinobacteria;Actinobacteria;Coryneba
cteriales;Nocardiaceae;Rhodococcus;Rhodococcus sp.
203804    cellular organisms;Bacteria;Proteobacteria;Gammaproteobacteria;Enterobacterales;Ente
robacteriaceae;unclassified Enterobacteriaceae;ant, tsetse, mealybug, aphid, etc. endosymbionts
;ant endosymbionts;Candidatus Blochmannia
50422    cellular organisms;Bacteria;Proteobacteria;Gammaproteobacteria;Alteromonadales;Shewan
ellaceae;Shewanella;Shewanella sp
```

# Would the full kraken database be more accurate?

If we were to build the most up-to-date full database as instructed in the Kraken manual (though the process is a bit tedious) the rate of classification ratio rises substantially:

```
kraken --db /export/refs/kraken_db/kraken_std_db/ --fastq-input subset.fq > results-full.out
```

that in turn produces:

```
1000000 sequences (75.00 Mbp) processed in 25.376s (2364.5 Kseq/m, 177.33 Mbp/m).
  826977 sequences classified (82.70%)
  173023 sequences unclassified (17.30%)
```

So even in the "best-case scenario" of a controlled experiment with 100 known species 17% of our sequences will be unclassified even though they match known genomes.

# How to use sourmash?

According to their documentation:

> sourmash is a command-line tool and Python library for computing MinHash sketches from DNA sequences, comparing them to each other, and plotting the results. This allows you to estimate sequence similarity between even very large data sets quickly and accurately.

```
pip install https://github.com/dib-lab/sourmash/archive/2017-ucsc-metagenome.zip
pip install khmer
```

Download and install the sourmash signature database:

```
mkdir -p ~/refs/sourmash
URL=https://s3-us-west-1.amazonaws.com/spacegraphcats.ucdavis.edu/microbe-refseq-sbt-k21-2017.
05.09.tar.gz
(cd ~/refs/sourmash && curl -k $URL | tar xzv)
```

Compute a signature on your subset of data:

```
sourmash compute --scaled 10000 -k 21 subset.fq
```

Characterize the signature:

```
sourmash sbt_gather -k 21 ~/refs/sourmash/refseq-k21.sbt.json subset.fq.sig -o sourmash.csv
```

The tool will produce somewhat differnt information on the screen versus the results file `sourmash.csv`

```
overlap    p_query p_genome
-------    ------- --------
1.4 Mbp    2.7%     24.5%      NZ_CP009464.1 Bacillus anthracis strain
1.1 Mbp    2.2%     40.1%      NZ_JOPX01000001.1 Staphylococcus aureus
1.1 Mbp    2.1%     26.0%      NZ_CP009685.1 Escherichia coli str. K-12
1.0 Mbp    1.9%     15.9%      NC_009077.1 Mycobacterium sp. JLS, compl
0.8 Mbp    1.6%      9.0%      NC_010628.1 Nostoc punctiforme PCC 73102
0.8 Mbp    1.5%      8.0%      NC_008268.1 Rhodococcus jostii RHA1, com
...
```

whereas the `sourmash.csv` file contains:

```
intersect_bp,f_orig_query,f_found_genome,name
1410000.0,0.24479166666666666,0.02744258466329311,"NZ_CP009464.1 Bacillus anthracis strain SK-
102, complete genome"
1140000.0,0.40070921985815605,0.022187621642662515,"NZ_JOPX01000001.1 Staphylococcus aureus Ly
sK 2 2011 psaLysK211.contig.0, whole genome shotgun sequence"
1100000.0,0.26004728132387706,0.02140910860256909,"NZ_CP009685.1 Escherichia coli str. K-12 su
bstr. MG1655, complete genome"
970000.0,0.15901639344262294,0.018878941222265473,"NC_009077.1 Mycobacterium sp. JLS, complete
 genome"
800000.0,0.08958566629339305,0.015570260801868432,"NC_010628.1 Nostoc punctiforme PCC 73102, c
omplete genome"
...
```

How well did the method work? Firs `sourmash` runs substantially slower than kraken, at least 10x slower. Then the accession numbers are not useful for matching purposes since the same sequence may be present under different accession numbers. Let us test how well the results match. Note the command line contraptions that we had to come up with:

```
cat sourmash.csv | cut -d ','  -f 4 | cut -d ' ' -f 2,3 | taxonkit name2taxid | cut -f 2 | sort
 | uniq | egrep "[0-9]+" > sourmash.taxid
```

Now let's compare the outputs:

```
# Taxids in both file.
comm -1 -2 expected.taxid sourmash.taxid  | wc -l
# 69

# Taxids missed in the sourmash classification.
comm -2 -3 expected.taxid sourmash.taxid  | wc -l
# 7

# Taxids present in the sourmash classification only.
comm -1 -3 expected.taxid sourmash.taxid
#6
```

what has been missed by sourmash:

```
comm -2 -3 expected.taxid sourmash.taxid | taxonkit lineage


1100    cellular organisms;Bacteria;FCB group;Bacteroidetes/Chlorobi group;Chlorobi;Chlorobia;
Chlorobiales;Chlorobiaceae;Chlorobium/Pelodictyon group;Pelodictyon;Pelodictyon luteolum
1172    cellular organisms;Bacteria;Terrabacteria group;Cyanobacteria/Melainabacteria group;Cy
anobacteria;Nostocales;Nostocaceae;Anabaena;Anabaena variabilis
1765    cellular organisms;Bacteria;Terrabacteria group;Actinobacteria;Actinobacteria;Coryneba
cteriales;Mycobacteriaceae;Mycobacterium;Mycobacterium tuberculosis complex;Mycobacterium bovi
s
1831    cellular organisms;Bacteria;Terrabacteria group;Actinobacteria;Actinobacteria;Coryneba
cteriales;Nocardiaceae;Rhodococcus;Rhodococcus sp.
203804    cellular organisms;Bacteria;Proteobacteria;Gammaproteobacteria;Enterobacterales;Ente
robacteriaceae;unclassified Enterobacteriaceae;ant, tsetse, mealybug, aphid, etc. endosymbionts
;ant endosymbionts;Candidatus Blochmannia
65    cellular organisms;Bacteria;Terrabacteria group;Chloroflexi;Chloroflexia;Herpetosiphonal
es;Herpetosiphonaceae;Herpetosiphon;Herpetosiphon aurantiacus
9    cellular organisms;Bacteria;Proteobacteria;Gammaproteobacteria;Enterobacterales;Erwiniace
ae;Buchnera;Buchnera aphidicola
```

Quite a few are the same as before.

# What is binning?

Binning is the process of grouping reads or contigs and assigning them to "bins" that will represent taxonomical ranks or taxonomical units. Typically, binning methods are based on sequence compositional features.

The incomplete nature of the sequencing copuled to amiguity introduced by the method itself (breaking long pieces of DNA into smaller fragments) makes it hard to assemble individual genes, much less recovering the full genomes of each organism. Binning techniques represent a "best effort" to identify reads or contigs with certain groups of organism.

Typically reads are first assembled into longer contigs, then these contigs are binned into categories.

# How do I assemble metagenomes?

This topic is so early in the development of the field that we can't, in good faith cover it here. There are many tools that can be run just as other assembler are run, but assessing their results and understanding tuning their quality, interpreting the assemblies are still in our opining no more than just guesstimates.

Most practical advice follows the rule: "Find an assembler that others say is good, keep running it with different parameters until the results seem to make sense". It is weak sauce - but the pretty much the only adsvice we can share as well.

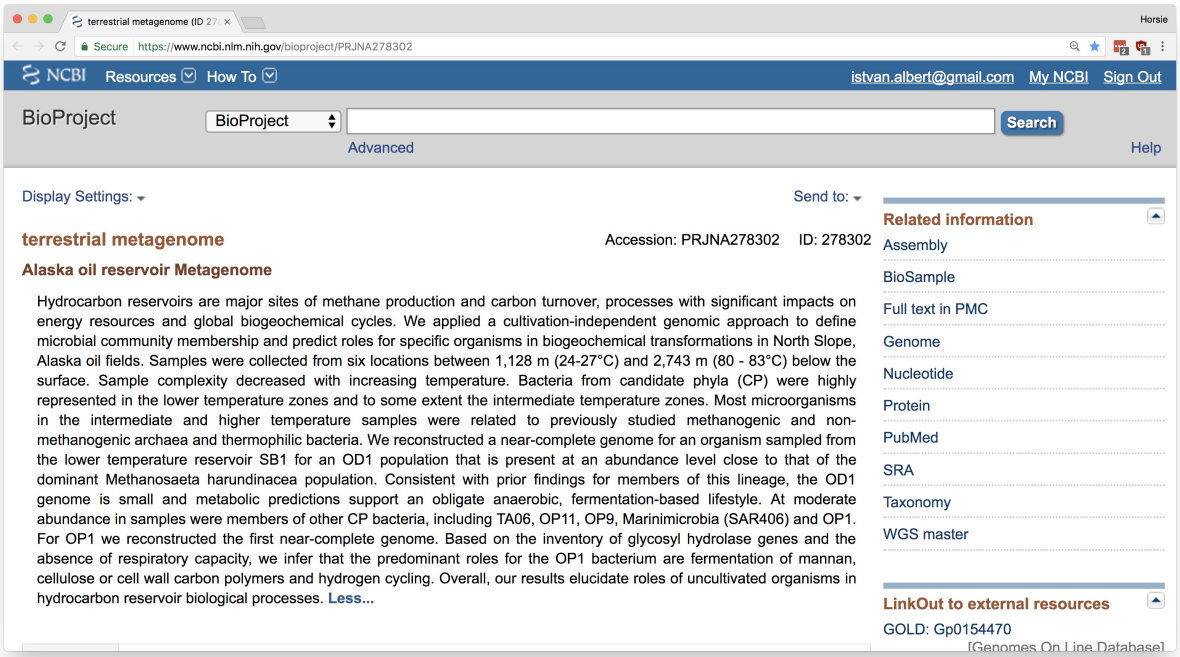
# Alaska Oil Reservoir Metagenome study

Note: this chapter is unfinished.

The reason we kept it is to show you what happens when you go to remote part of Alaska and sequence samples from the ice. Unsurprisingly only 10% of your samples can be assigned to known species. But then you have to write a paper from that. Quite the conundrum of course.

## Which data are we going to re-analyze?

We will obtain data for a terrestrial metagenome project with id `PRJNA278302`




The URL for the bioproject is at:

- https://www.ncbi.nlm.nih.gov/bioproject/PRJNA278302

## How do I get more information on the project?

The `runinfo` and `summary` formats contain information on the runs:

```
esearch -db sra -query PRJNA278302 | efetch -format runinfo > runinfo.txt
esearch -db sra -query PRJNA278302 | efetch -format summary > summary.xml
```

In the latter case the result is an XML file that you could view in a browser:

```
▼<EXPERIMENT_PACKAGE_SET>
  ▼<EXPERIMENT_PACKAGE>
    ▼<EXPERIMENT xmlns="" alias="S122_K3" accession="SRX997729" center_name="Lawrence Berkeley National Lab">
      ▼<IDENTIFIERS>
          <PRIMARY_ID>SRX997729</PRIMARY_ID>
          <SUBMITTER_ID namespace="Lawrence Berkeley National Lab">S122_K3</SUBMITTER_ID>
        </IDENTIFIERS>
      ▼<TITLE>
          metagenome K3(S122) from petroleum reservoir (this well has been treated with materials containing sulfide),
          Kuparuk formation, Alaska North Slope
        </TITLE>
      ▼<STUDY_REF accession="SRP057267">
        ▼<IDENTIFIERS>
            <PRIMARY_ID>SRP057267</PRIMARY_ID>
          </IDENTIFIERS>
        </STUDY_REF>
      ▼<DESIGN>
        ▼<DESIGN_DESCRIPTION>
            DNA was extracted from filters having produced water passed through them using a bead-beating and organic solvent
            extraction protocol. DNA was purified using a MoBio UltraSoil DNA extraction kit starting by adding Solution S3 to
            the extract and continuing per the manufacturer's protocol. DNA was prepared for sequencing using an Illumina
            sequencing kit (paired-end, 150 bp reads) for HiSeq instrument. Library preparation was performed by the Yale
            Center for Genome Analysis.
          </DESIGN DESCRIPTION>
```

Visualizing this file will allow you to understand its (convoluted) structure. Then you can use the `xtract` tool may be used to parse this XML file and extract various components of it and format those as a tabular file.

```
cat summary.xml | xtract -pattern "EXPERIMENT_PACKAGE" -element PRIMARY_ID, TITLE | cut -f 8,10
```

will connect the SRR run numbers to the title of each section.

```
SRR1977365    metagenome K3(S122) from petroleum reservoir (this well has been treated with ma
terials containing sulfide), Kuparuk formation, Alaska North Slope
SRR1977357    metagenome K2 (MPF) from not soured petroleum reservoir, Kuparuk formation, Alas
ka North Slope
SRR1977304    metagenome I2(PW) from soured petroleum reservoir, Ivishak formation, Alaska Nor
th Slope
SRR1977296    metagenome I1 (DS1215) from not soured petroleum reservoir , Ivishak formation,
Alaska North Slope
SRR1977249    metagenome SB2 from not soured petroleum reservoir, Schrader bluffer formation,
Alaska North Slope
SRR1976948    metagenome SB1 from not soured petroleum reservoir, Schrader bluffer formation,
Alaska North Slope
```

# How do I classify this data?

We can employ several strategies discussed in the previous chapters, though in general you should not be surprised if these don't quite work at the level of accuracy of precision that you were expecting:

```
# Get the data for one run. Add the -X 100000 flag to limit the data to a subset.
fastq-dump --split-files -skip-technical  SRR1976948
```

Now it is worth checking out just how many of our sequences can be classified before and after quality control:

```
kraken -db /export/refs/kraken_db/kraken_std_db/ --threads 16 -fastq-input SRR1976948_1.fastq
SRR1976948_2.fastq > results.out
```

in just 3 minutes the tool completes with the following result:

```
34456404 sequences (8648.56 Mbp) processed in 175.190s (11800.8 Kseq/m, 2962.00 Mbp/m).
  3720501 sequences classified (10.80%)
  30735903 sequences unclassified (89.20%)
```

It reports that that majority of the reads could not be classified.

# How many species were present?

```
kraken-report -db /export/refs/kraken_db/kraken_std_db/ results.out > results.txt

cat results.txt | awk ' $2>10 && $4=="S" { print $0}' | wc -l
1230
```

that starts with:

```
3.47  1196351 0       S       301375   Methanosaeta harundinacea
0.06  20863   0       S       2223     Methanosaeta concilii
0.01  4390    0       S       420950   Methanolobus psychrophilus
0.00  1136    0       S       2209     Methanosarcina mazei
0.00  571     0       S       2214     Methanosarcina acetivorans
0.00  377     0       S       2208     Methanosarcina barkeri
0.00  734     0       S       101192   Methanomethylovorans hollandica
0.00  619     0       S       2176     Methanohalophilus mahii
0.00  170     0       S       29291    Methanococcoides burtonii
0.00  73      0       S       2322     Methanohalobium evestigatum
```

# Does quality control improve the classification?

Let us apply quality control and see how the classification performance may change with that.

```
# Make the illumina adapter file.
echo ">illumina" > adapter.fa
echo "AGATCGGAAGAGCACACGTCTGAACTCCAGTCAC" >> adapter.fa

# Let apply adapter trimming and quality correction.
trimmomatic  PE -baseout trimmed.fq SRR1976948_1.fastq SRR1976948_2.fastq ILLUMINACLIP:adapter
.fa:2:30:5 SLIDINGWINDOW:4:30

# Error correction with tadpole
tadpole.sh in=trimmed_1P.fq in2=trimmed_2P.fq out=corrected_1P.fq out2=corrected_2P.fq mode=co
rrect k=50
```

Classify after error correction:

```
# Classify with kraken
kraken -db /export/refs/kraken_db/kraken_std_db/ --threads 16 -fastq-input corrected_1P.fq  co
rrected_2P.f > results-corrected.out
```

the result is unexpected! The classification produces fewer results overall only 2424 sequences were classified instead of 7424:

```
78272 sequences (14.88 Mbp) processed in 5.743s (817.7 Kseq/m, 155.45 Mbp/m).
  2424 sequences classified (3.10%)
  75848 sequences unclassified (96.90%)
```

In all honesty we don't know how to explain this. Chalk it up to one of the many mysteries you will undoubtebly run into in your career.The first rule of the "Bioinformatics Fight Club" is that there are no rules.

We can format the result differently, though that alone won't help with the results:

```
  kraken-report -db ~/refs/kraken/minikraken_20141208 results.out > results.txt
```

to filter at species level with at least 10 hits we can do:

```
cat results.txt | awk ' $2>10 && $4=="S" { print $0}'
```

that will produce:

```
1.44    1442      0    S    28108     Alteromonas macleodii
0.49     494      0    S    85698     Achromobacter xylosoxidans
0.01      11      0    S    994695    Candidatus Kinetoplastibacterium galatii
0.07      66      0    S    316277    Syntrophus aciditrophicus
0.02      17      0    S    181663    Desulfococcus oleovorans
0.35     349      0    S    1184387   Mesotoga prima
0.10     105      0    S    69499     Petrotoga mobilis
0.04      44      0    S    102134    Desulfotomaculum gibsoniae
0.01      11      0    S    58135     Desulfotomaculum kuznetsovii
0.06      58      0    S    110500    Pelotomaculum thermopropionicum
0.01      12      0    S    233055    Desulfitobacterium dichloroeliminans
0.02      19      0    S    35701     Heliobacterium modesticaldum
0.02      21      0    S    336261    Thermovirga lienii
1.98    1980      0    S    301375    Methanosaeta harundinacea
0.02      18      0    S    2223      Methanosaeta concilii
0.83     827      0    S    2198      Methanoculleus marisnigri
0.45     445      0    S    83986     Methanoculleus bourgensis
0.14     140    140    S    1379702   Methanobacterium sp. MB1
0.41     411    411    S    374840    Enterobacteria phage phiX174 sensu lato
```

# Are there other methods to classify samples?

There is no shortage of approaches that claim to do better. In reality the results are typically decidedly less reliable.

# How do I use "sourmash" to understand my data?

Based on the tool description

> sourmash can also be used to quickly search large databases of genomes for matches to query genomes and metagenomes;

First install the database:

```
mkdir -p ~/ref/sourmash
cd ~/ref/sourmash
wget https://s3-us-west-1.amazonaws.com/spacegraphcats.ucdavis.edu/microbe-refseq-sbt-k21-2017
.05.09.tar.gz
tar xzvf microbe-refseq-sbt-k21-2017.05.09.tar.gz
```

You can now use `sourmash` to classify:

```
# Compute a signature.
sourmash compute --scaled 10000 -k 21 SRR1976948_1.fastq

# Classify the signature.
sourmash sbt_gather -k 21 ~/refs/sourmash/refseq-k21.sbt.json SRR1976948_1.fastq.sig
```

it will produce:

```
overlap    p_query p_genome
-------    ------- --------
130.0 kbp   0.9%      4.7%      NZ_LN515531.1 Methanobacterium formicicu
120.0 kbp   0.8%      4.9%      NC_009051.1 Methanoculleus marisnigri JR
60.0  kbp   0.4%      2.3%      NC_017527.1 Methanosaeta harundinacea 6A
50.0  kbp   0.3%      1.6%      NC_017934.1 Mesotoga prima MesG1.Ag.4.2,
```

so out of 100,000 reads it was able to classify just 3 percent of them into 4 categories.

# Installing metagenomics tools

## How to install taxonkit?

Taxonkit webpage: https://github.com/shenwei356/taxonkit

```
mkdir -p ~/.taxonkit
(cd ~/.taxonkit && curl -k https://ftp.ncbi.nlm.nih.gov/pub/taxonomy/taxdump.tar.gz | tar xzv )
```

On a mac install `findutils`

```
brew install findutils --with-default-names
```

## How do I install the "minikraken" database?

Get the "minikraken" database

```
mkdir -p ~/refs/kraken
(cd ~/refs/kraken && curl -k http://ccb.jhu.edu/software/kraken/dl/minikraken.tgz | tar zxv)
```

## How to build the

```
DBNAME=~/refs/kraken
MINIDB=~/refs/mini-kraken

kraken-build --download-taxonomy --db $DBNAME

kraken-build --download-library bacteria --db $DBNAME

kraken-build --build --db $DBNAME --work-on-disk --jellyfish-hash-size 640M --threads 4

kraken-build --build --db ~/refs/kraken --jellyfish-hash-size 640M

kraken-build --shrink 10000 --db $DBNAME --new-db $MINIDB
```

# Setting up a MacOS computer

## What is the first step?

First of all, ensure that your Mac runs the latest version of MacOS. Go to the "Apple Icon" -> "About This Mac" -> "Software Update". Verify that your Mac is updated to the latest version of MacOS.

## What compiler do I need?

A compiler is software that turns a "text" based program into one that the computer can execute. You will need to install the "XCode" compiler. Go to the "App Store" and find the "XCode" application. Install it.

You will also need to install an addition the command line tools to XCode by opening the "Terminal" application then writing the following into it:

```
xcode-select --install
```

This will trigger an installation procedure.

In addition, you may occasionally need to re-launch the XCode application whenever the licence agreement created by Apple changes. Notably you can accept a license agreement change from the command line by typing (requires your password):

```
sudo xcodebuild -license accept
```

The automatic software update will update XCode as well. In those cases you may need to re-accept the license.

# What is Homebrew?

HomeBrew is a MacOS specific package manager that we use to facilitate the installation of tools. While most of the book will use the `conda` package manager there are tools that are best installed with Homebrew. Visit the HomeBrew page for the installation script.

Once installed, a program named:

```
brew
```

will be available at the command line. This program can be used to install other programs like so:

```
brew install <program name goes here>
```

# How do I use Homebrew?

To use `brew` for bioinformatics you will need to "tap" the "science formulas":

```
# This is used to "tap" formulas used in science
# Needs to be done only once.
brew tap homebrew/science
```

# What software are required?

There are a number of important libraries that tools may require. It is best if these are installed right away.

First install the X11 libraries. Unlike most other installation commands, this may take a long time to finish (30 minutes or more), and it will ask for a root password.

```
brew cask install xquartz
```

Install other required libraries that we list below. These will finish much faster.

```
brew install gd libharu git imagemagick lzo hdf5 bison wget
brew install findutils --with-default-names
```

# How do I install Java?

The Java programming language is an important for many bioinformatics tools.

Make sure to install the **JDK** (Java Development Kit) and not the **JRE** (Java Runtime Environment) version of Java. The latest version of java can be installed with either:

```
brew cask install java
```

Or alternatively you may visit the Java JDK for MacOS page. Test your installation by typing:

```
java -version
```

# How do I upgrade software?

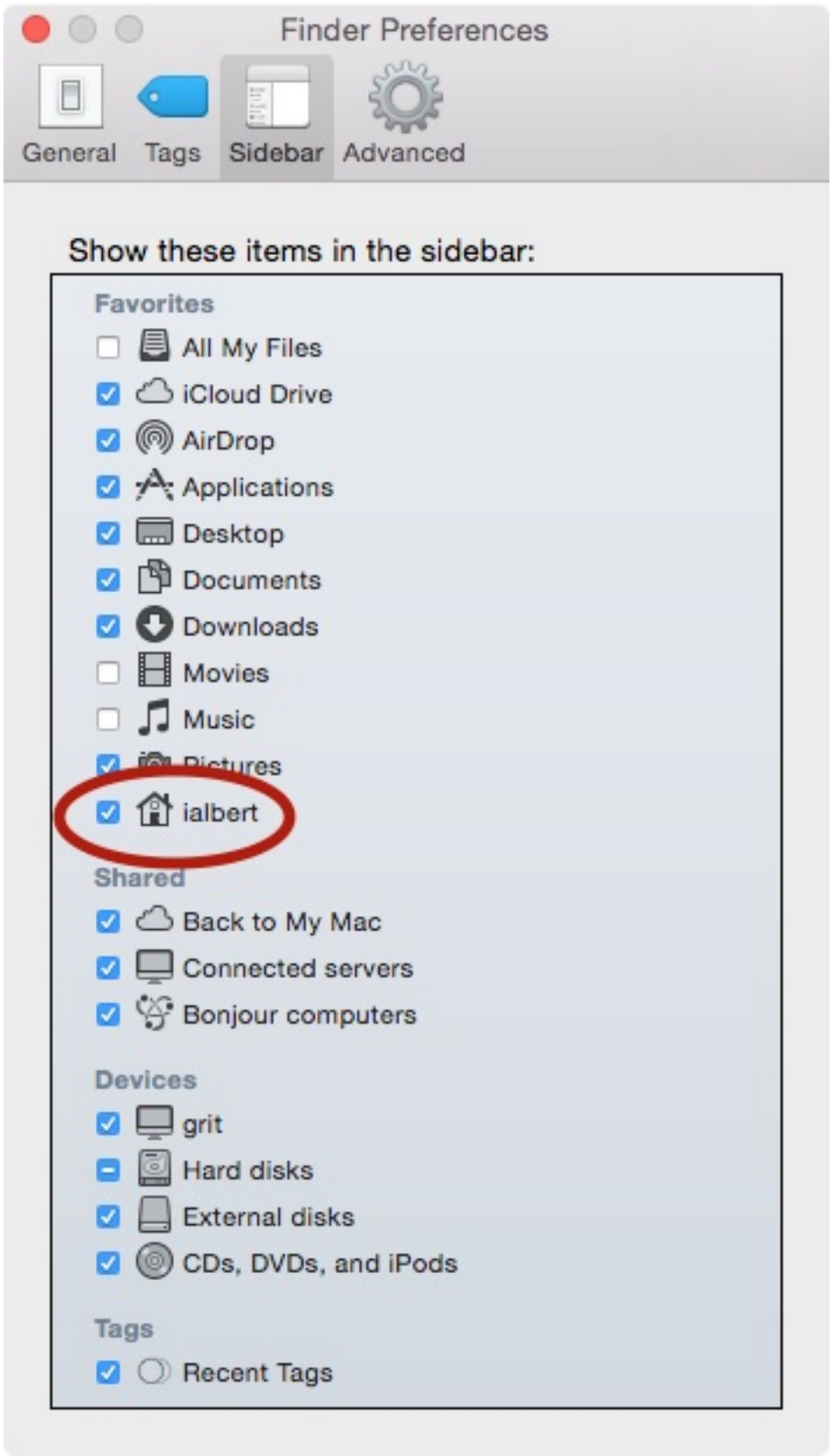
Libraries may become outdated in time in which case it need to be upgraded. Periodically, you many need to run the `brew upgrade` update these:

```
brew upgrade gd libharu git imagemagick lzo hdf5 bison wget findutils
```

# How do I view my home directory in the Finder?

By default the Finder will now show the home directory that your Terminal opens to.

To be able to view your home directory within the Mac Finder you need to tick a checkbox next to it. Go to Finder->Preferences. Your home directory is named as your username. My username is `ialbert`, yours will be different. See below:




## Will the Finder show all files?

No some files won't be visible via the Finder. For example the Finder will not show so called "hidden" files, these are files whose name starts with a period: `.`

There is nothing really hidden about these files, it is just a convention that Unix adopted. By default the system avoids showing file names that are typically used for configuration.

## What is the next step?

Continue of with the terminal setup

# Setting up Linux

## Which distributions of Linux does this book support?

There are numerous Linux distributions with various naming schemes. Our commands refer to `Ubuntu/Debian` based systems.

To set up their system users will need to install packages via so-called package managers: `apt-get` or `yum` on Fedora based systems.

## What is the first step for Linux?

1. Update your Linux system.
2. Install the required Linux libraries and Java.

## How do I update a Linux based system?

On Ubuntu Linux start a "Terminal" then run the following:

```
sudo apt-get update && sudo apt-get upgrade -y
```

This will update your distribution (while printing copious amounts of information on the screen) then upgrade all installed packages.

## What are the required libraries for Linux?

Depending on the Linux distribution's initial configuration one may need to install all, some, or none of the package shown below. On Ubuntu the following was necessary:

```
sudo apt-get install -y curl unzip build-essential ncurses-dev
sudo apt-get install -y byacc zlib1g-dev python-dev git cmake
sudo apt-get install -y default-jdk ant
```

## What is the next step?

Continue of with the terminal setup

# Setting up Windows 10

Starting with the August 2016 Anniversary Edition the Windows 10 operating system supports running Linux based software.

The Linux subsystem for Windows is a fully configurable Linux environment. What this means is that once Ubuntu on Windows is installed, users can continue setting up their computer as if it were a Linux system.

## How do I install Ubuntu on Windows?

There are different methods that you can follow.

Official documentation states that the Ubuntu subsystem may be installed as a Windows App. Some unstated requirements must also apply as we were not successful in doing so even when following the official documenation below. While the method described below did not work for us, it might work for you. Try this first:

- Bash on Ubuntu on Windows

We followed the following instructions that let us install Linux on Windows:

- Install the Windows Subsystem for Linux

Here are few older installation instructions, these may not apply anymore but might be helpful for those with older systems:

- Another explanation on installing bash on Ubuntu on Windows It is very similar to the first but it feels more user friendly.
- How to reinstall Ubuntu Bash. This guide is to reinstall the entire Ubuntu Bash if you wish to do so.
- How to get Ubuntu 16.06 on Windows Bash. This guide describes the steps for updating an Ubuntu 14.04 to Ubuntu 16.04

## How do I set up the filesystem with Ubuntu on Windows?

Typically you would want to be able to access the files from both Linux and with Windows Explorer. Setting this up will be somewhat circuitous.

Note that you don't HAVE to do it this way. It is just very HANDY to be able to access your files both from inside the Linux Terminal and via Windows Explorer.

The files on the Windows "side" of your computer are visible in your Linux terminal under the path called `/mnt/c/` .

For example to create a directory that is visible under both Windows and Linux type the following into your Terminal (replace `ialbert` with your username).

Creates a directory on the Desktop:

```
mkdir -p '/mnt/c/Users/ialbert/Desktop/unix'
```

Link this newly created directory into your home directory under Linux:

```
ln -s  '/mnt/c/Users/ialbert/Desktop/unix' ~/unix
```

Now you have a folder named "unix" both on your Desktop and inside the Linux terminal that opens into your home directory.

Do all you work from this `unix` folder from now on. All the files you create will by visible and accessible from both the Linux and Windows operating systems.

In the next chapters, we will instruct you to create folders at various locations. When using Windows Bash create these in this new `unix` folder instead.

# How do I finish setting up Ubuntu on Windows?

Once the Ubuntu subsystem is installed as above follow the Setup Linux page.

# Installing Conda

The Conda Conda is an open source package management system and environment management system for installing multiple versions of software packages and their dependencies and switching easily between them. It works on Linux, OS X and Windows, and was created for Python programs but can package and distribute any software.

We will use a version of it called `bioconda` customized to install bioinformatics packages. You will need to install `conda` then tap the channels that correspond to `bioconda`.

# How do I install conda?

Start with by installing miniconda the simplified version of this tool that can be installed as a simple executable program.

On MacOS (press space to page through the licensing agreement):

```
curl -O https://repo.continuum.io/miniconda/Miniconda3-latest-MacOSX-x86_64.sh
bash Miniconda3-latest-MacOSX-x86_64.sh
```

On Linux:

```
curl -O https://repo.continuum.io/miniconda/Miniconda3-latest-Linux-x86_64.sh
bash Miniconda3-latest-Linux-x86_64.sh
```

Close the terminal and open a new one to make `conda` activate. Check that it worked by typing:

```
conda
```

It should list a number of its options.

# What is the next step?

Continue of with the conda setup

# How to Install Everything

## How to install software?

Tool installation can be greatly automated for both Mac OS and Linux systems.

The typical installation instruction is:

```
conda install -y mytool
```

Occasionally on MacOS we may need to install the tool with `brew`:

```
brew install mytool
```

> In some (rare) cases brew and conda install a really old version of a software that may affect other tools as well. Usually there the installation will stop and there will be a warning - so read any notice that my pop up.

See the How to set up my computer page for a full system setup.

## How install from source?

In some cases software needs to be installed from the source. Sometimes the `brew` and `conda` packages may install an older version of the tool. At other times the software is only available in it's so called source code format.

This involves downloading the source code, unpacking the software and following the installation instructions. Sometimes this means running either `make` or some other command. We do list the source code installation for a few tools.

When we install from source we also need to update the `PATH` variable to make the new program more easily accessible. See also How to set the PATH

# What software tools are discussed in the Handbook?

The content below links to installation and usage instructions for every tool mentioned in the book. The list is grouped by the utility of the software package.

## Data sources

All data sources offer web based user interfaces. In order to automate the data access we need to be able to access these resources from the command line.

- EDirect: Entrez Direct ia a suite of scripts written in the Perl language that can be used to connect to the NCBI Entrez data interfaces
- SRA Toolkit is a suite of programs that can be used to download data from the Short Read Archive (SRA).
- ReadSeq: file format conversion is a Java based tool that can convert biological data from one format to another.
- EMBOSS Bioinformatics utilities are an open source software analysis package specially developed for the needs of the molecular biology.
- Jim Kent's tools are an a number of utility programs developed by Jim Kent of UCSC's fame. Some of these tools are the only existing converters of formats such as bigWig, bigBed

# Sequence Ontology

The Sequence Ontology (SO) is a collaborative ontology project for the definition of sequence features used in biological sequence annotation.

# Gene Ontology

A gene set enrichment analysis refers to the process of discovering potential common characteristics present in a list of genes.

There numerous web based Gene Ontology tools that do not require installation.

- AgriGO Web based GO Analysis Toolkit and Database for Agricultural Community.
- DAVID The GO tool biologists love. It has the "generous statistics" that always seem to find something.
- Panther Appears to be the "officially endorsed" GO enrichment tools. It is embedded into the official GO website. It produces only tabular text formats and no visualizations.

The rest below are standalone tools that install directly on a computer.

- ErmineJ is a Java based program to compute gene set enrichment.
- GOA Tools: file format conversion is a Python package that computes over and under representation of GO terms.

# Data quality control

Quality control typically follows an iterative process:

1. Evaluate (visualize) data quality.
2. Stop QC if the quality appears to be satisfactory.
3. Perform one or more data altering steps then go to step 1.

In addition the need to perform various sequence manipulations is very common. Software tools:

- FastQC is a Java program that allows the visualization of sequencing data.
- BBDuk an adaptor trimming and quality filtering tool. Part of the BBMap package.
- Trimmomatic adapter trimming and filtering.
- Cutadapt adapter trimming and filtering with a wide variety of adapter matching options.
- Seqtk Sequence Toolkit. A collection of efficient tools to manipulate and filter sequences.
- SamTools SamTools. A suite for manipulating SAM files.

# General aligners

- EMBOSS Aligners optimal alignment tools for local and global alignment.
- LASTZ is a generic pairwise alignment tool that can produce outputs in myriads of ways. It is a tool
- LAST (not to be confused with `lastz`) is also a generic pairwise alignment tool.

# BLAST

- BLAST Basic Local Alignment Search Tool (BLAST) a suite of aligners and other helper to perform fast searches for local alignments on databases of nucleotide and peptide sequences

# Short Read Aligners

- BWA Burrows-Wheeler Aligner, a software package for mapping low-divergent sequences against a large reference genome, such as the human genome.
- Bowtie is an ultrafast and memory-efficient tool for aligning sequencing reads to long reference sequences. It is particularly good at aligning reads of about 50 up to 100s or 1,000s of characters, and particularly good at aligning to relatively long (e.g. mammalian) genomes
- BBMap A suite of programs including short read alignment.

# Variation calling

- Bcftools Variant caller and VCF filtering tools.
- FreeBayes Bayesian variant and halplotype caller.
- GATK Genome Analysis Toolkit.
- snpEff Variant effect prediction toolbox.

# RNA-Seq analysis

- Using Tophat
- Using CuffLinks
- Using HiSat
- Using featureCounts
- Using kallisto

## Interval analysis

## Assemblers

# How to set up your Bash profile

You may want to apply certain settings automatically when a terminal is opened.

## What is a "shell"?

As the Wikipedia page on Shell(computing)) states:

> In computing, a shell is a user interface for access to an operating system's services. In general, operating system shells use either a command-line interface (CLI) or graphical user interface (GUI), depending on a computer's role and particular operation. It is named a shell because it is a layer around the operating system kernel.

## What is Bash?

Bash is a Unix shell and command language written for the GNU project as a free software replacement for the Bourne shell. The name is an acronym for "Bourne-again shell".

## What are shell profiles?

A file that is processed when a shell is opened is called a *shell profile*.

This file has to have a special name and has to be located in the home directory. All the commands listed in this shell profile file will be applied when a new shell is initialized.

## Can we have multiple shell profiles?

Yes. Due to historical and other usage considerations, multiple files may contain shell configurations. There are rules regarding which of these settings are applied when more than one shell profile file is present.

Josh Staiger in .bash_profile vs .bashrc writes:

> When you login (type username and password) via console, either sitting at the machine, or remotely via ssh: `.bash_profile` is executed to configure your shell before the initial command prompt. But, if you've already logged into your machine and open a new terminal window (xterm) inside Gnome or KDE, then `.bashrc` is executed before the window command prompt. `.bashrc` is also run when you start a new bash instance by typing `/bin/bash` in a terminal.
>
> An exception to the terminal window guidelines is Mac OS X's Terminal.app, which runs a login shell by default for each new terminal window, calling `.bash_profile` instead of `.bashrc`. Other GUI terminal emulators may do the same, but most tend not to."

# What's the best setup for multiple shell profiles?

You should handle this in the following way:

1. The file called `.bash_profile` should be set up to automatically load `.bashrc`.
2. The main shell profile file that you use should be `.bashrc` and it should contain all shell related settings.

The easiest way to set this up is to add the following lines to `.bash_profile`:

```
if [ -f ~/.bashrc ]; then
   source ~/.bashrc
fi
```

This command loads `.bashrc` when loading the `.bash_profile`. You will never need to look at the `.bash_profile` file again. From now on you need to only change the `.bashrc` file.

Both of these files probably already exist on your system, if not you can create them as below.

# How can I create the minimal settings?

You may also apply our recommended settings with the following (apply only once!):

```
curl http://data.biostarhandbook.com/install/bash_profile.txt >> ~/.bash_profile
curl http://data.biostarhandbook.com/install/bashrc.txt >> ~/.bashrc
```

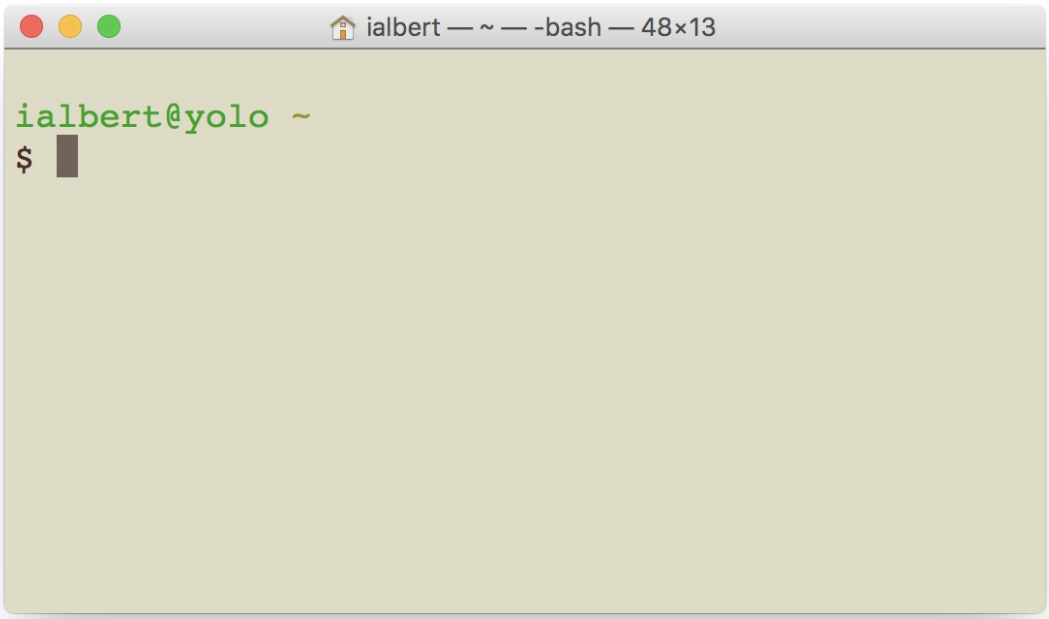
# What should my .bashrc file contain?

Minimally we recommend the following:

```
#
# A minimal BASH profile.
#

# ON Mac OS uncomment the line below.
# alias ls='ls -hG'

# On Linux use the following:
# alias ls='ls -h --color'

# Safe versions of the default commands.
# Will ask permissions before overwriting files.
alias rm='rm -i'
alias mv='mv -i'
alias cp='cp -i'

# Extend the program search PATH and add the ~/bin folder.
export PATH=~/bin:$PATH

# Makes the prompt much more user friendly.
# But I do agree that the command looks a bit crazy.
export PS1='\[\e]0;\w\a\]\n\[\e[32m\]\u@\h \[\e[33m\]\w\[\e[0m\]\n\$ '

# This is required for Entrez Direct to work.
# Disables strict checking of an encryption page.
export PERL_LWP_SSL_VERIFY_HOSTNAME=0
```

## Can I customize the terminal colors and fonts?

Yes, and please do so. You can vary font sizes, background colors etc. It makes a tremendous difference when reading it for extended periods of time. Here is an easy to read terminal:




## How do I edit my shell profile?

You can use the `nano` editor

```
nano ~/.bash_profile
```

then:

```
nano ~/.bashrc
```

## How do I activate a shell profile?

Changing a shell profile file will not automatically apply the changed settings to terminals that are already open.

New terminals will use the new settings but existing terminal windows need to be instructed to apply the new settings by sourcing the file in question:

```
source ~/.bashrc
```

## What is the next step?

Continue of with the computer setup

# How do I set up my PATH?

One of the most frustrating issues faced by newcomers to the Unix command line is how to run a program they have just installed. This typically happens if they install a program from source instead of via a global installer.

Suppose you install the program `fastq-dump` but then as you try to run it you could get this:

```
$ fastq-dump
fastq-dump: command not found
```

The main concept to remember here is that our computers look at only a few locations when trying to find a program. When bash cannot run a program it is because it cannot "see" the program. When we install a new program ourselves, we need to tell our computer where to look for it. There are different ways to do this:

## Solution 1: Use the full program path

Run the program by its fully qualified path (location). First you need to figure out where that is. Go to the location where you installed the software. From that software directory, type `pwd` and that will tell you the full path that you can then use:

```
/Users/ialbert/src/sratoolkit.2.5.2-mac64/bin/fastq-dump
```

We may use a shorter form when the program directory opens from our home directory:

```
~/src/sratoolkit.2.5.2-mac64/bin/fastq-dump
```

If that works, then we can always invoke the program by this full name. It is a bit hard on eyes. For a simpler way, see the next solution.

## Solution 2: Extend the search PATH to include the installation directory.

If we wanted to be able to run the program just by typing its name (say `fastq-dump` ) we need to instruct the computer to also look for programs to run in the `/Users/ialbert/src/sratoolkit.2.5.2-mac64/bin/` directory. To do so, we need add this location to the so called *search path* of the shell.

The shell looks at the variable called `PATH` and searches all locations listed in the `PATH` for names of programs to run. What is the current value of the `PATH` ? Type the following:

```
echo $PATH
```

This will produce a colon `:` separated list of all the locations that bash will look in when searching for a program. We can extend this by adding the new location via the following construct:

```
export PATH=~/src/sratoolkit.2.5.2-mac64/bin:$PATH
```

The new `PATH` will be constructed as the new directory + `:` + old value of the `PATH`. Now we should be able to run the program as

```
fastq-dump
```

Now, the above is a solution, but typing this (and all other) `export` commands we might add every time we open a new terminal is too tedious. We would be better off applying the `PATH` and other settings automatically. To do this, we must take one more step down the system admin rabbit hole. In this case, you need to append this information to the so-called *shell profile* as described in How to set the profile.

# Solution 3: Create shortcuts

There are many cases when we only need to access a single program, and adding the whole directory to the path is tedious and error prone. In this case, the solution is to first create one directory that is being searched by our bash shell, then create symbolic links (shortcuts) from the programs we want to run in this `~/bin` directory. Since the directory is already on the search path, any new programs added there will also become accessible.

```
# Creates the ~/bin directory
mkdir -p ~/bin

# Add this directory to the search path.
export PATH=~/bin:$PATH

#
# Link fastq-dump program into the ~/bin folder
#
# Syntax: ln -s source destination
#
ln -s ~/src/sratoolkit.2.5.2-mac64/bin/fastq-dump ~/bin/fastq-dump
```

Of course we still need to add `~/bin` to our path. But now we only have to do it once. After that, linking the program into `~/bin` will make it available everywhere. The export command should be:

```
export PATH=~/bin:$PATH
```

To load these settings automatically see the page How to set the profile

# What is the next step?

Continue of with the computer setup

# Troubleshooting the PATH

It is surprisingly easy to accidentally break everything so thoroughly that not even basic UNIX commands work anymore.

```
$ ls
-bash: ls: command not found
```

Sooner or later we all do this. Wear it as a badge of honor.

# How can this happen?

If you mistype the `PATH` variable while adding it into the `.bashrc` shell profile you may end up with the unenviable situation of applying an incorrect setting every time your terminal initializes. For example, on a beautiful and sunny morning you'll do a:

```
echo 'export PATH=~/bin:PATH' >> ~/.bashrc
```

It all looks great, but there is a fatal typo there. The net effect of it all will be that even basic system programs that used to work before won't do so anymore:"

```
$ ls
-bash: ls: command not found
```

**Congratulations!** You've just **nuked** your `PATH`. You forgot the `$` sign on the right hand side of the `PATH` variable. Instead of using the value of variable named `$PATH` you are setting it to be the actual word `PATH`. Your command should have looked like this:

```
echo 'export PATH=~/bin:$PATH' >> ~/.bashrc
```

The difference is easy to see on this website since it is colored differently. It may not be so easy to notice in your file.

First let's look at the bright side -- there is probably no command line guru out there that did not do this at one point in their career. Obviously you are well on your way towards "guru-ness". Rock on!

Jules Winnfield of Pulp Fiction explains it best:

> The `PATH` to any righteous tool is beset on all sides by the inequities of `UNIX` and the tyranny of Environment Variables. Blessed is he who, in the name of charity and good will, shepherds the weak through the valley of darkness, for he is truly his brother's keeper and the finder of lost programs.

# How to fix this?

Access your terminal. Oh yes, it is still broken. But you can temporarily override the `PATH` within this terminal with:

```
PATH=/bin:/usr/bin
```

Now edit your `.bashrc` with `nano` and remove the lines that cause the trouble:

```
nano ~/.bashrc
```

Now `nano` is a bit quirky for a text editor. Take it as punishment meted out for clobbering your path. It could be worse. At least you don't have to use `vi` (But some of us love vi!).

Edit the file, correct the offending line(s), then save the file with `ctrl + O` (that is a capital letter O), then exit the editor with `ctrl + X`.

Open a NEW terminal window and verify that your "corrections" work. In times like these it is very easy to make another error while "fixing" the first one. Rinse and repeat until everything checks out. It will eventually.

**Good luck, comrade!**

# Entrez Direct

Entrez Direct consist of a suite of scripts written in the Perl language that can be used to connect to the NCBI Entrez data interfaces.

The suite includes commands that coresponds to each of the Entrez API endpoints: `esearch`, `efetch`, `elink`, `efilter`, `epost`, `einfo` as well as a tool called `xtract` that can be used to extract elements of an XML document.

# Documentation:

- Entrez Direct Book: E-utilities on the UNIX Command Line
- Entrez Direct Release Notes

# Installation

Note: There is a program called `efetch` included with some versions of Linux that is named identically but is unrelated to the `efetch` program from Entrez Direct. You may even be prompted to install that by your operating system. **DO NOT** use `apt-get` to install `efetch`. it will be a different, unrelated program.

Entrez Direct will change more than once during a year. There is no obvious way to tell that your version is outdated other than installing a newer version. In addition package managers such as `conda` and `brew` may distribute older versions of the code.

If you find that any of the programs produce errors make sure to first reinstall upgrade the existing code or reinstall from source.

On Mac OS:

```
brew install edirect
```

On Linux:

```
conda install -y entrez-direct
```

# Source code install

Do this when your previous installation does not seem to work.

```
mkdir -p ~/src
# Get the source code.
curl ftp://ftp.ncbi.nlm.nih.gov/entrez/entrezdirect/edirect.zip > ~/src/edirect.zip
# Unzip into  ~/src
unzip ~/src/edirect.zip  -d ~/src

# This is a suite of tools thus is best to add the entire directory to the search path.
# Refer to the "How to set the PATH" section of the book.
echo 'export PATH=~/src/edirect:$PATH' >> ~/.bashrc
source  ~/.bashrc

# The tools that can be run are visible in source folder.
ls ~/src/edirect
```

# Test

Fetch the ebola genome:

```
efetch -version
```

On my system it prints:

```
5.50
```

Test obtaining a dataset:

```
efetch -db=nuccore -format=fasta -id=AF086833 | head
```

# Troubleshooting

You may get an error of the form: `500 Can't verify SSL peers without knowing which Certificate Authorities to trust`

This is caused by accessing an encrypted page without a certificate being installed. Set the following variable to squash this error message:

```
export PERL_LWP_SSL_VERIFY_HOSTNAME=0
```

# EFetch Problems

Depending on your setup installing efetch may be too complicated. Occasionally a number of libraries are missing from the system and the user does not have permission (or expertise) to install them. We wrote a very simple replacement script that can fetch data in case `efetch` cannot be installed. We call it `wonderfetch`:

```
mkdir -p ~/bin
curl http://data.biostarhandbook.com/scripts/wonderfetch.sh > ~/bin/wonderfetch
chmod +x ~/bin/wonderfetch
```

In those cases you may be able to replace many instances of `efetch` with `wonderfetch`. It takes three parameters, database accession format like so:

```
wonderfetch  nucleotide  AF086833  fasta
```

The sole purpose of this script is to allow you to execute `efetch` commands in a manner similar to as was listed in the book.

# How does the wonderfetch work?

Entrez Direct offers a so called web API that can be accessed via URL like the following:

```
curl -k "https://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db=nuccore&id=AF086833&rettype=fasta"
```

The `wonderfetch` program automates generating this link and accesses it via `curl`.

# Entrez database names

```
Entrez Database    UID common name    E-utility Database Name
BioProject         BioProject ID         bioproject
BioSample          BioSample ID          biosample
Biosystems         BSID                  biosystems
Books              Book ID               books
Conserved Domains  PSSM-ID               cdd
dbGaP              dbGaP ID              gap
dbVar              dbVar ID              dbvar
Epigenomics        Epigenomics ID        epigenomics
EST                GI number             nucest
Gene               Gene ID               gene
Genome             Genome ID             genome
GEO Datasets       GDS ID                gds
GEO Profiles       GEO ID                geoprofiles
GSS                GI number             nucgss
HomoloGene         HomoloGene ID         homologene
MeSH               MeSH ID               mesh
NCBI C++ Toolkit   Toolkit ID            toolkit
NCBI Web Site      Web Site ID           ncbisearch
NLM Catalog        NLM Catalog ID        nlmcatalog
Nucleotide         GI number             nuccore
OMIA               OMIA ID               omia
PopSet    PopSet ID    popset
Probe    Probe ID    probe
Protein    GI number    protein
Protein Clusters    Protein Cluster ID    proteinclusters
PubChem BioAssay    AID    pcassay
PubChem Compound    CID    pccompound
PubChem Substance    SID    pcsubstance
PubMed    PMID    pubmed
PubMed Central    PMCID    pmc
SNP    rs number    snp
SRA    SRA ID    sra
Structure    MMDB-ID    structure
Taxonomy    TaxID    taxonomy
UniGene    UniGene Cluster ID    unigene
UniSTS    STS ID    unists
```

# The SRA Toolkit

The SRA tookit allows access to the Short Read Archive. It is a suite of tools, the most used among them are: `fastq-dump` and `sam-dump`. The SRA toolkit is distributed as binary programs for each platform.

- The SRA Handbook contains a detailed description of the Short Read Archive.
- The SRA Toolkit Documentation describes the command line tools that may be used to access data.
- SRA Toolkit GitHub is the software development homepage.

## Installation

```
brew install sratoolkit
```

or

```
conda install -y sra-tools
```

## Executables

The SRA toolkit is also available as executable programs for each platform. Visit the SRA Download Page to find the latest version of the software.

For example to install the latest version on Linux

```
mkdir -p ~/src
cd ~/src
curl -kOL https://ftp-trace.ncbi.nlm.nih.gov/sra/sdk/current/sratoolkit.current-ubuntu64.tar.g
z
tar xzvf sratoolkit.current-ubuntu64.tar.gz

# Update the PATH.
# You may need to change this when the version changes.
echo 'export PATH=~/src/sratoolkit.2.8.0-ubuntu64/bin:$PATH' >> ~/.bashrc

# Activate in the current shell.
source ~/.bashrc

# The tools that can be run are visible in:
ls ~/src/sratoolkit.2.5.2-ubuntu64/bin/
```

## Testing

Find out what version of `fastq-dump` you have:

```
fastq-dump -h
```

Test installation by running:

```
fastq-dump -X 5 -Z SRR390728
```

# How to manually obtain SRA files

There is a website called "instant reads" at ftp://ftp-trace.ncbi.nih.gov/sra/sra-instant that uses an path encoding system to map a SRR number to a URL for example `SRR1972739` will be mapped to `RR197/SRR1972739/SRR1972739.sra` can be downloaded as:

```
curl -O ftp://ftp-trace.ncbi.nih.gov/sra/sra-instant/reads/ByRun/sra/SRR/SRR197/SRR1972739/SRR1972739.sra
```

This will download a file named `SRR1972739.sra` . This file can be operated on the same way with fastq-dump:

```
fastq-dump -X 10000 --split-files SRR1972739.sra
```

# Bash on Windows

As it happens the internal implementation of `fastq-dump` precludes it from working on Windows. Use a helper script that we have created and call it the `wonderdump` :

```
curl http://data.biostarhandbook.com/scripts/wonderdump.sh > ~/bin/wonderdump
chmod +x ~/bin/wonderdump
```

Then enjoy unfettered fast and efficient access to the SRA files with:

```
 wonderdump SRR1972739 -X 10000 --split-files
```

To download batches of SRA ids put them into a file then run:

```
cat samples.txt | xargs -n 1 echo wonderdump | bash
```

It is magic I tell you!

# Troubleshooting

The SRA toolkit is notoriously obtuse in its error reporting.

For example asking for an invalid run won't just simply say cannot find run id. Instead it will produce a confusing error message on *"constructing within virtual database modules"*.

```
$ fastq-dump -X 5 -Z foobar
2016-09-19T13:24:33 fastq-dump.2.7 err: item not found while constructing within
virtual database module - the path 'foobar' cannot be opened as database or table
```

Some mysterious errors may go away after the magic incantation:

```
vdb-config --restore-defaults
```

To the change other settings visit the nerdy and confusing interface that is accessible via:

```
vdb-config -i
```

# ReadSeq

ReadSeq is a tool that can convert biological data from one format to another. For example `GenBank -> Fasta`. Remember that some format changes don't make sense for example `Fasta -> GFF` hence in those cases there will be no output.

- Readseq Official webpage
- Readseq Documentation

The tool requires the presence of the Java programming language

# Installation

Automated install for MacOS only.

```
brew install readseq
```

# All other platforms

```
mkdir -p ~/src/readseq
cd ~/src/readseq
curl -OL http://iubio.bio.indiana.edu/soft/molbio/readseq/java/readseq.jar

# The program can be run as:
java -jar ~/src/readseq/readseq.jar

# To simplify the invocation create a script in the ~/bin folder:
echo '#!/bin/bash' > ~/bin/readseq
echo 'java -jar ~/src/readseq/readseq.jar $@' >> ~/bin/readseq
chmod +x ~/bin/readseq
```

Test installation by running:

```
readseq
```

# Example usage

```
# First get a data in Genbank format
efetch -db=nuccore -format=gb -id=AF086833 > AF086833.gb
```

By default `readseq` will generate filenames from the input file. This can be confusing and annoying. For clarity we run it in piped mode `-p` flag that takes input from standard input and writes to the screen:

```
# Convert GenBank to FASTA
# This will extract the genomic DNA sequence in the Genbank.
cat AF086833.gb | readseq -p -format=FASTA | head

# Extract the sequences corresponding to CDS features.
# This generates a FASTA record for each CDS.
cat AF086833.gb | readseq -p -format=FASTA -feat=CDS | head

# Convert GenBank to an interval format GFF.
# All features will be present as coordinates.
cat AF086833.gb | readseq -p -format=GFF | head

# Convert only CDS (coding sequences to GFF)
# Only the CDS fields of the GFF will included.
cat AF086833.gb | readseq -p -format=GFF -field=CDS | head
```

# EMBOSS Bioinformatics Utilities

EMBOSS is **The European Molecular Biology Open Software Suite** an open source software analysis package specially developed for the needs of the molecular biology. EMBOSS offers programs covering areas such as:

- Sequence alignment
- Rapid database searching with sequence patterns
- Protein motif identification, including domain analysis
- Nucleotide sequence pattern analysis: for example to identify CpG islands or repeats
- Codon usage analysis for small genomes
- Rapid identification of sequence patterns in large scale sequence sets
- Website http://emboss.sourceforge.net
- Publication: EMBOSS: The European Molecular Biology Open Software Suite

## Installation

```
brew install emboss
```

or

```
conda install -y emboss
```

## Test

```
needle -h
```

## Alignments

These shortcuts make running and visually inspection alignments a lot simpler

```
# Create a folder for scripts.
mkdir -p ~/bin

# Install the wrappers for the EMBOSS aligners.
curl http://data.biostarhandbook.com/align/global-align.sh > ~/bin/global-align.sh
curl http://data.biostarhandbook.com/align/local-align.sh > ~/bin/local-align.sh

# Make the scripts executable.
chmod +x ~/bin/*-align.sh
```

You can now run:

```
global-align.sh THISLINE ISALIGNED
```

as well as:

```
local-align.sh THISLINE ISALIGNED
```

# Tips for Emboss tools

Emboss is an advanced software package that unfortunately has an awkward user interaction model.

The authors of this package have been trying to accommodate "non-technical" users - alas that happened at the expense of adding default behaviors that in the end only get in the way for both newbies and experts. By default the programs will continually interrupt your thought processes by asking for input: "Enter this", "Enter that" - the program is is trying to help - but does not do it the right way.

Use the `-filter` parameter on every tool to make them operate on input/output streams. That will shut off a good number of interruptions.

# Example usage

Get a data in Genbank format (this is an Entrez direct tool):

```
efetch -db nucleotide -format=gb -id=AF086833 > AF086833.gb
```

Get information on a sequence:

```
cat AF086833.gb | infoseq -filter
```

Convert to FASTA format:

```
cat AF086833.gb | seqret -filter | head
```

Select sub-sequence from 10 to 20:

```
cat AF086833.gb | seqret -filter -sbegin 10 -send 20
```

Select sub-sequence from 10 to 20 and revert complement:

```
cat AF086833.gb | seqret -filter -sbegin 10 -send 20 -srev
```

Using negative values will return sequences as if were counting from the end. This selects last ten bases:

```
cat AF086833.gb | seqret -filter -sbegin -10 -send -1
```

Replace the sequence id:

```
cat AF086833.gb | seqret -filter -sid FOOBAR | head
```

Convert to GFF3 format:

```
cat AF086833.gb | seqret -filter -feature -osformat gff3 > AF086833.gff
```

Convert to FASTA format:

```
cat AF086833.gb | seqret -filter -osformat fasta | head
```

Extract sequence features, the sequence corresponding to a feature tagged a certain way:

```
cat AF086833.gb | extractfeat -filter -join -type CDS  | head
```

Extract the stop codon of each CDS feature:

```
cat AF086833.gb | extractfeat -filter -join -type CDS  | seqret -filter -sbegin -3
```

Introduce mutations into the genome:

```
cat AF086833.gb | msbar -filter -point 4 | head
```

# WGsim/DWGsim

**Wgsim** (whole genome simulator) is a tool for simulating sequence reads from a reference genome. It is able to simulate diploid genomes with SNPs and insertion/deletion (INDEL) polymorphisms, and simulate reads with uniform substitution sequencing errors. It does not generate INDEL sequencing errors, but this can be partly compensated by simulating INDEL polymorphisms.

Wgsim outputs the simulated polymorphisms, and writes the true read coordinates as well as the number of polymorphisms and sequencing errors in read names. One can evaluate the accuracy of a mapper or a SNP caller with wgsim_eval.pl that comes with the package.

Webpage: https://github.com/lh3/wgsim

**DWGsim** is similarly named tool written by Nils Homer inspired by `wgsim`. This tool offers more options and it can generate reads corresponding to different sequencing platforms and produces its results in VCF format. `dwgsim` also has a more streamlined installation instructions.

Webpage: https://github.com/nh13/DWGSIM

# Installing DWGSim

```
brew install dwgsim
```

The conda version of `dwgsim` will roll back samtools to a previous version so on Linux you need to install from source.

# Testing

```
dwgsim
```

# Information on mutations

The information on the mutations that a read was subjected to is embedded into the read name in the form: `chr1_6181150_6181637_2:0:0_2:0:0_2/1`:

> By default, wgsim simulates a diploid genome with indels and then simulates reads from the simulated genome without indels. In the example above, the first 2:0:0 indicates that the first read has two sequencing errors and that no substitution and no gaps are involved in the simulated genome. The second 2:0:0 is for the second read. As sequencing errors and substitutions are added separately, they may overlap. You may apply "-e 0.0" to disable sequencing errors.

Both the `wgswim` and `dwgsim` distributions come with evaluation scripts `wgsim_eval.pl` and `dwgsim_eval` that can be used evaluate mapping qualities of SAM files where the reads names are encoded as above. The scripts will parse the read name and compare the results to the alignment and, from that establish mapping accuracy.

# Source code installation

Source code install for `wgsim`:

```
cd ~/src
git clone https://github.com/lh3/wgsim.git
cd wgsim
gcc -g -O2 -Wall -o wgsim wgsim.c -lz -lm

# Link the program to your bin folder
ln -s ~/src/wgsim/wgsim ~/bin/wgsim
```

A nicely formatted manual can be accessed via:

```
more ~/src/wgsim/README
```

Source install for dwgsim:

```
cd ~/src
git clone --recursive https://github.com/nh13/DWGSIM
cd DWGSIM
make

# Link the program to your bin folder
ln -s ~/src/DWGSIM/dwgsim ~/bin/dwgsim
```

# ErmineJ - geneset enrichment software

ErmineJ performs analyses of gene sets in high-throughput genomics data such as gene expression profiling studies. A typical goal is to determine whether particular biological pathways are "doing something interesting" in an experiment that generates long lists of candidates. The software is designed to be used by biologists with little or no informatics background.

Documentation: http://erminej.chibi.ubc.ca/

## Installation

This tool requires Java.

Due to the various security settings Java may have the webstart program may not launch the tool.

Visit the website and download the `erminej.jnlp`. Double clicking the program may not work, again due to security settings. You may need to right click and select "Open" manually.

## Download annotations

Visit http://chibi.ubc.ca/microannots/ to download the annotation in the format that your gene ids are in.

# GOA Tools: Python scripts to find enrichment of GO terms

`goatools` is a Python package with the following functionality:

- Process over and under representation of certain GO terms, based on Fisher's exact test.
- Implements numerous multiple correction routines including locally implemented routines for Bonferroni, Sidak, Holm, and false discovery rate.
- Includes multiple test corrections from `statsmodels` Python package: FDR Benjamini/Hochberg, FDR Benjamini/Yekutieli, Holm-Sidak, Simes-Hochberg, Hommel, FDR 2-stage Benjamini-Hochberg, FDR 2-stage Benjamini-Krieger-Yekutieli, FDR adaptive Gavrilov-Benjamini-Sarkar, Bonferroni, Sidak, and Holm.
- Process the obo-formatted file from Gene Ontology website. The data structure is a directed acyclic graph (DAG) that allows easy traversal from leaf to root.
- Read GO Association files: - Read GAF (GO Association File) files. - Read NCBI's gene2go GO association file. map GO terms (or protein products with multiple associations to GO terms) to GOslim terms (analog to the `map2slim.pl` script supplied by http://www.geneontology.org).

Documentation: https://github.com/tanghaibao/goatools

## Installation

This tool requires Python. Read the instructions for Python for information on how to set up virtual environment with it.

```
pip install numpy
pip install goatools
```

## Test installation

Run:

```
find_enrichment.py -h
```

# SeqKit

SeqKit is a cross-platform and ultrafast toolkit for FASTA/Q file manipulation in Go programming language. SeqKit provides 19 subcommands with independent functions, which could be integrated into command-line pipes to accomplish comprehensive manipulations.

- SeqKit Website
- Publication: SeqKit: A Cross-Platform and Ultrafast Toolkit for FASTA/Q File Manipulation

## Installation

Install via `conda` for Mac OS/Linux

```
conda install -c bioconda seqkit
```

SeqKit provides executable binary files for major operating systems, which can be directly used without any dependencies or pre-configurations.

Download and uncompress the binary file for your platform from download page.

## Usage

SeqKit adopts the structure of "command subcommand", i.e., users access functions of SeqKit from single entrance, `seqkit`, and specify a detailed function with subcommand name, e.g., `seq`.

Read usage and tutorial from the website or command line:

```
seqkit -h
seqkit seq -h
```

Available commands:

```
common      find common sequences of multiple files by id/name/sequence
faidx       create FASTA index file
fq2fa       covert FASTQ to FASTA
fx2tab      covert FASTA/Q to tabular format (with length/GC content/GC skew)
grep        search sequences by pattern(s) of name or sequence motifs
head        print first N FASTA/Q records
locate      locate subsequences/motifs
rename      rename duplicated IDs
replace     replace name/sequence by regular expression
rmdup       remove duplicated sequences by id/name/sequence
sample      sample sequences by number or proportion
seq         transform sequences (revserse, complement, extract ID...)
shuffle     shuffle sequences
sliding     sliding sequences, circular genome supported
sort        sort sequences by id/name/sequence/length
split       split sequences into files by id/seq region/size/parts
stat        simple statistics of FASTA files
subseq      get subsequences by region/gtf/bed, including flanking sequences
tab2fx      covert tabular format to FASTA/Q format
version     print version information and check for update
```

Use `seqkit [command] --help` for more information about a command.

# Sequence Toolkit: seqtk

Seqtk is a fast and lightweight tool for processing sequences in the FASTA or FASTQ format. It seamlessly parses both FASTA and FASTQ files which can also be optionally compressed by gzip.

Webpage and documentation: https://github.com/lh3/seqtk

# Installation

```
brew install seqtk
```

or

```
conda install -y seqtk
```

# Source code install

```
cd ~/src
git clone https://github.com/lh3/seqtk.git
cd seqtk
make

# Link the program to your bin folder
ln -s ~/src/seqtk/seqtk ~/bin/seqtk
```

# Test

`seqtk` operates with subcommands. That means that a typical invocation is:

```
seqtk <command>
```

Running seqtk alone or with a subcommand prints the help:

```
seqtk
```

or

```
seqtk seq
```

# Samtools

Samtools is a software for reading/writing/editing/indexing/viewing SAM/BAM/CRAM format.

- Webpage: http://www.htslib.org/
- SAM documentation: http://www.htslib.org/doc/samtools.html
- SAM Specification: http://samtools.github.io/hts-specs/
- Publication: The Sequence Alignment/Map format and SAMtools

Learn more about the `CRAM` format:

- CRAM goes mainline by Ewan Birney

# Installation

```
brew install samtools
```

or

```
conda install -y samtools
```

# Note

Samtools has changed the command line invocation (for the better). But this means that most of the tutorials on the web indicate an older and obsolete usage.

Use only samtools 1.3 or later.

# Source code installation

```
cd ~/src
curl -OkL https://github.com/samtools/samtools/releases/download/1.3/samtools-1.3.tar.bz2
tar jxvf samtools-1.3.tar.bz2
cd samtools-1.3
make

# Add directory to the path if necessary
echo export `PATH=~/src/samtools-1.3:$PATH` >> ~/.bashrc
source ~/.bashrc
```

Test that the installation succeeded:

```
samtools
```

Samtools has a nicely formatted manual:

```
man ~/src/samtools-1.3/samtools.1
```

# Usage

Samtools is used via subcommands:

```
samtools <command> file
```

Statistics on flags:

```
samtools flagstat data.sam
```

Statistics on the index:

```
samtools idxstats data.bam
```

# This advice may be outdated

We believe the new version of samtools generates random names rather than the same name for its temporary files. We haven't fully confirmed this yet so we'll leave the old advice here.

**Note:** when run in a parallel environment `samtools sort` will need the `-T` parameter set to ensure that temporary files are named differently! Wildly unexpected, mindboggingly confusing (and obviously wrong) results will tacitly be generated otherwise! Trust us on this one :-)

# Picard Tools

A set of Java command line tools for manipulating high-throughput sequencing data (HTS) data and formats.

- Webpage: http://broadinstitute.github.io/picard/

**Mac OSX**:

```
brew update
brew info picard-tools
brew install picard-tools
```

**Source code installation: Mac OSX, Linux, Cygwin/Windows**

Visit https://github.com/broadinstitute/picard/releases/ to identify the latest release.

```
cd ~/src
# Obtain the picard distribution.
curl -OL https://github.com/broadinstitute/picard/releases/download/1.140/picard-tools-1.140.z
ip
unzip picard-tools-1.140.zip
```

Test that the installation succeeded:

```
java -jar ~/src/picard-tools-1.140/picard.jar
```

Create a script that launches picard:

```
echo '#!/bin/bash' > ~/bin/picard
echo 'java -jar ~/src/picard-tools-1.140/picard.jar $@' >> ~/bin/picard

# Make the script executable
chmod +x ~/bin/picard
```

Test that the script works with:

```
picard
```

# TaxonKit

TaxonKit is a cross-platform and efficient NCBI Taxonomy Toolkit in Go programming language.

- TaxonKit Website

## Installation

Install via conda for Mac OS/Linux

```
conda install -c bioconda seqkit
```

TaxonKit provides executable binary files for major operating systems, which can be directly used without any dependencies or pre-configurations.

Download and uncompress the binary file for your platform from download page.

## Usage

TaxonKit adopts the structure of "command subcommand", i.e., users access functions of TaxonKit from single entrance, `taxonkit`, and specify a detailed function with subcommand name, e.g., `list`.

Read usage and tutorial from the website or command line:

```
taxonkit -h
taxonkit list -h
```

Available commands:

```
lineage     query lineage of given taxids from file/stdin
list        list taxon tree of given taxids
reformat    reformat lineage from stdin
version     print version information and check for update
```

Use `taxonkit [command] --help` for more information about a command.

# Bioawk

Bioawk is an extension to Brian Kernighan's awk, adding the support of several common biological data formats, including optionally gzip'ed BED, GFF, SAM, VCF, FASTA/Q and TAB-delimited formats with column names. It also adds a few built-in functions and an command line option to use TAB as the input/output delimiter. When the new functionality is not used, bioawk is intended to behave exactly the same as the original BWK awk.

Webpage: https://github.com/lh3/bioawk

# Install

```
brew install bioawk
```

or

```
conda install -y bioawk
```

**Source install for all platforms**:

```
cd ~/src
git clone https://github.com/lh3/bioawk.git
cd bioawk
make

# Link the program to your bin folder
ln -s ~/src/bioawk/bioawk ~/bin/bioawk
```

A nicely formatted manual can be accessed via:

```
man ~/src/bioawk/awk.1
```

The Bioawk specific extensions are described towards the end of the manual page. Test installation by running:

```
bioawk
```

# Jim Kent's utilities

The page http://hgdownload.cse.ucsc.edu/admin/exe/ contains a number of utility programs developed by Jim Kent.

Some of these tools are the only published used converters of formats such as bigWig, bigBed etc.

# MacOS

```
brew install kent-tools
```

# Manual Installation

The general way to install the tools is to navigate to the directory that contains binaries for your platform (Linux/Mac OSX/) then download the tool.

For example `bedGraphToBigWig` on MacOS X:

```
# Change this for Linux
URL=http://hgdownload.cse.ucsc.edu/admin/exe/macOSX.x86_64

# Get the files
curl $URL/bedGraphToBigWig > ~/bin/bedGraphToBigWig
chmod +x ~/bin/bedGraphToBigWig

curl $URL/faToTwoBit > ~/bin/faToTwoBit
chmod +x ~/bin/faToTwoBit
```

You can also obtain all codes an place them into

```
mkdir -p ~/bin/kent
```

# FastQC quality control visualization

A veritable workhorse of quality control FastQC is probably the most commonly used software of high throughput genomics. It is a java program that should be installed as a command line tool.

Webpage: http://www.bioinformatics.babraham.ac.uk/projects/fastqc/

## Installation

```
brew install fastqc
```

or

```
conda install -y fastqc
```

## Source code installation

This is helpful when one wants to understand what type of files come with `fastqc`

```
cd ~/src
curl -O http://www.bioinformatics.babraham.ac.uk/projects/fastqc/fastqc_v0.11.5.zip
unzip fastqc_v0.11.5.zip

# Link the fastqc executable to the ~/bin folder that
# you have already added to the path.
ln -sf ~/src/FastQC/fastqc ~/bin/fastqc

# Due to what seems a packaging error
# the executable flag on the fastqc program is not set.
# We need to set it ourselves.
chmod +x ~/bin/fastqc
```

Test installation by running:

```
fastqc -h
```

## Tips

Run the `--nogroup` option to turn off binning (do this for short reads only!):

```
fastqc --nogroup data.fq
```

The `FastQC/Configuration` directory contains a number of files of interest that can be customized if needed.

```
ls ~/src/FastQC/Configuration/
```

Produces:

```
adapter_list.txt
contaminant_list.txt
limits.tx
```

You may edit and add your contaminants or adapters then run `fastqc` with the `--adapter` or `--contaminant` options.

# Trimmomatic

Trimmomatic performs a variety of useful trimming tasks for illumina paired-end and single ended data. The software requires java to operate.

- Trimmomatic Website
- Publication: Trimmomatic: A flexible trimmer for Illumina Sequence Data. Bioinformatics (2014).

# Installation

```
brew install trimmomatic
```

or

```
conda install -y trimmomatic
```

# Installation

```
cd ~/src
curl -O http://www.usadellab.org/cms/uploads/supplementary/Trimmomatic/Trimmomatic-0.36.zip
unzip Trimmomatic-0.36.zip

# The program can be invoked via
java -jar ~/src/Trimmomatic-0.36/trimmomatic-0.36.jar

# The ~/src/Trimmomatic-0.36/adapters/ directory contains
# Illumina specific adapter sequences.
ls ~/src/Trimmomatic-0.36/adapters/
```

# How to run trimmomatic

Unfortunately running trimmomatic is as user unfriendly as it gets. To run it we "simply" type:

```
java -jar ~/src/Trimmomatic-0.36/trimmomatic-0.36.jar
```

That gets old very quickly. To simplify the invocation create a script in the ~/bin folder:

```
echo '#!/bin/bash' > ~/bin/trimmomatic
echo 'java -jar ~/src/Trimmomatic-0.36/trimmomatic-0.36.jar $@' >> ~/bin/trimmomatic
chmod +x ~/bin/trimmomatic
```

Test installation by running:

```
trimmomatic
```

# Cutadapt

Cutadapt is a tool to remove different types of unwanted sequence like primers, adapters, polyA-tails from high throughput sequencing reads.

Cutadapt can handle IUPAC wildcard characters in adapter sequence. It supports single-end, paired end and color space data.

- Cutadapt Website
- Publication: Cutadapt removes adapter sequences from high-throughput sequencing reads

# Installation

Cutadapt is automated with conda:

```
conda install -y cutadapt
```

On the Mac OS:

```
pip install --user --upgrade cutadapt
```

# Source code installtation

```
cd ~/src
git clone git@github.com:marcelm/cutadapt.git
cd cutadapt
python setup.py install --user
# Link to bin
ln -s ~/Library/Python/2.7/bin/cutadapt ~/bin/
```

# Test

Test installation by running:

```
cutadapt -h
```

# Example usage:

To trim a 3' adapter `AATGCCTAC` :

```
cutadapt -a AATGCCTAC -o out.fq in.fq
```

To trim 5' end adapter

```
cutadapt -g AATGCCTAC -o out.fq in.fq
```

You can also use it in combination with other commands with pipe.

```
cat in.fq | cutadapt -g AATGCCTAC - > out.fq
```

# BBTools/BBMap Suite

Author: **Hemant Kelkar, PhD**

This file will describe tools included in BBMap suite (https://sourceforge.net/projects/bbmap/).

- BBTools are written in pure Java, can run on any platform (PC, Mac, UNIX), and have no dependencies other than Java being installed (compiled for Java 6 and higher). All tools are efficient and multithreaded.
- BBTools can handle common sequencing file formats such as fastq, fasta, sam, scarf, fasta+qual, compressed or raw, with autodetection of quality encoding and interleaving.
- BBTool suite includes programs to do quality control, trimming, error-correction, assembly and alignment of high thoughput sequencing data.
- BBTools are open source and free for unlimited use.
- BBTools require no installation. All you need to do is following after you download the tar archive from BBTools SF site.

## Installation

```
brew install bbmap
```

or

```
conda install -y bbmap
```

## Installation from source

```
cd ~/src

curl-O <url>

# Extract the files using tar.
tar -xvfz BBMap_(version).tar.gz


stats.sh in=(installation directory)/resources/phix174_ill.ref.fa.gz
```

NOTE: BBMap (and all related tools) shellscripts will try to autodetect memory, but may fail (resulting in the `jvm` failing to start or running out of memory), depending on the system configuration. This can be overridden by adding the -XmxNNg flag to the parameter list of the shellscript (adjusted for your computer's physical memory,don't use more than 80% of total RAM) and it will be passed to java.

# Main Programs included in BBMap suite

Detailed help for all these programs is available inline by running the scripts without any input/options.

**Linux/Mac OS/UNIX**: BBTools can be run using the scripts provided.

**Windows or any other OS that does not support shell scripts** : Run BBTools programs as follows (example for bbmap, replace `NN` with a reasonable number, never use more than 80% of RAM available on your machine).

```
java -XmxNNg -cp /path_to/current align2.BBMap in=reads.fq out=mapped.sam
```

> BBTools programs will accept gzip-compressed, native fastq and are capable of writing output in multiple data formats. These are generally decided by the file extension you choose for output file names (e.g. `.fastq` or `.fq` will write uncompressed fastq files, `.fastq.gz` or `.fq.gz` will write gzip-compressed fastq files). `in=stdin.fq` will accept reads from \, and `out=stdout.fq` will write to \.
>
> As long as `samtools` is available in `$PATH` all BBTools programs can make use of `samtools` to read/write `bam` files.

# Main BBMap suite programs are described in individual sections. Following is a limited list of smaller utility programs that do useful things/produce statistics about sequence data :-)

## stats (stats.sh)

Used for: Generating detailed statistics for reads, assemblies.

Description: Generates basic assembly statistics such as scaffold count, N50, L50, GC content, gap percent, etc. For multiple files, please use statswrapper.sh. Works with fasta and fastq only (gzipped files are acceptable).

```
stats.sh in=<file>

# Let us run this on the mixed-bacterial-sequence data file we have downloaded from the Biosta
rhandbook data site.

stats.sh mixed-bacterial-reads.fq.gz

# The command produces following output

A       C       G       T       N       IUPAC   Other   GC      GC_stdev
0.2716  0.2286  0.2281  0.2716  0.0000  0.0000  0.0000  0.4568  0.1292

Main genome scaffold total:             35000
Main genome contig total:               35000
Main genome scaffold sequence total:    3.500 MB
Main genome contig sequence total:      3.500 MB        0.000% gap
Main genome scaffold N/L50:             35000/100
Main genome contig N/L50:               35000/100
Main genome scaffold N/L90:             35000/100
Main genome contig N/L90:               35000/100
Max scaffold length:                    100
Max contig length:                      100
Number of scaffolds > 50 KB:            0
% main genome in scaffolds > 50 KB:     0.00%


Minimum         Number          Number          Total           Total           Scaffold
Scaffold        of              of              Scaffold        Contig          Contig
Length          Scaffolds       Contigs         Length          Length          Coverage
--------        --------------  --------------  --------------  --------------  --------
    All                 35,000          35,000       3,500,000       3,500,000   100.00%
     50                 35,000          35,000       3,500,000       3,500,000   100.00%
    100                 35,000          35,000       3,500,000       3,500,000   100.00%
```

## pileup (pileup.sh)

Used for: Generating coverage for scaffolds

Description: Calculates per-scaffold coverage information from an unsorted sam or bam file.

```
pileup.sh in=<input> out=<output>
```

## readlength (readlength.sh)

Description: Generates a length histogram of input reads.

```
readlength.sh in=<input file> out=hist.out
```

## kmercountexact (kmercountexact.sh)

Description: Counts the number of unique kmers in a file. Generates a kmer frequency histogram and genome size estimate (in peaks output), and prints a file containing all kmers and their counts. Supports K=1 to infinity, though not all values are allowed.

```
kmercountexact.sh in=<file> khist=<file> peaks=<file> out=<file>

# Let us run the program on our mixed bacterial reads file.

kmercountexact.sh in=mixed-bacterial-reads.fq.gz khist=mixed.hist out=kmer.out
```

> This command will write a list of all unique `kmers` found in the run to the `kmer.out` file. It will also produce statistics about the run.

```
Executing kmer.KmerTableSet [in=mixed.fq.gz, khist=mixed.hist, peaks=mixed.peaks, out=kmer.out]


Initial:
Ways=31, initialSize=128000, prefilter=f, prealloc=f
Memory: max=39156m, free=38543m, used=613m

Estimated kmer capacity:        1521404499
After table allocation:
Memory: max=39156m, free=38134m, used=1022m

After loading:
Memory: max=39156m, free=35887m, used=3269m

Input:                          35000 reads             3500000 bases.

For K=31
Unique Kmers:                   2313250
Load Time:                      0.346 seconds.
Write Time:                     1.456 seconds.
```

## bbnorm (bbnorm.sh)

Description: Normalizes read depth based on kmer counts. Can also error-correct, bin reads by kmer depth, and generate a kmer depth histogram.

```
bbnorm.sh in=<input> out=<reads to keep> outt=<reads to toss> hist=<histogram output>
```

## seal (seal.sh)

Description: Performs high-speed alignment-free sequence quantification, by counting the number of long kmers that match between a read and a set of reference sequences. Designed for RNA-seq with alternative splicing.

```
seal.sh in=<input file> ref=<file,file,file...> rpkm=<file>
```

## bbmask (bbmask.sh)

Used for: Masking sequences

Description: Masks sequences of low-complexity, or containing repeat kmers, or covered by mapped reads. By default this program will mask using entropy with a window=80 and entropy=0.75

Input may be stdin or a fasta or fastq file, raw or gzipped. sam is optional, but may be a comma-delimited list of sam files to mask.

```
bbmask.sh in=<file> out=<file> sam=<file,file,...file>
```

## removesmartbell (removesmartbell.sh)

Used for: Removal fo SMRTbell adapters that may be present in PacBio data.

```
removesmartbell.sh in=<input> out=<output> split=t
```

# Basic Local Alignment Search Tool (BLAST)

BLAST stands for **Basic Local Alignment Search Tool** and is an algorithm for searching a sequence (called a query) against very large sequence databases (called a target).

- Web interface for BLAST
- Downloadable blast programs.

## Documentation

- Blast Handbook

> Note: The blast version may change in time. The name of the file indicates the version of the program. Investigate the version of the latest release and adapt the names of the programs accordingly. These instructions will install version `BLAST+ 2.5.0`

## Mac OSX

```
brew info blast
brew install blast
```

Alternatively one may install the binaries

```
cd ~/src
curl -O  ftp://ftp.ncbi.nlm.nih.gov/blast/executables/blast+/LATEST/ncbi-blast-2.5.0+-x64-maco
sx.tar.gz
tar zxvf ncbi-blast-2.5.0+-x64-macosx.tar.gz
export PATH=~/src/ncbi-blast-2.5.0+/bin:$PATH
```

## Linux

```
cd ~/src
curl -O  ftp://ftp.ncbi.nlm.nih.gov/blast/executables/blast+/LATEST/ncbi-blast-2.5.0+-x64-linu
x.tar.gz
tar zxvf ncbi-blast-2.5.0+-x64-linux.tar.gz
export PATH=~/src/ncbi-blast-2.5.0+/bin:$PATH

# See the programs that are available
ls ~/src/ncbi-blast-2.5.0+/bin/
```

## Windows

BLAST can be installed on Windows system.

- Visit ftp://ftp.ncbi.nlm.nih.gov/blast/executables/blast+/LATEST/
- Download the file `ncbi-blast-2.5.0+-win64.exe`
- Double-click and install

# Test

Test installation by running:

```
which blastn
blastn -version
```

# LASTZ aligner

The LASTZ aligner is developed by Bob Harris in collaboration with Webb Miller.

It is a program for pairwise aligning of DNA sequences. Originally designed to handle sequences the size of human chromosomes and from different species, it is also useful for sequences produced by NGS sequencing technologies such as Roche 454.

- LASTZ Aligner Webpage
- LASTZ Aligner Documentation

All platforms (installs the latest version):

```
 cd ~/src
curl -O http://www.bx.psu.edu/~rsharris/lastz/newer/lastz-1.03.73.tar.gz
tar xzvf lastz-1.03.73.tar.gz
cd lastz-distrib-1.03.73/

# Build the tool. (see notes below).
make

# Link the executable to the ~/bin folder
 ln -sf ~/src/lastz-distrib-1.03.73/src/lastz ~/bin
```

The compilation above may fail on the Mac due to overly stringent warnings (treats all warnings as errors). If that happens edit the file `~/src/lastz-distrib-1.03.73/src/Makefile` and remove the `-Werror` flag for the `defineAll` variable. The program should compile successfully.

# MacOSX

The `lastz` version that installs via homebrew is an older version:

```
brew update
brew info lastz
brew install lastz
```

# Test

```
lastz
```

# LAST aligner

The LAST aligner is developed by Martin Firth.

- LAST Aligner Webpage
- LAST Aligner Documentation
- Publications describing LAST

Installing on all platforms:

```
 cd ~/src
curl -O http://last.cbrc.jp/last-719.zip
unzip last-719.zip
cd last-719
make
```

Link the executables into the `~/bin` folder

```
ln -s ~/src/last-719/src/lastdb ~/bin
ln -s ~/src/last-719/src/lastal ~/bin
ln -s ~/src/last-719/src/last-split ~/bin
```

There are a number of utility scripts that could useful. For example

```
ln -s ~/src/last-719/scripts/maf-sort ~/bin/
```

And many others in the `~/src/last-719/scripts/` folder. The **Mac OSX** version that install with brew is an older version:

```
brew update
brew info last
brew install last
```

Test that the installation worked:

```
lastal -h
```

Using the LAST aligner (data included in the download):

```
# A shortcut to the data directory
DATA=~/src/last-719/examples/

# Make a LAST database with the human sequence:
lastdb -c $DATA/humanMito $DATA/humanMito.fa

# Align the mouse mitochondrial genome to the human mitochondrial genome.
lastal -e25 $DATA/humanMito $DATA/mouseMito.fa > output.maf
```

The output is in the MAF (multiple Alignment Format) not to be confused with MAF (Mutation Alignment Format) ;-)

# The BWA Aligner

BWA stands for **Burrows-Wheeler Aligner** and is a software package for mapping low-divergent sequences against a large reference genome, such as the human genome. It consists of three algorithms:

- BWA-backtrack
- BWA-SW
- BWA-MEM.

**BWA-MEM** (Maximally Exact Matches), which is the latest, is generally recommended for high-quality queries as it is faster and more accurate. BWA-MEM also has better performance than BWA-backtrack for 70-100bp Illumina reads.

BWA is one of the most used high throughput aligners in the world yet it is "unpublished" and by the classic standards of science is "non-peer reviewed". It is an interesting example with an enlightening back story of how the scientific peer review model can fail. The "unpublished" writeup on bwa-mem can be read on arxiv: Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM

# Installation:

```
brew install bwa
```

or

```
conda install -y bwa
```

# All other platforms:

```
cd ~/src
curl -OL http://sourceforge.net/projects/bio-bwa/files/bwa-0.7.15.tar.bz2
tar jxvf bwa-0.7.15.tar.bz2
cd bwa-0.7.15
make
export PATH=~/src/bwa-0.7.15:$PATH
```

# Test installation:

Run:

```
bwa
```

BWA has a nicely formatted manual:

```
man ~/src/bwa-0.7.12/bwa.1
```

# Example:

```
# get reference
ACC=JQ340389
mkdir -p ./refs/
efetch -db=nuccore -format=fasta -id=$ACC >./refs/$ACC.fa

#simulate reads from reference
dwgsim -N 10000 ./refs/$ACC.fa $ACC
READ1=$ACC.bwa.read1.fastq
READ2=$ACC.bwa.read2.fastq

#index the reference
bwa index ./refs/$ACC.fa

#run bwa mem
bwa mem ./refs/$ACC.fa $READ1 $READ2 | samtools view -h -b - | samtools sort >$ACC.bam
samtools index $ACC.bam
```

# Bowtie Sequence Aligner

Bowtie is an ultrafast, memory-efficient short read aligner. It aligns short DNA sequences (reads) to the human genome at a rate of over 25 million 35-bp reads per hour. Bowtie indexes the genome with a Burrows-Wheeler index to keep its memory footprint small: typically about 2.2 GB for the human genome (2.9 GB for paired-end).

It has two versions:

- Bowtie 1 - first version that is trades accuracy for speed. The aligner cannot handle indels (insertions and deletions) hence it is best for a subset of problems where that is not a problem. For example Bowtie 1 is the tool of choice for mapping microRNAs.
- Bowtie 2 - slower than Bowtie 1 but implements a more accurate and rigorous alignment procedure.

Publications:

- Bowtie1: Ultrafast and memory-efficient alignment of short DNA sequences to the human genome, Genome Biology, 2009
- Bowtie2: Fast gapped-read alignment with Bowtie 2. Nature Methods. 2012

To avoid confusion it is best if you install only one version: `bowtie2`. There are substantial differences in the way we invoke and run these commands.

If you need `bowtie1` installed you will need to slightly adapt the installation commands.

In addition `bowtie2` comes as a suite of programs. It is best to add the installation folder to the execution path rather than linking the programs.

# Mac OSX

```
brew info bowtie2
brew install bowtie2
```

# Mac OSX alternative

```
cd ~/src
curl -OL http://sourceforge.net/projects/bowtie-bio/files/bowtie2/2.2.9/bowtie2-2.2.9-macos-x8
6_64.zip
unzip bowtie2-2.2.9-macos-x86_64.zip
export PATH=~/src/bowtie2-2.2.9:$PATH
```

# Linux

```
cd ~/src
curl -OL http://sourceforge.net/projects/bowtie-bio/files/bowtie2/2.2.9/bowtie2-2.2.9-linux-x8
6_64.zip
unzip bowtie2-2.2.9-linux-x86_64.zip
export PATH=~/src/bowtie2-2.2.9:$PATH
```

# Test

Run:

```
bowtie2
```

# Example

```
# get reference
ACC=JQ340389
mkdir -p ./refs/
efetch -db=nuccore -format=fasta -id=$ACC >./refs/$ACC.fa

#simulate reads from reference
dwgsim -N 10000 ./refs/$ACC.fa $ACC

READ1=$ACC.bwa.read1.fastq
READ2=$ACC.bwa.read2.fastq

#index the reference
bowtie2-build ./refs/$ACC.fa ./refs/$ACC

#run bowtie2
bowtie2 -x ./refs/$ACC -1 $READ1 -2 $READ2 | samtools view -h -b - | samtools sort >$ACC.bam
samtools index $ACC.bam
```

# BBMAP quality control

## How can I scan and remove contaminants from my sequence?

By using `bbduk` or `bbduk2` from BBMap suite (see NOTE below).

Used for: Removal of adapter contamination, other trimming (e.g. quality-, length-based). Paired-end (PE) data should always be processed as a pair (R1/R2 reads).

Description: Compares reads to the kmers in a reference dataset, optionally allowing an edit distance. Splits the reads into two outputs - those that match the reference, and those that don't. Can also trim (remove) the matching parts of the reads rather than binning the reads.

```
bbduk.sh in=<input file> out=<output file> ref=<contaminant files>
```

Following section assumes that you have downloaded the test data we are going to use. If not, you can grab a copy by following first three steps from this page http://read.biostarhandbook.com/sam/sam-analyze.html

```
bbduk.sh in1=SRR1972739_1.fastq in2=SRR1972739_2.fastq out1=SRR1972739_1_trim.fastq out2=SRR19
72739_2_trim.fastq ktrim=r k=20 hdist=1 tpe tbo ref=/nas02/apps/bbmap-36.49/bbmap/resources/ad
apters.fa
```

`bbduk` takes two input files (for paired-end data) and produces two output files that contains adapter scanned/trimmed data ( `in=, out=` ). BBMap includes a "resources" folder that contains most commonly used adapter sequences in a file ( `ref=/path_to/bbmap/resources/adapters.fa` ). When kmer right-trimming, trim both reads to the minimum length of either ( `tpe` ). Trim adapters based on where paired reads overlap ( `tbo` ). Use of `tbo` and `tpe` is recommended when trimming paired-end data. Kmer length of `20` is used for finding contaminants along with a "hamming distance" of `1` .

Post-run, `bbduk` produces detailed statistics for the run. They are generally be written to \ but can be easily captured to a log file (example below).

```
BBDuk version 36.49
Initial:
Memory: max=19601m, free=19192m, used=409m

Added 140923 kmers; time:       0.248 seconds.
Memory: max=19601m, free=18271m, used=1330m

Input is being processed as paired
Started output streams: 0.032 seconds.
Processing time:                0.355 seconds.

Input:                          20000 reads             2020000 bases.
KTrimmed:                       248 reads (1.24%)       14934 bases (0.74%)
Trimmed by overlap:             210 reads (1.05%)       1794 bases (0.09%)
Total Removed:                  100 reads (0.50%)       16728 bases (0.83%)
Result:                         19900 reads (99.50%)    2003272 bases (99.17%)

Time:                           0.642 seconds.
Reads Processed:       20000    31.16k reads/sec
Bases Processed:       2020k    3.15m bases/sec
```

Alternative program: bbduk2.sh

BBDuk2 is like BBDuk but can kfilter, kmask, and ktrim in a single pass. It does not replace BBDuk, and is only provided to allow maximally efficient pipeline integration when multiple steps will be performed. The syntax is slightly different.

Description: Compares reads to the kmers in a reference dataset, optionally allowing an edit distance. Splits the reads into two outputs - those that match the reference, and those that don't. Can also trim (remove) the matching parts of the reads rather than binning the reads.

```
bbduk2.sh in=<input file> out=<output file> fref=<contaminant files>

# This would be an example of an actual run using bbduk2

bbduk2.sh in1=SRR1972739_1.fastq in2=SRR1972739_2.fastq out1=SRR1972739_1_duk2_trim.fastq out2=
SRR1972739_2_duk2_trim.fastq k=20 hdist=1 tpe tbo fref=/nas02/apps/bbmap-36.49/bbmap/resources
/adapters.fa
```

See the text under `bbduk.sh` command example above for explanation of parameters.

# Notes about `bbduk` and `bbduk2`

`bbduk` can operate in one of 4 kmer-matching modes: Right-trimming (ktrim=r), left-trimming (ktrim=l), masking (ktrim=n), and filtering (default). But it can only do one at a time because all kmers are stored in a single table. It can still do non-kmer-based operations such as quality trimming at the same time.

`bbduk2` can do all 4 kmer operations at once and is designed for integration into automated pipelines where you do contaminant removal and adapter-trimming in a single pass to minimize filesystem I/O. Both have identical capabilities and functionality otherwise, but the syntax is different.

# How can I find reads with a specific sequence?

Take a look at this example which uses `bbduk.sh`.

```
bbduk.sh -Xmx1g in=reads.fq outm=matched.fq outu=unmatched.fq restrictleft=25 k=25 literal=AAA
AACCCCCTTTTTGGGGGAAAAA
```

In this case, all reads starting with `AAAAACCCCCTTTTTGGGGGAAAAA` will end up in "matched.fq" and all other reads will end up in `unmatched.fq`. Specifically, the command means "look for 25-mers in the leftmost 25 bp of the read", which will require an exact prefix match, though you can relax that if you want.

So you could bin all the reads with your known sequence, then look at the remaining reads to see what they have in common. You can do the same thing with the tail of the read using `restrictright` instead, though you can't use both restrictions at the same time.

```
bbduk.sh in=reads.fq outm=matched.fq literal=NNNNNNCCCCGGGGGTTTTTAAAAA k=25 copyundefined
```

With the `copyundefined` flag, a copy of each reference sequence will be made representing every valid combination of defined letter. So instead of increasing memory or time use by 6^75, it only increases them by 4^6 or 4096 which is completely reasonable, but it only allows substitutions at predefined locations. You can use the `copyundefined`, `hdist` and `qhdist` flags together for a lot of flexibility - for example, `hdist=2 qhdist=1` and 3 Ns in the reference would allow a hamming distance of 6 with much lower resource requirements than `hdist=6`. Just be sure to give BBDuk as much memory as possible.

# How can I identify certain reads that contain a specific sequence?

```
bbduk.sh in=reads.fq out=unmatched.fq outm=matched.fq literal=ACGTACGTACGTACGTAC k=18 mm=f hdi
st=2
```

Make sure "k" is set to the exact length of the sequence. `hdist` controls the number of substitutions allowed. `outm` gets the reads that match. By default this also looks for the reverse-complement; you can disable that with `rcomp=f`.

# How to extract reads that share *k-mers* with your sequences?

```
bbduk.sh in=a.fa ref=b.fa out=c.fa mkf=1 mm=f k=31
```

This command will print to `c.fa` all the sequences in `a.fa` that share 100% of their 31-mers with sequences in `b.fa`.

# BBMap Sequence Aligner

BBTools/BBMap suite (https://sourceforge.net/projects/bbmap/) is a large collection of programs dedicated to various tasks bioinformaticians analyzing high-throughput sequencing data commonly encounter.

The tools are written in Java, can run on any platform, and have no dependencies other than Java being installed (compiled for Java 7 and higher).

## Installation

Installation is identical on all platforms:

```
# Example below refers to a specific (and probably old version of BBMap by the time you read t
his. Always get the current version available from sourceforge.

cd ~/src
curl -OL http://sourceforge.net/projects/bbmap/files/BBMap_36.32.tar.gz
tar xzvf BBMap_36.32.tar.gz

# Wiew all the programs that were added.
ls ~/src/bbmap

# Add the BBMap suite to the execution path
export PATH=~/src/bbmap:$PATH
```

Test installation by running:

```
bbmap.sh
```

## How do I align high thoughput sequence data with bbmap?

Following section assumes that you have downloaded the test data we are going to use. If not, you can grab a copy by following first three steps from this page http://read.biostarhandbook.com/sam/sam-analyze.html

```
# Index reference for use with BBMap

bbmap.sh ref=$REF
```

This process creates indexes used by BBMap for alignment (these are generally program specific and will need to be created for any new aligner you are planning to use).

Index files are created in a directory called **ref**. BBMap uses its own file organization for the indexes, so do not touch any files under **ref** folder once the indexes are created.

```
# Align with bbmap.sh

bbmap.sh in1=$R1 in2=$R2 path=./ out=stdout statsfile=bbmap.stats | samtools sort > bbmap.bam
samtools index bbmap.bam
```

bbmap generates excellent statistics, which we captured in **bbmap.stats** (see command line above). If *statsfile*= option is not provided in bbmap.sh command line then these statistics would be written to \.

```
cat bbmap.stats
```

allows us to look at the statistics for this alignment :

```
Reads Used:             20000   (2020000 bases)

Mapping:                2.947 seconds.
Reads/sec:              6786.13
kBases/sec:             685.40


Pairing data:           pct reads       num reads       pct bases          num bases

mated pairs:             57.8200%            5782        57.8200%            1167964
bad pairs:               10.7800%            1078        10.7800%             217756
insert size avg:          237.81


Read 1 data:            pct reads       num reads       pct bases          num bases

mapped:                  70.8700%            7087        70.8700%             715787
unambiguous:             70.8700%            7087        70.8700%             715787
ambiguous:                0.0000%               0         0.0000%                  0
low-Q discards:           0.0000%               0         0.0000%                  0

perfect best site:        5.5700%             557         5.5700%              56257
semiperfect site:         5.5700%             557         5.5700%              56257
rescued:                  2.1400%             214

Match Rate:                   NA              NA        95.3153%             685534
Error Rate:              92.0841%            6526         4.6782%              33647
Sub Rate:                92.0135%            6521         3.9855%              28665
Del Rate:                 2.8080%             199         0.4784%               3441
Ins Rate:                 4.3319%             307         0.2143%               1541
N Rate:                   0.2258%              16         0.0065%                 47


Read 2 data:            pct reads       num reads       pct bases          num bases

mapped:                  70.4500%            7045        70.4500%             711545
unambiguous:             70.4500%            7045        70.4500%             711545
ambiguous:                0.0000%               0         0.0000%                  0
low-Q discards:           0.7800%              78         0.7800%               7878

perfect best site:        5.7500%             575         5.7500%              58075
semiperfect site:         5.7500%             575         5.7500%              58075
rescued:                  2.7800%             278

Match Rate:                   NA              NA        95.3431%             681052
Error Rate:              91.8382%            6470         4.6460%              33187
Sub Rate:                91.7956%            6467         4.0283%              28775
Del Rate:                 2.5692%             181         0.3881%               2772
Ins Rate:                 4.5280%             319         0.2296%               1640
N Rate:                   0.1136%               8         0.0109%                 78
```

# How do I align long reads (> 300 bp - kilobases) with bbmap?

BBMap has a dedicated variant that is used for aligning reads from PacBio (or Oxford Nanopore) where the reads lengths are much longer than standard Illumina reads. Internally BBMap has a read length cap of 6 kb but it gets around this limit by breaking up the reads (which are longer than 6 kb into chunks 6kb or less).

We can map simulated PacBio reads that we generated in the {#randomreads} section as follows:

```
mapPacBio.sh threads=2 -Xmx20g k=7 in=NC_010468_synth_pacbio.fq.gz ref=NC_010468_new.fa maxlen=
1000 minlen=200 idtag ow int=f qin=33 out=mapped1.sam minratio=0.15 ignorequality slow ordered
 maxindel1=40 maxindel2=400 overwrite=t

# The "maxlen" flag shreds them to a max length of 1000; you can set that up to 6000. But @Bri
an has found 1000 gave a higher mapping rate
```

Note: Mapping long reads takes significantly longer than regular reads so allow for some time for this example data to run.

# What should I do if I want to align PE and SE reads together with BBMap?

BBMap itself can only run single-ended or paired-ended in a single run, but it has a wrapper that can accomplish mapping mixed mapping like this

```
$ bbwrap.sh in1=read1.fq,singletons.fq in2=read2.fq,null out=mapped.sam append
```

This will write all the reads to the same output file but only print the headers once.

# How do I align small reads requiring exact matches (e.g. miRNA) with bbmap?

When mapping small RNA's with BBMap use the following flags to report only perfect matches ( `ambig=all vslow perfectmode maxsites=1000` ). Mapping should be very fast in this mode (despite the `vslow` flag). `vslow` mainly removes masking of low-complexity repetitive kmers, which is not usually a problem but can be with extremely short sequences like microRNAs.

# What options should I use for BBMap alignments to call variants with variant callers?

If you are planning to call variants using files aligned with BBMap then be sure to include the following flags in your `bbmap.sh` command line `mdtag trd sam=1.3` .

# BBMap Read Merger

Used for: Merging paired-end (PE) reads where they are expected to overlap (e.g. 16S amplicon sequencing).

Description: Merges paired reads into single reads by overlap detection.With sufficient coverage, can also merge nonoverlapping reads by kmer extension. Kmer modes requires much more memory, and should be used with the bbmerge-auto.sh script.

```
Usage for interleaved files:    bbmerge.sh in=<reads> out=<merged reads> outu=<unmerged reads>
Usage for paired files:         bbmerge.sh in1=<read1> in2=<read2> out=<merged reads> outu1=<u
nmerged1> outu2=<unmerged2>
```

We are going to use the synthetic dataset that we generated using `randomreads.sh` to demonstrate how `bbmerge.sh` works [Put a links for {randomreads} page here]. We could have used the interleaved reads file but for the example below we are using R1/R2 reads in separate files.

Note: Since this is a synthetic dataset that was generated to contain most reads that overlap. With real life datasets the percentage of reads that can be merged will vary according to the insert sizes of your libraries and length of sequencing reads.

```
bbmerge.sh in1=NC_010468_synth_R1.fq.gz in2=NC_010468_synth_R2.fq.gz out=NC_010468_synth_merge
d.fq.gz outu1=NC_010468_synth_R1_unmerged.fq.gz outu2=NC_010468_synth_R2_unmerged.fq.gz
```

This should produce log output similar to this

```
Writing mergable reads merged.
Started output threads.
Total time: 96.201 seconds.

Pairs:                  1000000
Joined:                 997954       99.795%
Ambiguous:              1450           0.145%
No Solution:            596            0.060%
Too Short:              0           0.000%

Avg Insert:             249.7
Standard Deviation:     61.4
Mode:                   251

Insert range:           100 - 400
90th percentile:        333
75th percentile:        294
50th percentile:        250
25th percentile:        206
10th percentile:        167
```

# Alignment to multiple genomes/decontaminating datasets

## How can I bin/split reads in my samples using multiple references?

This question is often asked on Biostars. `bbsplit` is the perfect tool to handle this task. It is very flexible and can work with multiple sequences/folders of data.

Used for: Simultaneous alignment to multiple genomes. This feature is useful for binning of reads (which can be used for decontamination) into "genome" specific pools.

Description: Maps reads to multiple references simultaneously. Outputs reads to a file for the reference they best match, with multiple options for dealing with ambiguous mappings.

```
To index:     bbsplit.sh build=<1> ref_x=<reference fasta> ref_y=<another reference fasta>
To map:       bbsplit.sh build=<1> in=<reads> out_x=<output file> out_y=<another output file>
```

To be concise, and do everything in one command:

```
bbsplit.sh ref=x.fa,y.fa in=reads.fq basename=o%.fq

# this is equivalent to

bbsplit.sh build=1 in=reads.fq ref_x=x.fa ref_y=y.fa out_x=ox.fq out_y=oy.fq
```

By default paired reads will yield interleaved output, but you can use the # symbol to produce twin output files. For example, `basename=o%_#.fq` will produce `ox_1.fq, ox_2.fq, oy_1.fq, and oy_2.fq`.

> Added feature: One can specify a directory for the "ref=" argument. If anything in the list is a directory, it will use all fasta files in that directory. The files need a fasta extension, like .fa or .fasta, but can be compressed with an additional .gz after that. Reason this is useful to use with BBSplit is to have it split input into one output file per reference file.

NOTE 1: By default BBSplit uses fairly strict mapping parameters; you can get the same sensitivity as BBMap by adding the flags `minid=0.76 maxindel=16k minhits=1`. With those parameters it is extremely sensitive.

NOTE 2: BBSplit has different ambiguity settings for dealing with reads that map to multiple genomes. In any case, if the alignment score is higher to one genome than another, it will be associated with that genome only (this considers the combined scores of read pairs - pairs are always kept together). But when a read or pair has two identically-scoring mapping locations, on different genomes, the behavior is controlled by the `ambig2` flag - `ambig2=toss` will discard the read, `all` will send it to all output files, and `split` will send it to a separate file for ambiguously-mapped reads (one per genome to which it maps).

NOTE 3: Zero-count lines are suppressed by default, but they should be printed if you include the flag `nzo=f` (nonzeroonly=false).

NOTE 4: BBSplit needs multiple reference files as input; one per organism, or one for target and another for everything else. It only outputs one file per reference file.

Another BBMap tool `seal.sh`, on the other hand, which is similar, can use a single concatenated file, as it (by default) will output one file per reference sequence within a concatenated set of references.

We are now going to go through a simple example for using `bbsplit` below.

```
# We are going to get fasta formatted sequence for three bacterial genomes from NCBI.

efetch -db nucleotide -id NC_003454 -format fasta > NC_003454.fa
efetch -db nucleotide -id NC_010468 -format fasta > NC_010468.fa
efetch -db nucleotide -id NC_013520 -format fasta > NC_013520.fa

# We will get one test data file from Biostarhandbook's data site

wget http://data.biostarhandbook.com/reads/mixed-bacterial-reads.fq.gz

# Even though we are referring to bacterial reads here the solution is generally applicable fo
r any mixed data.
# We can use any combination of one or more genomes to bin reads to every genome included in t
he run.

bbsplit.sh in=mixed-bacterial-reads.fq.gz ref=NC_010468.fa,NC_013520.fa,NC_003454.fa basename=
out_%_#.fq.gz outu=rest.fq.gz minratio=0.76 minhits=1
```

bbsplit should produce the following files after the runs.

```
ls -sh1tr

2.4M rest.fq.gz
608K out_NC_013520_1.fq.gz
608K out_NC_003454_1.fq.gz
608K out_NC_010468_1.fq.gz
```

Note: `mixed-bacterial-reads.fq.gz` file has now been split into three files which contain the reads we are interested in. Remaining reads are put in `rest.fq.gz`

# "Re-pair" Paired-end illumina sequence files

Paired-end (PE) sequenceing is a very useful tool that provides spatial information about fragments being sequenced. Default output of PE puts the read pair (designated as R1 and R2) in two separate files. When you scan these files for presence of adapter sequences and other contaminants it is advisable to process PE data files together. PE aware trimming/scanning programs keep the order of reads in sync across the two read files. The order of reads inside the file is important because most NGS aligners expect the reads to be in identical order in both files (and may not check for that).

The order of reads in PE may be disturbed if the files are incorrectly trimmed or some how become corrupt. In best of circumstances this will generate an error when you use such files as an input for a NGS aligner. (Note: In worst case scenarion, aligners may process the data as is. This may result in a high number of discordant alignments).

It is possible to "re-sync" PE files (that may have gone out of sync in terms of read order) using `repair.sh` from BBMap suite. Here is an illustrated example.

Note: When using real (i.e. large data files) you will need to add `-XmxNNg` parameter to the commands below to assign appropriate amount of RAM.

We are going to create a mismatched PE read file for this example by using `reformat.sh` from BBMap. We will be using the test data set that was obtained in this section: https://read.biostarhandbook.com/tools/bbmap/bbmap-aligner.html. Note that we are using a "properly trimmed" file as an input to deliberately create a broken pair.

```
reformat.sh in=SRR1972739_2_trim.fastq samplerate=0.9 out=SRR1972739_2_broken.fastq

# This creates a "broken" R2 reads file since we are sampling only ~90% of reads from the orig
inal.
You should see log output similar to this

Input is being processed as unpaired
Input:                          9950 reads              1001636 bases
Processed:                      8973 reads              903164 bases
Output:                         8973 reads (90.18%)     903164 bases (90.17%)

Time:                           0.226 seconds.
Reads Processed:        8973    39.63k reads/sec
Bases Processed:        903k    3.99m bases/sec
```

If we use this file as an input for a `bbmap.sh` alignment (refer to the bbmap alignment link above for detailed exaplanation of how to use bbmap for alignments).

```
bbmap.sh in1=SRR1972739_1_trim.fastq in2=SRR1972739_2_broken.fastq path=./ out=stdout statsfile
=bbmap_broken.stats | samtools sort > bbmap_broken.bam

# bbmap recognizes that the reads are not in sync and generates the following log.
# I have highlighted the important line by separating it from other text.

Retaining first best site only for ambiguous mappings.
Set genome to 1

Loaded Reference:       0.081 seconds.
Loading index for chunk 1-1, build 1
Generated Index:        1.150 seconds.
Analyzed Index:         2.404 seconds.
Unspecified format for output stdout; defaulting to sam.
Started output stream:  0.027 seconds.
Cleared Memory:         0.141 seconds.
Processing reads in paired-ended mode.
Started read stream.
Started 8 mapping threads.
Exception in thread "Thread-11" java.lang.AssertionError:

There appear to be different numbers of reads in the paired input files.

The pairing may have been corrupted by an upstream process.  It may be fixable by running repa
ir.sh.
        at stream.ConcurrentGenericReadInputStream.pair(ConcurrentGenericReadInputStream.java:
480)
        at stream.ConcurrentGenericReadInputStream.readLists(ConcurrentGenericReadInputStream.
java:345)
        at stream.ConcurrentGenericReadInputStream.run(ConcurrentGenericReadInputStream.java:1
89)
        at java.lang.Thread.run(Thread.java:745)
```

We will now use `repair.sh` to fix the 'broken' read pairing and re-sync the PE reads to generate "fixed" read files. Output from repair.sh is interleaved PE reads (read1_R1, read1_R2, read2_R1, read2_R2 etc.). "Broken" single reads will be captured in `SRR1972739_broken_reads.fastq` file.

```
repair.sh in1=SRR1972739_1_trim.fastq in2=SRR1972739_2_broken.fastq out=SRR1972739_fixed.fastq
 outsingle=SRR1972739_broken_reads.fastq

# We get the following log output

Set INTERLEAVED to false
Started output stream.

Input:                          18923 reads             1904800 bases.
Result:                         18923 reads (100.00%)   1904800 bases (100.00%)
Pairs:                          17946 reads (94.84%)    1806328 bases (94.83%)
Singletons:                     977 reads (5.16%)       98472 bases (5.17%)

Time:                           0.407 seconds.
Reads Processed:       18923    46.46k reads/sec
Bases Processed:       1904k    4.68m bases/sec
```

We can now split the fixed reads into individual R1/R2 read files by using `reformat.sh` .

```
reformat.sh in=SRR1972739_fixed.fastq out1=SRR1972739_1_fixed.fastq out2=SRR1972739_2_fixed.fa
stq addcolon

Set INTERLEAVED to true
Input is being processed as paired
Input:                          17946 reads             1806328 bases
Output:                         17946 reads (100.00%)   1806328 bases (100.00%)

Time:                           0.464 seconds.
Reads Processed:       17946    38.68k reads/sec
Bases Processed:       1806k    3.89m bases/sec
```

For brevity these two steps can be combined into a single process by doing repair and reformatting in one step

```
repair.sh in1=SRR1972739_1_trim.fastq in2=SRR1972739_2_broken.fastq out=stdout.fq outsingle=SR
R1972739_broken_reads.fastq | reformat.sh in=stdin.fq out1=SRR1972739_1_fixed.fastq out2=SRR19
72739_2_fixed.fastq interleaved addcolon
```

This is a very handy utility to deal with out-of-sync PE reads.

# BBMap suite - Simulation of data

Used for: Generating synthetic high-throughput datasets.

Description: Generates random synthetic reads from a reference genome. Read names indicate their genomic origin. Allows precise customization of things like insert size and synthetic mutation type, sizes, and rates. Both Illumina and PacBio type reads can be generated using randomreads.

Note: You can specify paired reads, an insert size distribution, read lengths (or length ranges), and so forth. Randomreads is specifically designed to give excellent control over mutations. You can specify the number of snps, insertions, deletions, and Ns per read, either exactly or probabilistically; the lengths of these events is individually customizable, the quality values can alternately be set to allow errors to be generated on the basis of quality. All of the reads are annotated with their genomic origin, so you will know the correct answer when mapping.

Bear in mind that 50% of the reads are going to be generated from the plus strand and 50% from the minus strand. So, either a read will match the reference perfectly, OR its reverse-complement will match perfectly.

```
Usage:   randomreads.sh ref=<file> out=<file> length=<number> reads=<number>
```

1. Simulating a high though-put sequencing illumina dataset.

We are going to download accession number NC_010468 in Fasta format (in case you have not done this).

```
efetch -db=nuccore -format=fasta -id=NC_010468 > NC_010468.fa
```

We are going to simplify the fasta header in `NC_010468.fa` by the following command so the new header reads `>NC_010468_E_coli`

```
sed '1 s/^.*$/\>NC_010468_E_coli/' NC_010468.fa > NC_010468_new.fa
```

We are going to use this reference sequence file to generate a synthetic illumina paired-end dataset (2 x 300 bp) that will have overlapping reads.

```
randomreads.sh ref=NC_010468_new.fa out=NC_010468_synth.fq.gz length=300 paired=t mininsert=10
0 maxinsert=400 reads=1000000
```

When `randomreads.sh` run completes it should produce log output similar to text below.

```
Set genScaffoldInfo=true
Deleting chr1.chrom.gz
Writing chunk 1
Waiting for writing to finish.
Finished.
snpRate=0.0, max=0, unique=true
insRate=0.0, max=0, len=(0-0)
delRate=0.0, max=0, len=(0-0)
subRate=0.0, max=0, len=(0-0)
nRate  =0.0, max=0, len=(0-0)
genome=1
PERFECT_READ_RATIO=0.0
ADD_ERRORS_FROM_QUALITY=true
REPLACE_NOREF=false
paired=true
read length=300
reads=1000000
insert size=100-400
Wrote NC_010468_synth
Time:   48.177 seconds.

# By default the reads generated are in interleaved format. They can be separated into individ
ual read files by `reformat.sh`

reformat.sh in=NC_010468_synth.fq.gz out1=NC_010468_synth_R1.fq.gz out2=NC_010468_synth_R2.fq.
gz addcolon

# Following output should be generated in the log file.

Set INTERLEAVED to true
Input is being processed as paired
Input:                          2000000 reads           600000000 bases
Output:                         2000000 reads (100.00%)         600000000 bases (100.00%)

Time:                           39.083 seconds.
Reads Processed:       2000k    51.17k reads/sec
Bases Processed:        600m    15.35m bases/sec
```

1. Generating a Pacific biosciences format dataset with randomreads (which uses PacBio error model).

We are simulating a PacBio dataset containing 10,000 reads with a min read length of 1kb and a max read length of 5kb.

```
randomreads.sh ref=NC_010468_new.fa out=NC_010468_synth_pacbio.fq.gz minlength=1000 maxlength=
5000 reads=10000 pacbio=t

# This should generate the following log output

Set genScaffoldInfo=true
Deleting chr1.chrom.gz
Writing chunk 1
Waiting for writing to finish.
Finished.
snpRate=0.0, max=0, unique=true
insRate=0.0, max=0, len=(0-0)
delRate=0.0, max=0, len=(0-0)
subRate=0.0, max=0, len=(0-0)
nRate  =0.0, max=0, len=(0-0)
genome=1
PERFECT_READ_RATIO=0.0
ADD_ERRORS_FROM_QUALITY=true
REPLACE_NOREF=false
paired=false
read length=1000-5000
reads=10000
Wrote NC_010468_synth_pacbio.fq.gz
Time:     5.268 seconds.
```

1. Generating more complex/interesting synthetic datasets that can simulate a metagenome.

In practice you would likely want to do the simulation with many genomes but for simplicity we are going to use four. We already have our Ebola ( `AF086833.fa` ) and E. coli ( `NC_010468.fa` ) genomes from previous studies. Let us grab two more genomes from NCBI.

```
efetch -db=nuccore -format=fasta -id=NC_009614 > NC_009614.fa
efetch -db=nuccore -format=fasta -id=NC_013520 > NC_013520.fa
```

Note: coverage=X will automatically set "reads" to a level that will give X average coverage (decimal point is allowed).

metagenome will assign each scaffold a random exponential variable, which decides the probability that a read be generated from that scaffold. So, if you concatenate together 20 bacterial genomes, you can run randomreads and get a metagenomic-like distribution. It could also be used for RNA-seq when using a transcriptome reference.

The coverage is decided on a per-reference-sequence level, so if a bacterial assembly has more than one contig, you may want to glue them together first with `fuse.sh` before concatenating them with the other references.

First we need to `cat` our references in a single file.

```
cat NC_010468.fa NC_013520.fa NC_009614.fa AF086833.fa > metag.fa
```

We are then going to use this as an input for the metagenome generation followed by `reformat.sh` to separate the reads into two files.

```
randomreads.sh ref=metag.fa out=stdout length=300 paired=t mininsert=100 maxinsert=400 reads=5
000000 metagenome=t coverage=3 | reformat.sh in=stdin out1=metag_R1.fq.gz out2=metag_R2.fq.gz
interleaved addslash

# This produces the following log

Set genScaffoldInfo=true
Deleting chr1.chrom.gz
Input is being processed as paired
Writing chunk 1
Waiting for writing to finish.
Finished.
d=0.21242003534101328, sum=0.21242003534101328
d=0.03351212974244176, sum=0.24593216508345506
d=0.010454110909057856, sum=0.2563862759925129
d=0.021468052065878238, sum=0.27785432805839116
[0.21242003534101328, 0.03351212974244176, 0.010454110909057856, 0.021468052065878238]
sum=0.27785432805839116, mult=3.5990081816895425
d=0.7645014451470885, sum=0.7645014451470885
d=0.12061042912888936, sum=0.8851118742759778
d=0.037624430693989123, sum=0.922736304969967
d=0.07726369503003286, sum=0.9999999999999998
[0.7645014451470885, 0.8851118742759778, 0.922736304969967, 0.9999999999999998]
sum=0.9999999999999998
snpRate=0.0, max=0, unique=true
insRate=0.0, max=0, len=(0-0)
delRate=0.0, max=0, len=(0-0)
subRate=0.0, max=0, len=(0-0)
nRate  =0.0, max=0, len=(0-0)
genome=1
PERFECT_READ_RATIO=0.0
ADD_ERRORS_FROM_QUALITY=true
REPLACE_NOREF=false
paired=true
read length=300
reads=60302
insert size=100-400
Wrote stdout
Time:     6.215 seconds.
Input:                      120604 reads              36181200 bases
Output:                     120604 reads (100.00%)     36181200 bases (100.00%)

Time:                            6.584 seconds.
Reads Processed:        120k     18.32k reads/sec
Bases Processed:      36181k     5.50m bases/sec
```

1. We can generate the same set of reads with and without SNPs by fixing the seed to a positive number like this.

Data with no SNP.

```
randomreads.sh ref=NC_010468_new.fa out=NC_010468_no_snp.fq.gz maxsnps=0 adderrors=false reads=
1000 minlength=180 maxlength=300 seed=5
```

The following log entry is generated

```
Set genScaffoldInfo=true
Deleting chr1.chrom.gz
Writing chunk 1
Waiting for writing to finish.
Finished.
snpRate=0.0, max=0, unique=true
insRate=0.0, max=0, len=(0-0)
delRate=0.0, max=0, len=(0-0)
subRate=0.0, max=0, len=(0-0)
nRate  =0.0, max=0, len=(0-0)
genome=1
PERFECT_READ_RATIO=0.0
ADD_ERRORS_FROM_QUALITY=false
REPLACE_NOREF=false
paired=false
read length=180-300
reads=1000
Wrote NC_010468_no_snp.fq.gz
Time:     0.643 seconds.
```

Data with 2 SNP.

```
randomreads.sh ref=NC_010468_new.fa out=NC_010468_with_snp.fq.gz maxsnps=2 snprate=1 adderrors=
false reads=1000 minlength=180 maxlength=300 seed=5
```

The following log entry is generated

```
Set genScaffoldInfo=true
Deleting chr1.chrom.gz
Writing chunk 1
Waiting for writing to finish.
Finished.
snpRate=1.0, max=2, unique=true
insRate=0.0, max=0, len=(0-0)
delRate=0.0, max=0, len=(0-0)
subRate=0.0, max=0, len=(0-0)
nRate  =0.0, max=0, len=(0-0)
genome=1
PERFECT_READ_RATIO=0.0
ADD_ERRORS_FROM_QUALITY=false
REPLACE_NOREF=false
paired=false
read length=180-300
reads=1000
Wrote NC_010468_with_snp.fq.gz
Time:     0.640 seconds.
```

# How do I simulate perfect 1x coverage (100%) for a reference?

You can use `shread.sh` instead of `randomreads.sh` as follows `shred.sh in=ref.fasta out=reads.fastq length=200`

The difference is that `randomReads.sh` will make reads in a random order from random locations, ensuring flat coverage on average, but it won't ensure 100% coverage unless you generate many fold depth. `shred.sh`, on the other hand, gives you exactly 1x depth and exactly 100% coverage (it is not capable of modelling errors). So, the use-cases are different.

# How can I generate "fake" paired-end reads from single-end data?

```
bfakereads.sh in=reads.fq.gz out1=r1.fq.gz out2=r2.fq.gz length=100
```

That will generate fake pairs from the input file, with whatever length you want (maximum of input read length). We use it in some cases for generating a fake LMP library for scaffolding from a set of contigs. Read 1 will be from the left end, and read 2 will be reverse-complemented and from the right end; both will retain the correct original qualities. And " /1" " /2" will be suffixed after the read name.

# Reformat (and filter) sequence data

Used for: Format conversion/manipulation/analysis of sequence data.

Description: Reformats reads to change ASCII quality encoding, interleaving, file format, or compression format. Optionally performs additional functions such as quality trimming, subsetting, and subsampling. Supports fastq, fasta, fasta+qual, scarf, oneline, sam, bam, gzip, bz2.

Usage: `reformat.sh in=<file> in2=<file2> out=<outfile> out2=<outfile2>`

Note 1: `in2=` and `out2=` are for paired reads and are optional.

Note 2: If input is paired and there is only one output file, reads will be written interleaved.

# How do I find unknown primers/count k-mers?

Let us say you expect your primers to be in the first 20 bases of your reads. We would start by capturing first 20 bases in a file by:

```
reformat.sh in=reads.fq out=trimmed.fq ftr=19
```

This will trim all but the first 20 bases (all bases after position 19, zero-based). `ftr` stands for force trim right.

We can then do

```
kmercountexact.sh in=trimmed.fq out=counts.txt fastadump=f mincount=10 k=20 rcomp=f
```

This will generate a file containing the counts of all 20-mers that occurred at least 10 times, in a 2-column format that is easy to sort.

```
ACCGTTACCGTTACCGTTAC    100
AAATTTTTTTCCCCCCCCCC    85
```

If the primer is 20 bp long it should be easy to pick it out.

# How do I remove `N` basecalls?

You can use `bbduk` or `reformat` with `qtrim=rl trimq=1` options.

This will only trim trailing and leading bases with Q-score below 1, which means Q0, which means N (in either fasta or fastq format). The BBMap package automatically changes q-scores of Ns that are above 0 to 0 and called bases with q-scores below 2 to 2, since occasionally some Illumina software versions produces odd things like a handful of Q0 called bases or Ns with Q>0, neither of which make any sense in the Phred scale.

# How can I sample a read file?

Sampling random reads from a file is easy to do with `reformat.sh`.

Following command samples 300 reads from

```
reformat.sh in=SRR1972739_1.fastq out=SRR1972739_1_sample.fastq sample=300

# This should generate a log

Input is being processed as unpaired
Input:                      10000 reads              1010000 bases
Output:                     300 reads (3.00%)     30300 bases (3.00%)

Time:                           0.254 seconds.
Reads Processed:       10000     39.30k reads/sec
Bases Processed:       1010k     3.97m bases/sec
```

You can sample as a % of the total reads by doing (10% in following example):

```
reformat.sh in=SRR1972739_1.fastq out=SRR1972739_1_10perc.fastq samplerate=0.1

# This generates the following log. We can see that ~10% reads were sampled.

Input is being processed as unpaired
Input:                      10000 reads              1010000 bases
Processed:                  1025 reads              103525 bases
Output:                     1025 reads (10.25%)     103525 bases (10.25%)

Time:                           0.363 seconds.
Reads Processed:        1025     2.83k reads/sec
Bases Processed:        103k     0.29m bases/sec
```

# How can I verify proper pairing of reads for paired-end data?

You can verify if the reads are in proper pairs (order) by doing following for interleaved reads:

```
reformat.sh in=reads.fq.gz verifypairing
```

If those reads are in separate files then use instead:

```
reformat.sh in1=r1.fq.gz in2=r2.fq.gz vpair
```

# How can I filter a SAM/BAM file by read length?

You can filter a SAM/BAM file to extract reads that satisfy certain read length criteria.

```
reformat.sh in=x.sam out=y.sam minlength=50 maxlength=200
```

# How do I filter/detect spliced reads in q SAM/BAM file?

You can set `maxdellen` (50 in example below) to whatever length deletion event you consider the minimum to signify splicing, which depends on the organism.

```
reformat.sh in=mapped.bam out=filtered.bam maxdellen=50
```

# How can I collect unmapped reads from a SAM/BAM file?

You can separate unmapped reads from a SAM/BAM file using `reformat.sh` as follows

```
reformat.sh in=all.sam out=unmapped.fq.gz unmappedonly
repair.sh in=unmapped.fq.gz  out=r1.fq.gz out2=r2.fq.gz outs=singleton.fq
```

# Deduplicating sequence datasets

Used for: Removing duplicate reads/sequences from a set of files.

Description: Accepts one or more files containing sets of sequences (reads or scaffolds). Removes duplicate sequences, which may be specified to be exact matches, subsequences, or sequences within some percent identity. Can also find overlapping sequences and group them into clusters. Input may be fasta or fastq, compressed or uncompressed.

```
# A simple dedupe run can be done by:

dedupe.sh in=<file or stdin> out=<file or stdout>

# An example of running Dedupe for clustering short reads:

dedupe.sh in=x.fq am=f ac=f fo c pc rnc=f mcs=4 mo=100 s=1 pto cc qin=33 csf=stats.txt pattern=
cluster_%.fq dot=graph.dot
```

```
# We are going to download a test file for dedupe.sh run from Biostarhandbook data site.

wget http://data.biostarhandbook.com/reads/duplicated-reads.fq.gz

dedupe.sh in=duplicated-reads.fq.gz out=deduped.fq.gz
```

dedupe also produces a comprehensive set of statistics after the run completes. They are generally be written to \ but can be easily captured to a log file (example below).

```
Initial:
Memory: max=39156m, free=38543m, used=613m

Found 7445 duplicates.
Finished exact matches.    Time: 0.236 seconds.
Memory: max=39156m, free=36296m, used=2860m

Found 0 contained sequences.
Finished containment.      Time: 0.135 seconds.
Memory: max=39156m, free=34457m, used=4699m

Removed 0 invalid entries.
Finished invalid removal.  Time: 0.008 seconds.
Memory: max=39156m, free=34457m, used=4699m

Input:                          15000 reads             1515000 bases.
Duplicates:                     7445 reads (49.63%)     751945 bases (49.63%)           0 coll
isions.
Containments:                   0 reads (0.00%)         0 bases (0.00%)         113772 collisi
ons.
Result:                         7555 reads (50.37%)     763055 bases (50.37%)

Printed output.            Time: 0.239 seconds.
Memory: max=39156m, free=34049m, used=5107m

Time:                           0.627 seconds.
Reads Processed:       15000    23.92k reads/sec
Bases Processed:       1515k    2.42m bases/sec
```

# Read error correction and assembly of data

Used for: Assembly/error correction of data. Works especially well with viral genomes

Description: Uses kmer counts to assemble contigs, extend sequences, or error-correct reads. Tadpole has no upper bound for kmer length, but some values are not supported. Specifically, it allows 1-31, multiples of 2 from 32-62, multiples of 3 from 63-93, etc.

```
Assembly:     tadpole.sh in=<reads> out=<contigs>
Extension:    tadpole.sh in=<reads> out=<extended> mode=extend
Correction:   tadpole.sh in=<reads> out=<corrected> mode=correct
```

Let us use some of the Ebola test data we used for our trimming example above.

```
# Try varying the k value to see what happens to the number of contigs produced.

tadpole.sh k=20 in1=SRR1972739_1_trim.fastq in2=SRR1972739_2_trim.fastq out=ebola.contig.fa
```

Once the run completes `tadpole` produces a set of statistics about the run and result.

> Note: Since this is a small test dataset we don't do so well in terms of contigs we are producing.

```
Initialization Time:          0.177 seconds.

Loading kmers.

Estimated kmer capacity:     1465266554
After table allocation:
Memory: max=53389m, free=51996m, used=1393m

After loading:
Memory: max=53389m, free=48932m, used=4457m

Input:                          19900 reads         2003272 bases.
Unique Kmers:                   296264
Load Time:                      2.223 seconds.

Reads Processed:       19900     8.95k reads/sec
Bases Processed:       2003k     0.90m bases/sec

Building contigs.

Seeding with min count = 7900, 4646, 2733, 1607, 945, 556, 327, 192, 113, 66, 38, 22, 13, 7, 4
, 3 Unspecified format for output ebola.contig; defaulting to fasta.

After building contigs:
Memory: max=53389m, free=46425m, used=6964m

Bases generated:                19093
Contigs generated:              67
Longest contig:                 979
Contig-building time:           0.256 seconds.

Total Time:                   2.667 seconds.
```

# bcftools

Bcftools are a set of utilities that manipulate variant calls in the Variant Call Format (VCF) and its binary counterpart BCF. All commands work transparently with both VCFs and BCFs, both uncompressed and BGZF-compressed.

Bcftools is designed to work on a stream. It regards an input file "-" as the standard input (stdin) and outputs to the standard output (stdout). Several commands can thus be combined with Unix pipes.

- Webpage: http://www.htslib.org/
- Bcftools documentation: http://www.htslib.org/doc/bcftools.html
- The SNP calling method is described in: The Sequence Alignment/Map format and SAMtools

# Installation

```
brew install bcftools
```

or

```
conda install bcftools
```

Test that the installation succeeded:

```
bcftools
```

# FreeBayes Variation Caller

Bayesian haplotype-based polymorphism discovery and genotyping.

Publication: Haplotype-based variant detection from short-read sequencing

Website: https://github.com/ekg/freebayes

Prerequisites install `cmake` and `wget` :

# Install

```
brew install freebayes
```

or

```
conda install freebayes
```

# Source code installation

```
cd ~/src
git clone --recursive git://github.com/ekg/freebayes.git
cd freebayes
make
# For reasons that I don't yet understand the first make command
# fails but the second succeeds.
make
ln -s ~/src/freebayes/bin/freebayes ~/bin/freebayes
```

Test installation by running:

```
freebayes
```

# GATK Variation Caller

The Genome Analysis Toolkit or GATK is a software package for analysis of high-throughput sequencing data. The toolkit offers a wide variety of tools, with a primary focus on variant discovery and genotyping.

Website: https://www.broadinstitute.org/gatk/

The software works on all platforms that run java.

## Installation

```
brew install gatk
```

or

```
conda install gatk
```

You may need to manually obtain the GATK file and install it when the tool first runs.

## Source code installation

Register on the webpage and download the `GenomeAnalysisTK-3.4-46.tar.bz2` file. The version of the software will change in time.

```
mkdir -p ~/src/gatk
mv ~/Downloads/GenomeAnalysisTK-3.4-46.tar.bz2 ~/src/gatk
cd ~/src/gatk
tar jxvf GenomeAnalysisTK-3.4-46.tar.bz2
```

Test the installation

```
java -jar ~/src/gatk/GenomeAnalysisTK.jar -h
```

Create a script that launches `gatk`:

```
echo '#!/bin/bash' > ~/bin/gatk
echo 'java -jar ~/src/gatk/GenomeAnalysisTK.jar $@' >> ~/bin/gatk

# Make the script executable
chmod +x ~/bin/gatk
```

# snpEff

Genetic variant annotation and effect prediction toolbox.

Website: http://snpeff.sourceforge.net/

The software works on all platforms that run java.

# Installation

The snpEff installation with homebrew exists but at the time of writing that did not seem to work properly. Try

```
brew install snpeff
```

if, it raises problems please install from source.

On Linux do:

```
conda -y install snpeff
```

# Source code installation

```
# A separate directory for downloads, source and executables
mkdir -p ~/down ~/src ~/bin

# Download the source.
curl -kL http://downloads.sourceforge.net/project/snpeff/snpEff_latest_core.zip > ~/down/snpEf
f_latest_core.zip

# Unpack into the source directory
unzip ~/down/snpEff_latest_core.zip -d ~/src
```

Create a starter script to snpEff

```
echo '#!/bin/bash' > ~/bin/snpEff
echo 'java -jar ~/src/snpEff/snpEff.jar $@' >> ~/bin/snpEff
chmod +x ~/bin/snpEff
```

Test the installation

```
snpEff databases | grep ebola
```

Download the ebola database

```
snpEff download ebola_zaire
```

# Using snpEff

To use `snpEff` you will need to download annotation databases build for the genomic builds of interest.

```
# Download information on all known databases
snpEff databases > listing.txt

# Look at all options.
more listing.txt
```

# Build a custom annotation

snpEff allows building a custom annotation, thought the process is a bit more complicated than it absolutely needs to be. There are a number of assumptions and expectations for file locations and naming that need to be followed.

Let's build annotation for accession number `AF086833`

```
#This is the accession number.
ACC=AF086833

# The data needs to be stored in a separate subdirectory
DATA=./db/$ACC

# Make a directory to store the data in.
mkdir -p $DATA

# Download the fasta and genbank files into the  db/AF086833 directory.
efetch -db=nuccore -format=gb -id=$ACC > $DATA/genes.gbk
efetch -db=nuccore -format=fasta -id=$ACC > $DATA/$ACC.fasta
```

We need to build a configuration file that specifies information on the data. Let's name this file `AF086833.config` , it contains:

```
#Ebola Genome AF086833
AF086833.genome : Ebola virus - Mayinga, Zaire
AF086833.codonTable : Bacterial_and_Plant_Plastid
```

To build the new annotation we need to concatenate the original snpEff data base and our new fileBuild the new annotations

```
snpEff.jar build -genbank -c ./snpEff.config -dataDir ./db -v $ACC
```

# TopHat

Note that this program requires that Bowtie2 be installed as well.

Website: https://ccb.jhu.edu/software/tophat/index.shtml

# Create new environment

Tophat requires the presence of Python 2. This may or may not already be installed. If not you will need to create a conda environment:

```
conda create -n tophat python=2
```

Activate the environment with

```
source activate tophat
```

# Installation

```
brew install tophat
```

# Mac OSX manual install:

```
cd ~/src
curl -kO https://ccb.jhu.edu/software/tophat/downloads/tophat-2.1.0.OSX_x86_64.tar.gz
tar xzvf tophat-2.1.0.OSX_x86_64.tar.gz

# This is a suite of tools thus is best to
# add the entire directory to the search path.
export PATH=~/src/tophat-2.1.0.OSX_x86_64:$PATH

# See the progams included with TopHat.
ls ~/src/tophat-2.1.0.OSX_x86_64
```

# Linux manual install

```
cd ~/src
curl -kO https://ccb.jhu.edu/software/tophat/downloads/tophat-2.1.0.Linux_x86_64.tar.gz
tar xzvf tophat-2.1.0.Linux_x86_64.tar.gz

# This is a suite of tools thus is best to
# add the entire directory to the search path.
export PATH=~/src/tophat-2.1.0.Linux_x86_64:$PATH

# See the progams included with TopHat.
ls ~/src/tophat-2.1.0.Linux_x86_64
```

# Test TopHat

```
tophat -h
```

# Cufflinks/Cuffdiff Transcriptome Assembler and Differential Expression computation

Website: http://cole-trapnell-lab.github.io/cufflinks/

## Install

```
brew install cufflinks
```

or

```
conda install -y cufflinks
```

**Note**: The Mac OSX cufflinks obtained as above often crashes even though it is the newest version. I believe there is a programming error that causes this. We may need to roll back and use an older version of cufflinks as below.

## MacOS

```
cd ~/src
curl -kL http://cole-trapnell-lab.github.io/cufflinks/assets/downloads/cufflinks-2.1.1.OSX_x86
_64.tar.gz | tar xzv

# Enable the path.
echo 'export PATH=~/src/cufflinks-2.1.1.OSX_x86_64:$PATH' >> ~/.bashrc
source ~/.bashrc
```

## Linux

```
cd ~/src
curl -kL http://cole-trapnell-lab.github.io/cufflinks/assets/downloads/cufflinks-2.2.1.Linux_x
86_64.tar.gz | tar xzv

# Enable the path.
echo 'export PATH=~/src/cufflinks-2.2.1.Linux_x86_64:$PATH' >> ~/.bashrc
source ~/.bashrc
```

# HISAT2 aligner

HISAT2 is a fast and sensitive alignment program for mapping next-generation sequencing reads (both DNA and RNA) against the general human population (as well as against a single reference genome).

Website: HiSat Website

# Install

On Mac OSX:

```
brew install hisat2
```

on Linux:

```
conda install -y hisat2
```

# Binary installation

Find and download the Linux version (also applies as an alternative on Mac OSX)

```
cd ~/src
curl -OL ftp://ftp.ccb.jhu.edu/pub/infphilo/hisat2/downloads/hisat2-2.0.3-beta-Linux_x86_64.zip
unzip hisat2-2.0.3-beta-Linux_x86_64.zip
```

The **Linux** link the executables:

```
ln -s ~/src/hisat2-2.0.3-beta/hisat2-build ~/bin
ln -s ~/src/hisat2-2.0.3-beta/hisat2 ~/bin
```

Test that the programs work with

```
hisat2
```

# SubRead

This package contains an aligner and feature counter. The programs that will use are:

- `subread-align` - short read aligner
- `featureCounts` - feature counter

Website: http://bioinf.wehi.edu.au/subread-package/

# Install

There is a bioconda package for `subread`:

```
conda install -y subread
```

# Source code install

For all other platforms Obtain the source code:

```
mkdir -p ~/src ~/bin
cd ~/src
curl -OkL http://sourceforge.net/projects/subread/files/subread-1.5.1/subread-1.5.1-source.tar
.gz
tar zxvf subread-1.5.1-source.tar.gz
cd subread-1.5.1-source/src
```

## Compile on Mac OSX

```
make -f Makefile.MacOS
```

## Compile on Linux

```
make -f Makefile.Linux
```

Link the executables into the `~bin` folder:

```
ln -fs ~/src/subread-1.5.1-source/bin/featureCounts ~/bin
ln -fs ~/src/subread-1.5.1-source/bin/subread-align ~/bin
ln -fs ~/src/subread-1.5.1-source/bin/subread-buildindex ~/bin
```

Check that the programs work

```
featureCounts
subread-align
```

# FeatureCounts

The `featureCounts` program installs as part of the `Subread` package.

See Installing Subread

# Kallisto Aligner

## Install

MacOS:

```
brew install kallisto
```

on Linux:

```
conda install -y kallisto
```

## Install as binary

```
# Make the source directory.
mkdir -p ~/src ~/bin

# The URL to the download.
URL=https://github.com/pachterlab/kallisto/releases/download/v0.43.0/kallisto_linux-v0.43.0.tar.gz

# Unpack to ~/src
curl -kLO | tar zvz -C ~/src

# Link kallisto to the path
ln -s kallisto_linux-v0.43.0/kallisto ~/bin
```

# MACS: Model Based Analysis for ChIP-Seq data

MACS is short for Model Based Analysis for ChIP-Seq data, which was developed in Sheirly Liu's lab at Harvard by Tao Liu. It is one of the most popular peak-calling algorithms used in literature. First, you need to install MACS. Please refer to the section that discusses creating python enviroments.

## Installation

```
# Create an python2.7 based enviroment.
conda create --name py2 python=2.7

# Activate the environment.
source activate py2

# Install MACS into this environment.
conda install macs
```

## Usage

We can now run `macs` to see the usage instructions:

```
macs
Usage: macs <-t tfile> [-n name] [-g genomesize] [options]

Example: macs -t ChIP.bam -c Control.bam -f BAM -g h -n test -w --call-subpeaks
...
```

# GEM: Genome wide Event finding and Motif discovery

Citation:

- High Resolution Genome Wide Binding Event Finding and Motif Discovery Reveals Transcription Factor Spatial Binding Constraints, PLoS Computational Biology (2012)

Website:

- Genome wide Event finding and Motif discovery

**Note:** Not to be confused with a similarly named tool GEM RNA-Seq aligner published in Nature Methods, also in 2012.

## Installation

```
# The URL of the tool.
URL=http://groups.csail.mit.edu/cgs/gem/download/gem.v2.7.tar.gz

# This will be the source directory.
mkdir -p ~/src
cd ~/src

# Download and unpack the tool
curl -O http://groups.csail.mit.edu/cgs/gem/download/gem.v2.7.tar.gz
tar xzv gem.v2.7.tar.gz
```

View the files:

```
ls ~/src/gem
```

## Usage

```
java -jar ~/src/gem/gem.jar
```

In general the tool requires quite a few paramters to run.

# About the author

Dr. István Albert is renowned for being at the forefront of the Bioinformatics frontier and pushing the field forward in new directions.

He helped pioneer the Galaxy Bioinformatics Platform, an open source web-based tool that allows users to perform and share data-intensive biomedical research. He also developed BooleanNet, a biological system simulation software and the GeneTrack software platform that automates and processes large sets of biological data.

Currently, István is the lead software developer, maintainer and administrator of the website, Biostars: Bioinformatics Questions and Answers, the most comprehensive information source in Bioinformatics that is visited by millions of people every year.

Besides being well-known for his scientific innovations in three different scientific fields: Physics, Computer Science and Biology, Dr. Albert is a celebrated educator. The courses he develops are consistently highly ranked and revered by students. In 2014, István was awarded the Paul M. Althouse Outstanding Teaching Award at Penn State for the superb quality of his lectures. The lecture information that has garnered him such high praise is the very same material that you will find in the Biostar Handbook.

Dr. Albert is a Professor of Bioinformatics at Pennsylvania State University and the Director of the Bioinformatics Consulting Center at this institute. István established this cutting-edge research facility where he has advised and assisted dozens of brilliant young scientists in their ambitious endeavors. He also leads the Graduate Certificate Program in Applied Bioinformatics, a comprehensive Bioinformatics training course offered through the Penn State World Campus that allows anyone, anywhere, to attain in-depth knowledge of this exciting new field.

# Development timeline

## What does the book cover at this time?

Scroll down on the menu on the left to see the sections and the entries associated with them. An section with a label but no articles under it is currently under development and should have content released within a month.

## What comes in 2017 and beyond?

We encourage suggestions from readers on the topics they would like to see covered. Our plan currently includes the following:

- Genome and Transcriptome Assembly (early 2017)
- Metagenomics (early 2017)
- Chip-Seq (mid-2017)
- Workbook with assignments, exercises, and answer keys.
- Evaluations and tests. Allow readers to evaluate how well they understand the concepts.

All along we will expand and enrich the existing chapters. New methods and techniques will be added to each methodology. Obsolete approaches will be retired.

## How was this book developed?

This book is the result of a multi-year experience of teaching life scientists on how to analyze their data. Almost two years ago we decided to collect the materials into a book called the Biostar Handbook. Along the way, we tossed out a relatively complete version of it as it was deemed inadequate. As Fred Brooks once wrote in the Mythical Man Month "Plan To Throw One Away." And throw away one we did.

Then it occurred to us -- why are we writing a traditional book when the Biostars Q&A and online style works so much better. So we switched to that -- and found that this approach worked spectacularly well for both instructors and students as well!

## What was the timeline so far?

- **2010-2013**: The proposal for the BMMB Applied Bioinformatics accepted by the Penn State Graduate School. The course offered as an elective course at Penn State sees enrollment over capacity every single year since its launch. The course is practically rewritten from scratch every single year.
- **2013-2015**: A transition is made to the web-based distribution of materials. The code is reused from semester to semester and shared through a web repository. The material is enriched and improved every single year. We decide to combine all content into a single traditional book titled *The Biostar Handbook.*

- **Spring 2016**: The "first edition" of the Biostar Handbook is completed. It is a monolithic, "traditional" book designed to be printed out. The shortcomings of the model became painfully apparent. Some of the content in the book was already obsolete. The decision is made to transition to a web-based book that can be quickly and efficiently updated.
- **Fall 2016**: The web-book is launched as a textbook for the BMMB 852: Applied Bioinformatics course. Chapters are completed on week-by-week schedule as they are presented in class.