Introduction to Applied Bioinformatics

Bioinformatics Platform

Who are we?

Salim Bougouffa
Senior Bioinformatician

Husen Umer
Bioinformatics Scientist

Issaac Rajan
Staff Bioinformatician

Allan Kamau
Database Developer

Who are you?

YOUR NAME

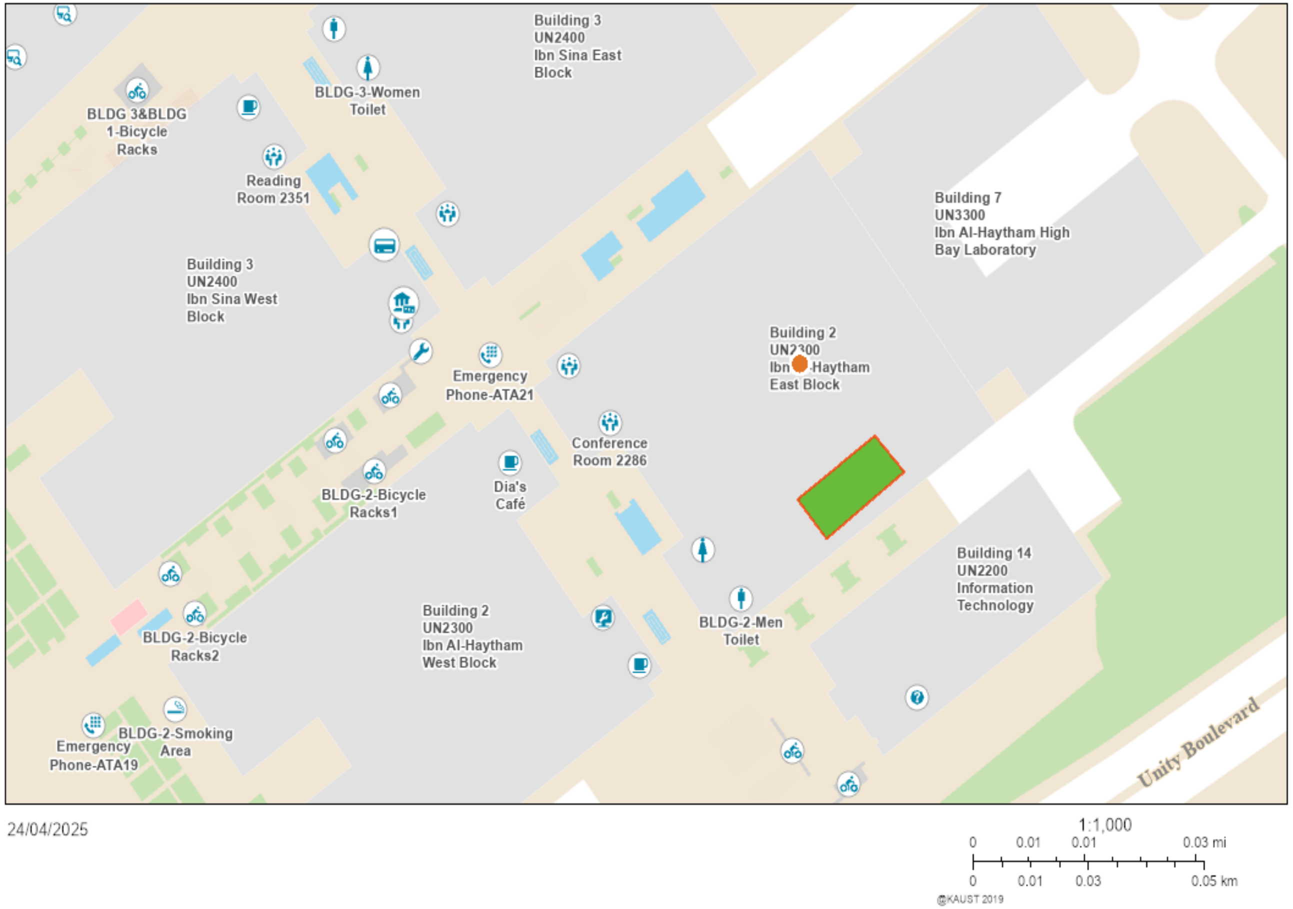
YOUR ROLE

YOUR GROUP

YOUR RESEARCH

Find us

bioinformatics@kaust.edu.sa



More courses to come

Tell us what you would like to see & Spread the word

Course Outcomes

📚 **Bioinformatics Concepts**
Understand core bioinformatics concepts and get insights into various bioinformatics applications

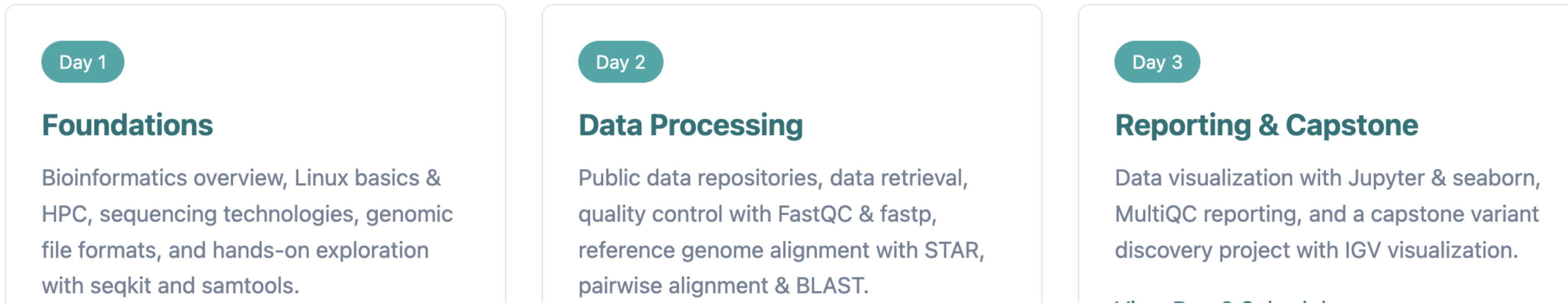
💻 **Sequencing Data**
Get familiar with short-read & long-read sequencing data and genomic file formats

🧬 **Linux & HPC**
Navigate the Linux command line and use HPC (Ibex) for bioinformatics analysis

📄 **Public Data Access**
Access public data repositories, perform QC, alignment, and visualization

🎓 **Practical Skills**
Gain practical bioinformatics skills including QC, alignment, BLAST, and visualization

Program Overview



Day 1

Foundations

Bioinformatics overview, Linux basics & HPC, sequencing technologies, genomic file formats, and hands-on exploration with seqkit and samtools.

Day 2

Data Processing

Public data repositories, data retrieval, quality control with FastQC & fastp, reference genome alignment with STAR, pairwise alignment & BLAST.

Day 3

Reporting & Capstone

Data visualization with Jupyter & seaborn, MultiQC reporting, and a capstone variant discovery project with IGV visualization.

Course webpage: https://bioinfo-kaust.github.io/introduction-to-bioinformatics

Lecture Outline

PART 1

What Is Bioinformatics?

Definitions, origins, and the landscape of the field

Defining Bioinformatics

Bioinformatics is the application of computational and statistical methods to understand biological data — bridging biology, computer science, mathematics, and statistics.

Biology-Centered View

Using computation to answer biological questions: gene function, evolutionary relationships, disease mechanisms

Computer Science View

Developing algorithms and software tools to process, store, and analyze large-scale biological datasets

Data Science View

Extracting knowledge from complex, high-dimensional biological data through statistical modeling and machine learning

A Brief History

1960s–70s	First protein sequence databases; Dayhoff's Atlas of Protein Sequence
1980s	BLAST algorithm; GenBank founded; sequence alignment becomes routine
1990s	Human Genome Project launched; rapid growth of public databases (NCBI, EMBL, DDBJ)
2000s	HGP completed; microarrays; next-gen sequencing arrives, data volume explodes
2010s	RNA-Seq, ChIP-Seq, single-cell genomics; cloud computing enters biology
2020s	AlphaFold, spatial transcriptomics, long-read sequencing, AI/ML become mainstream

Bioinformatics vs Related Fields

Field	Focus	Key Methods
Bioinformatics	Sequence & structure analysis, genomics tools	Alignment, annotation, phylogenetics
Computational Biology	Modeling biological systems & processes	Simulations, mathematical modeling
Systems Biology	Network-level understanding of organisms	Pathway analysis, flux modeling
Biostatistics	Statistical design for biological studies	Clinical trials, survival analysis, hypothesis testing
Data Science (Bio)	Patterns in large-scale biological data	Machine learning, deep learning, visualization

These fields overlap significantly — most working bioinformaticians draw from all of them.

The Data Explosion in Biology

~73M

Sequences in
UniProt (2024)

40+ PB

Data in NCBI
Sequence Read Archive

$200

Cost to sequence
a human genome today

Why This Matters

The Human Genome Project took 13 years and $2.7 billion. Today, a genome can be sequenced in hours for under $200. Biology has shifted from being data-poor to data-rich. The bottleneck is no longer generating data — it is analyzing, interpreting, and making sense of it. This is exactly the problem bioinformatics solves.

PART 2

Why Bioinformatics Matters Now

Real-world impact across research and industry

Where Bioinformatics Is Used

Genomics & Transcriptomics

Genome assembly, variant calling, differential gene expression, RNA-Seq analysis

Drug Discovery

Target identification, virtual screening, structure-based drug design, ADMET prediction

Precision Medicine

Patient stratification, pharmacogenomics, cancer genomics, biomarker discovery

Metagenomics

Microbial community profiling, 16S/ITS analysis, environmental DNA studies

Structural Biology

Protein structure prediction (AlphaFold), molecular docking, homology modeling

Agriculture & Ecology

Crop genomics, pathogen surveillance, conservation genetics, biodiversity analysis

The Modern Biologist Needs Computation

Traditional Biology	Modern Computational Biology
One gene at a time	Thousands of genes simultaneously
Manual data analysis in Excel	Automated pipelines & scripts
Small sample sizes	Population-scale datasets
Hypothesis-driven only	Hypothesis-generating + driven
Results in notebooks & PDFs	Shared code & reproducible workflows
Limited reproducibility	Version-controlled, auditable research

PART 3

Core Concepts & Mental Toolkit

Programming, data management, statistics, and reproducibility

Programming Fundamentals

Programming is not optional in modern bioinformatics. It is the language through which you communicate with your data and make your analyses reproducible.

Python	General-purpose, rich ecosystem (BioPython, pandas, scikit-learn, matplotlib). Best for: data wrangling, ML, automation, pipelines.
R	Statistical computing powerhouse (Bioconductor, DESeq2, ggplot2, edgeR). Best for: statistical analysis, visualization, RNA-Seq/omics.
Bash / Command Line	Essential for file manipulation, running tools, job scheduling on HPC clusters. Every bioinformatician must be comfortable on the command line.

Data Management Principles

F

Findable

Rich metadata, persistent identifiers (DOIs, accession numbers), indexed in searchable resources

A

Accessible

Retrievable via standard protocols (HTTP, FTP), clear access conditions, long-term availability

I

Interoperable

Use standard formats and controlled vocabularies (Gene Ontology, SNOMED), machine-readable metadata

R

Reusable

Clear licenses, provenance documented, community standards followed, sufficient for replication

Practical tips: Use organized directory structures. Keep raw data read-only. Document every processing step. Use meaningful file names. Back up your data (and your code). Deposit in public repositories (GEO, SRA, Zenodo).

Statistical Thinking for Bioinformatics

Distributions & Descriptive Statistics

Understand your data's shape before modeling it. Mean, median, variance, skewness. Biological data is rarely normal.

Hypothesis Testing

t-tests, ANOVA, chi-squared, Wilcoxon tests. Know when each is appropriate and what assumptions they require.

Multiple Testing Correction

Testing 20,000 genes = ~1,000 false positives at $p<0.05$. Bonferroni and Benjamini-Hochberg FDR corrections are essential.

Bayesian vs Frequentist

Many bioinformatics tools use Bayesian methods (e.g., BRAKER, BEAST). Understand priors, posteriors, and likelihoods.

Experimental Design

Replicates (biological vs technical), batch effects, confounders, power analysis. Poor design cannot be fixed by better statistics.

Dimensionality Reduction

PCA, t-SNE, UMAP for visualizing high-dimensional data. Crucial for scRNA-Seq, proteomics, metabolomics.

Reproducible Research

The Reproducibility Crisis: Studies estimate that 50–90% of published research findings cannot be independently reproduced. In computational biology, undocumented software versions, missing parameters, and unavailable code are the top culprits.

Version Control (Git)

Track every change to your code. Use GitHub/GitLab to share. Tag versions that produce published results. Never email scripts.

Environment Management

Conda, Docker, Singularity. Pin exact package versions. A conda environment.yml or Dockerfile is your reproducibility insurance.

Workflow Managers

Snakemake, Nextflow, CWL. Define analysis as a directed acyclic graph. Automatic dependency tracking and parallelization.

Literate Programming

Jupyter Notebooks, R Markdown, Quarto. Combine code, results, and narrative in one document. Show your reasoning.

PART 4

Thinking Like a Bioinformatician

Workflow design, problem decomposition, and getting started

A Typical Bioinformatics Workflow

1

Question

Define a clear, testable biological question

2

Data Acquisition

Obtain raw data (sequencing, public databases, experiments)

3

Quality Control

Assess data quality, trim adapters, filter low-quality reads

4

Processing

Alignment, assembly, quantification, variant calling

5

Analysis

Statistical testing, clustering, differential expression, enrichment

6

Interpretation

Biological context, pathway analysis, literature integration

7

Visualization

Plots, heatmaps, genome browsers, interactive dashboards

8

Reporting

Reproducible documentation, data deposition, publication

Common Pitfalls for Beginners

Running tools without understanding them

Read the documentation. Understand what each parameter does before using default settings.

Ignoring data quality

Always run QC first. Garbage in = garbage out. Check FastQC reports before any analysis.

Not controlling for multiple testing

If you test thousands of hypotheses, adjust your p-values. Report FDR-adjusted values.

Hardcoding paths and parameters

Use config files, command-line arguments, and relative paths for portable, shareable code.

Forgetting to document your work

Future-you in 6 months won't remember. Use README files, comments, and lab notebooks.

Working in isolation

Collaborate. Ask for code review. Use forums (Biostars, SEQanswers). Attend workshops.

Getting Started — Your Action Plan

Install Python/R and a code editor (VS Code). Create a GitHub account. Run your first script.

Complete a Python or R basics course (e.g., Software Carpentry). Learn to navigate the command line. Run FastQC on real data.

Work through a full tutorial pipeline (e.g., RNA-Seq with DESeq2 or variant calling with GATK). Start using Git daily.

Apply bioinformatics to your own research project. Join your local bioinformatics community. Build your first reproducible workflow.

Recommended Learning Resources

Courses & Tutorials

Software Carpentry (shell, Git, Python/R), Rosalind.info (bioinformatics problems), MIT OpenCourseWare, Coursera Genomic Data Science

Textbooks

"Bioinformatics Data Skills" (Vince Buffalo), "Biological Sequence Analysis" (Durbin et al.), "R for Data Science" (Wickham & Grolemund)

Communities

Biostars, SEQanswers, Bioconductor support, r/bioinformatics, local meetups & workshops, Galaxy Training Network

Practice Platforms

Galaxy (web-based, no coding required), Rosalind, HackerRank (coding skills), Kaggle (data science competitions with bio datasets)

Key Takeaways

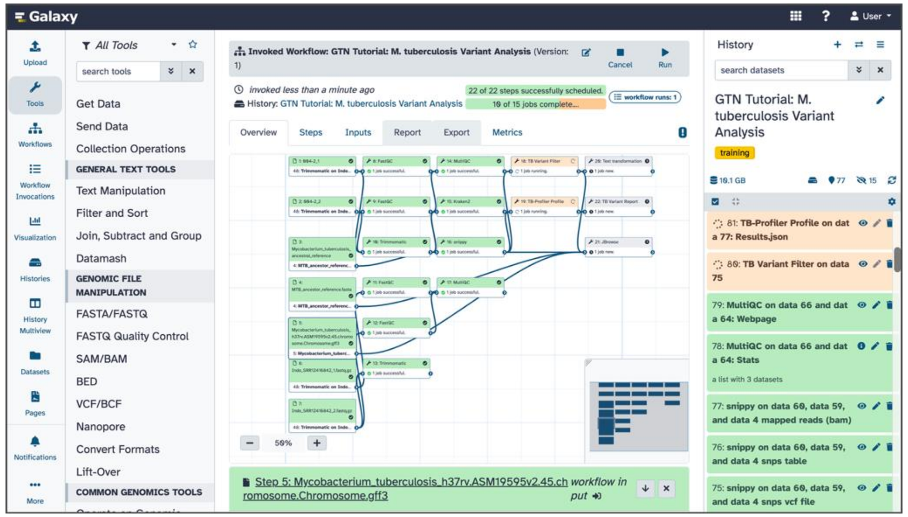
- ✓ Bioinformatics is the bridge between biological questions and computational answers — it is now a core skill, not a niche specialty.
- ✓ Biology has become a data science. The ability to handle large, complex datasets is as important as pipetting.
- ✓ Start with Python or R, learn the command line, and get comfortable with version control (Git) early.
- ✓ Statistical literacy — especially multiple testing correction and experimental design — will save you from false discoveries.
- ✓ Reproducibility is non-negotiable: document everything, use workflow managers, share your code.
- ✓ You don't need to master everything at once. Pick a project, learn what you need, build from there.

Linux Command Line

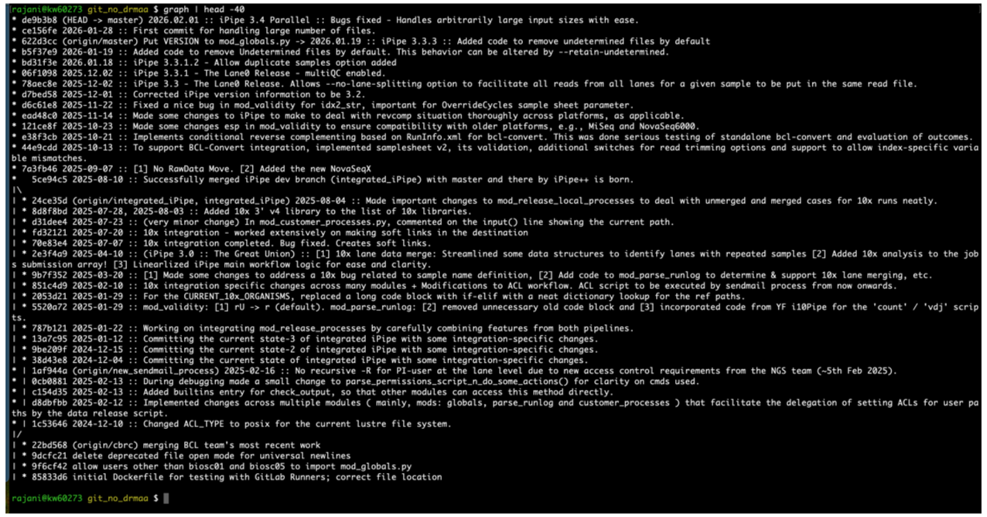
Introduction and Hands-on

Part 1 - Introduction

GUI vs Command line



Galaxy
Upload
Tools
Workflows
Workflow Invocations
Visualization
Histories
History Multiview
Datasets
Pages
Notifications
More
All Tools
search tools
Get Data
Send Data
Collection Operations
GENERAL TEXT TOOLS
Text Manipulation
Filter and Sort
Join, Subtract and Group
Datamash
GENOMIC FILE MANIPULATION
FASTA/FASTQ
FASTQ Quality Control
SAM/BAM
BED
VCF/BCF
Nanopore
Convert Formats
Lift-Over
COMMON GENOMICS TOOLS
Invoked Workflow: GTN Tutorial: M. tuberculosis Variant Analysis (Version: 1)
Cancel
Run
invoked less than a minute ago
History: GTN Tutorial: M. tuberculosis Variant Analysis
22 of 22 steps successfully scheduled.
10 of 15 jobs complete...
workflow runs: 1
Overview
Steps
Inputs
Report
Export
Metrics
56%
Step 5: Mycobacterium_tuberculosis_h37rv.ASM19595v2.45.chromosome.Chromosome.gff3
workflow input
History
search datasets
GTN Tutorial: M. tuberculosis Variant Analysis
training
16.1 GB
77
15
81: TB-Profiler Profile on data 77: Results.json
89: TB Variant Filter on data 75
79: MultiQC on data 66 and data 64: Webpage
78: MultiQC on data 66 and data 64: Stats
a list with 3 datasets
77: snippy on data 60, data 59, and data 4 mapped reads (bam)
76: snippy on data 60, data 59, and data 4 snps table
75: snippy on data 60, data 59, and data 4 snps vcf file
User



```
rajani@kw60273 git_no_drmaa $ graph | head -40
* de9b3b8 (HEAD -> master) 2026.02.01 :: iPipe 3.4 Parallel :: Bugs fixed - Handles arbitrarily large input sizes with ease.
* ce156fe 2026-01-28 :: First commit for handling large number of files.
* 622d3cc (origin/master) Put VERSION to mod_globals.py -> 2026.01.19 :: iPipe 3.3.3 :: Added code to remove undetermined files by default
* b5f37e9 2026-01-19 :: Added code to remove Undetermined files by default. This behavior can be altered by --retain-undetermined.
* bd31f3e 2026.01.18 :: iPipe 3.3.1.2 - Allow duplicate samples option added
* 06f1098 2025.12.02 :: iPipe 3.3.1 - The Lane0 Release - multiQC enabled.
* 78aec8e 2025-12-02 :: iPipe 3.3 - The Lane0 Release. Allows --no-lane-splitting option to facilitate all reads from all lanes for a given sample to be put in the same read file.
* d7bed58 2025-12-01 :: Corrected iPipe version information to be 3.2.
* d6c61e8 2025-11-22 :: Fixed a nice bug in mod_validity for idx2_str, important for OverrideCycles sample sheet parameter.
* ead48c0 2025-11-14 :: Made some changes to iPipe to make to deal with revcomp situation thoroughly across platforms, as applicable.
* 121ce8f 2025-10-23 :: Made some changes esp in mod_validity to ensure compatibility with older platforms, e.g., MiSeq and NovaSeq6000.
* e38f3cb 2025-10-21 :: Implements conditional reverse complementing based on RunInfo.xml for bcl-convert. This was done serious testing of standalone bcl-convert and evaluation of outcomes.
* 44e9cdd 2025-10-13 :: To support BCL-Convert integration, implemented samplesheet v2, its validation, additional switches for read trimming options and support to allow index-specific varia
ble mismatches.
* 7a3fb46 2025-09-07 :: [1] No RawData Move. [2] Added the new NovaSeqX
*   5ce94c5 2025-08-10 :: Successfully merged iPipe dev branch (integrated_iPipe) with master and there by iPipe++ is born.
|\
| * 24ce35d (origin/integrated_iPipe, integrated_iPipe) 2025-08-04 :: Made important changes to mod_release_local_processes to deal with unmerged and merged cases for 10x runs neatly.
| * 8d8f8bd 2025-07-28, 2025-08-03 :: Added 10x 3' v4 library to the list of 10x libraries.
| * d31dee4 2025-07-23 :: (very minor change) In mod_customer_processes.py, commented on the input() line showing the current path.
| * fd32121 2025-07-20 :: 10x integration - worked extensively on making soft links in the destination
| * 70e83e4 2025-07-07 :: 10x integration completed. Bug fixed. Creates soft links.
| * 2e3f4a9 2025-04-10 :: (iPipe 3.0 :: The Great Union) :: [1] 10x lane data merge: Streamlined some data structures to identify lanes with repeated samples [2] Added 10x analysis to the job
s submission array! [3] Linearlized iPipe main workflow logic for ease and clarity.
| * 9b7f352 2025-03-20 :: [1] Made some changes to address a 10x bug related to sample name definition, [2] Add code to mod_parse_runlog to determine & support 10x lane merging, etc.
| * 851c4d9 2025-02-10 :: 10x integration specific changes across many modules + Modifications to ACL workflow. ACL script to be executed by sendmail process from now onwards.
| * 2053d21 2025-01-29 :: For the CURRENT_10x_ORGANISMS, replaced a long code block with if-elif with a neat dictionary lookup for the ref paths.
| * 5520a72 2025-01-29 :: mod_validity: [1] rU -> r (default). mod_parse_runlog: [2] removed unnecessary old code block and [3] incorporated code from YF i10Pipe for the 'count' / 'vdj' scrip
ts.
| * 787b121 2025-01-22 :: Working on integrating mod_release_processes by carefully combining features from both pipelines.
| * 13a7c95 2025-01-12 :: Committing the current state-3 of integrated iPipe with some integration-specific changes.
| * 9be209f 2024-12-15 :: Committing the current state-2 of integrated iPipe with some integration-specific changes.
| * 38d43e8 2024-12-04 :: Committing the current state of integrated iPipe with some integration-specific changes.
* | 1af944a (origin/new_sendmail_process) 2025-02-16 :: No recursive -R for PI-user at the lane level due to new access control requirements from the NGS team (~5th Feb 2025).
* | 0cb0881 2025-02-13 :: During debugging made a small change to parse_permissions_script_n_do_some_actions() for clarity on cmds used.
* | c154d35 2025-02-13 :: Added builtins entry for check_output, so that other modules can access this method directly.
* | d8dbfbb 2025-02-12 :: Implemented changes across multiple modules ( mainly, mods: globals, parse_runlog and customer_processes ) that facilitate the delegation of setting ACLs for user pa
ths by the data release script.
* | 1c53646 2024-12-10 :: Changed ACL_TYPE to posix for the current lustre file system.
|/
| * 22bd568 (origin/cbrc) merging BCL team's most recent work
| * 9dcfc21 delete deprecated file open mode for universal newlines
| * 9f6cf42 allow users other than biosc01 and biosc05 to import mod_globals.py
| * 85833d6 initial Dockerfile for testing with GitLab Runners; correct file location

rajani@kw60273 git_no_drmaa $ 
```

GUI vs Command Line

GUI (Graphical User Interface)

- Point-and-click interface
- Visual feedback
- Intuitive for beginners
- Mouse-driven navigation
- Limited automation
- Resource intensive

CLI (Command Line Interface)

- Text-based commands
- Precise control
- Powerful for experts
- Keyboard-driven
- Easy automation (scripts)
- Lightweight & fast

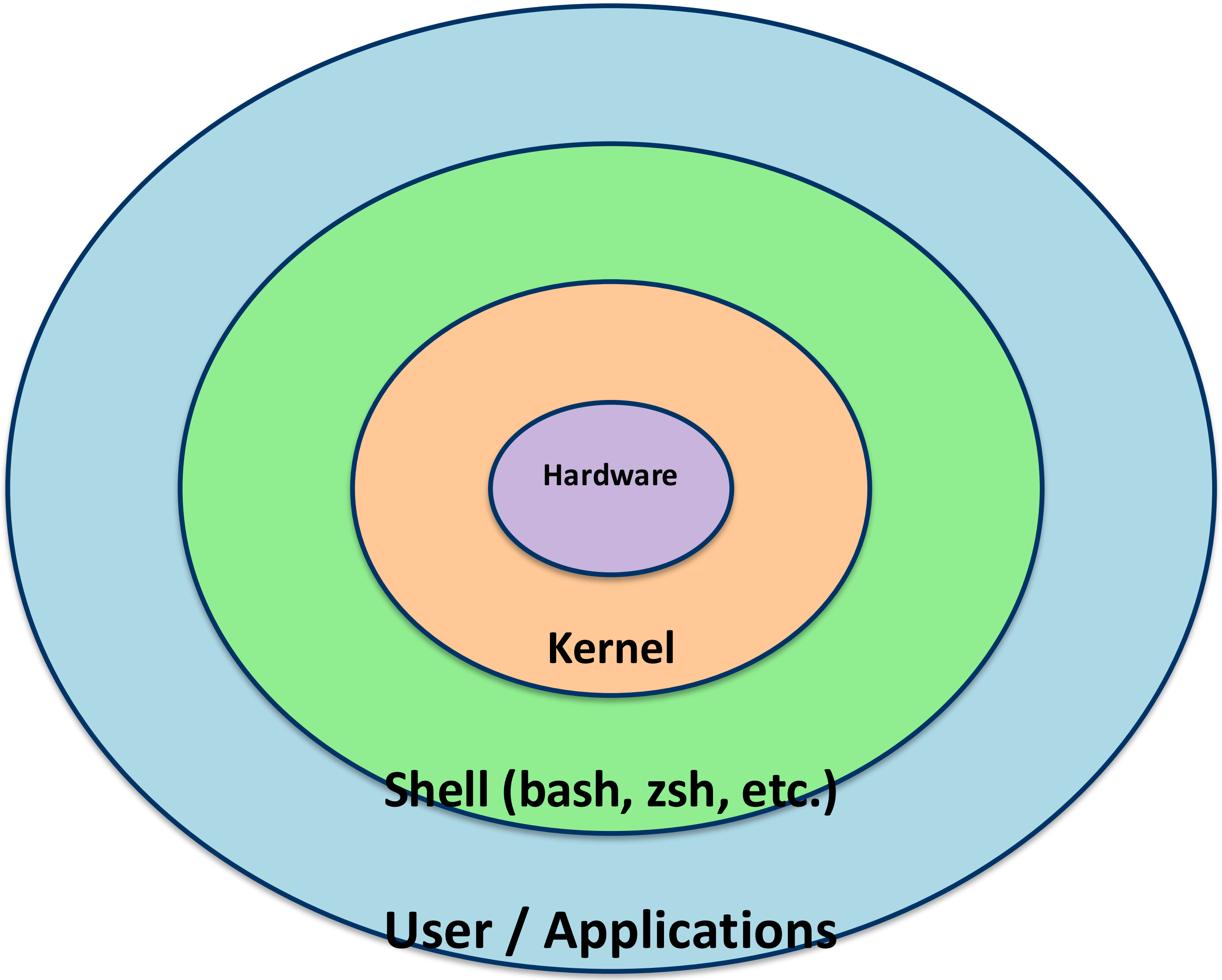
Linux and the Shell

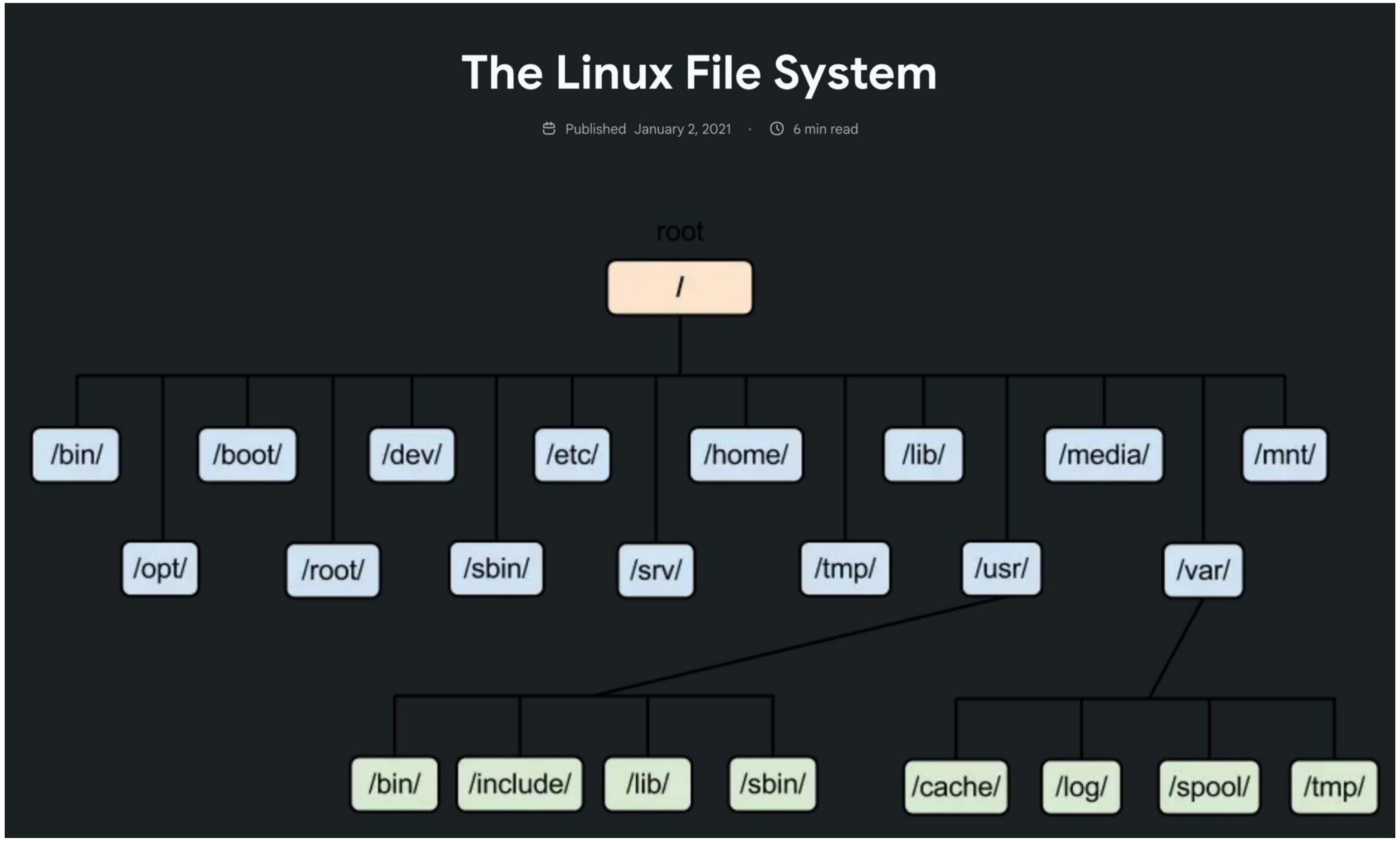
User:
Interacts with the system

Shell:
Command interpreter

Kernel:
Core of the OS

Hardware:
Physical components





Source: https://goldinscrib.hashnode.dev/the-linux-file-system

Part 2 – Quick Overview

How do we interact with the Shell?

Understanding the File System

Categorization of Commands

- **Navigation the file system**
  `pwd, cd, ls`
- **File operations**
  `cp, mv, rm, mkdir`
- **Viewing file contents**
  `cat, less, head, tail`

Combining Commands

- **Pipes**
  `|`
  Connect commands
- **Output redirection**
  `> , >>`
  Write to files
- **Read from input**
  `<`
  Read from files

Part 3 - Hands-on

[Optional] Login to Ibex

Command:

```
ssh <USERNAME>@ilogin.ibex.kaust.edu.sa
```

Knowing where you are in the file system

Command:

```
pwd
```

Soon after login:

```
Linux:  /home/<USERNAME>
MacOS:  /Users/<USERNAME>
```

Listing contents – Introducing options

Command:

```
ls

ls -l

ls -a

ls -al
```

Knowing more about a command

Command:

```
man  <command>

man pwd

man ls
```

Let's make a directory: linux_is_fun

Command:

```
mkdir linux_is_fun
```

Expected outcome:

```
Linux: /home/<USERNAME>/linux_is_fun
MacOS: /Users/<USERNAME>/linux_is_fun
```

Making nested directories…

Requirement:

```
Linux: /home/<USERNAME>/linux_is_fun/dir1/dir2
MacOS: /Users/<USERNAME>/linux_is_fun/dir1/dir2
```

Making a nested directory structure

Command:

```
mkdir -p linux_is_fun/dir1/dir2
```

Requirement: linux_is_fun/dir1/dir2

Expected outcome:

```
Linux: /home/<USERNAME>/linux_is_fun/dir1/dir2
MacOS: /Users/<USERNAME>/linux_is_fun/dir1/dir2
```

Directory navigation [1] Traverse to the newly created directory

Command:

```
cd linux_is_fun
```

To:

```
Linux: /home/<USERNAME>/linux_is_fun
MacOS: /Users/<USERNAME>/linux_is_fun
```

Creating a new file -- Text Editors!

Command:

```
nano

nano file1.txt
```

Expected outcome:

```
Linux: /home/<USERNAME>/linux_is_fun/file1.txt
MacOS: /Users/<USERNAME>/linux_is_fun/file1.txt
```

Other ways to make a new file / append…

Command:

```
cat >  file1.txt # begins a new file

cat >> file1.txt # appends
```

Expected outcome:

```
Linux: /home/<USERNAME>/linux_is_fun/file1.txt
MacOS: /Users/<USERNAME>/linux_is_fun/file1.txt
```

Copying files...

Command:

```
cp file1.txt file2.txt
```

Question: How will you verify if file2.txt exists?

Ways to explore file contents

Command:

```
cat sequence.txt

less sequence.txt

head / tail
```

Moving files...

Command:

```
mv file2.txt dir1/.
```

Directory navigation [2] Changing to a different directory

Command:

```
cd dir1/dir2
```

To:

```
Linux: /home/<USERNAME>/linux_is_fun/dir1/dir2
MacOS: /Users/<USERNAME>/linux_is_fun/dir1/dir2
```

Directory navigation [3] Moving one-level up...

Command:

```
cd ..
```

To:

```
Linux: /home/<USERNAME>/linux_is_fun/dir1
MacOS: /Users/<USERNAME>/linux_is_fun/dir1
```

Removing a file

Command:

```
rm file2.txt
```

Moving back to where we were...

Directory navigation [4] Going back to previous directory and back

Command:

```
cd -

cd dir2
```

Expected final directory:

```
Linux: /home/<USERNAME>/linux_is_fun/dir1
MacOS: /Users/<USERNAME>/linux_is_fun/dir1
```

Removing a directory

Command:

```
rmdir dir2

rm -r dir2
```

Moving back to where we were...

Caution: No trash folder in linux. Deletion is usually permanent.

Shortcut to go to home directory

Command:

```
cd


cd ~
```

To:

```
Linux: /home/<USERNAME>/
MacOS: /Users/<USERNAME>/
```

Other useful commands

Commands:

```
cat

head

tail

wc

touch
```

Navigation Commands

Command	Description	Example
`pwd`	Print working directory	`pwd`
`ls`	List directory contents	`ls -la`
`cd`	Change directory	`cd Documents`
`cd ..`	Go up one level	`cd ..`
`cd ~`	Go to home directory	`cd ~`
`cd -`	Go to previous directory	`cd -`

ls Options

- `ls -l`: Long format (permissions, size, date)
- `ls -a`: Show hidden files (starting with .)
- `ls -h`: Human-readable sizes
- `ls -la`: Combine options

File Operations

Command	Description	Example
`mkdir`	Create directory	`mkdir data`
`touch`	Create empty file	`touch file.txt`
`cp`	Copy file/directory	`cp file.txt backup.txt`
`mv`	Move or rename	`mv old.txt new.txt`
`rm`	Remove file	`rm file.txt`
`rmdir`	Remove empty directory	`rmdir data`
`rm -r`	Remove directory + contents	`rm -r data/`

⚠ Important

`rm` is permanent! There is no trash/recycle bin. Be careful, especially with `rm -r`

Viewing File Contents

Command	Description	Example
`cat`	Display entire file	`cat file.txt`
`head`	First lines (default 10)	`head -n 20 file.txt`
`tail`	Last lines	`tail -n 20 file.txt`
`less`	Page through file	`less file.txt`
`wc`	Count lines/words/chars	`wc -l file.txt`

Navigating in less

- Space/Page Down: Next page
- b/Page Up: Previous page
- /pattern: Search forward
- q: Quit

Pipes and Redirection

Combine commands and control output

Symbol	Meaning
|	Pipe output to next command
>	Redirect output to file (overwrite)
>>	Append output to file
<	Read input from file

Wildcards and Patterns

Match multiple files at once

Pattern	Matches
*	Any characters
?	Any single character
[abc]	Any of a, b, or c
[0-9]	Any digit

Common Mistakes to Avoid

File Names

- Avoid spaces (use _ or -)
- Avoid special characters
- Case matters! (file.txt $\neq$ File.txt)

Paths

- Check you're in the right directory
- Use tab completion
- Use `ls` to verify files exist

Dangerous Commands

- `rm -rf /` — DON'T!
- `rm *` — Be careful
- Always double-check before removing

⚠ Important

When in doubt, use `ls` first to see what will be affected!

Genomic File Formats

Day 1, Session 5

KAUST Bioinformatics Platform

King Abdullah University of Science Technology

Feb 2026

Session Objectives

- Understand common genomic file formats
- Read and interpret FASTA, FASTQ, BAM, VCF, GTF, and BED files
- Know which tools work with which formats
- Recognize format-specific conventions and gotchas

Why File Formats Matter

The Challenge

- Different tools expect different formats
- Need to store sequences + metadata
- Must be readable by computers AND humans
- File sizes can be huge (TB for genomes)

Formats We'll Cover

- **FASTA**: Reference sequences
- **FASTQ**: Raw sequencing reads
- **SAM/BAM**: Aligned reads
- **VCF**: Genetic variants
- **GTF/GFF**: Gene annotations
- **BED**: Genomic regions

Key Concept

Understanding file formats is fundamental — you'll work with these every day in bioinformatics!

FASTA Format

Reference sequences and assemblies

FASTA Format

The most common sequence format — simple and universal

```
>NM_000546.6 Homo sapiens tumor protein p53 (TP53), mRNA
GATGGGATTGGGGTTTTCCCCTCCCATGTGCTCAAGACTGGCGCTAAAAG
TTTTGAGCTTCTCAAAAGTCTAGAGCCACCGTCCAGGGAGCAGGTAGCTG
CTGGGCTCCGGGGACACTTTGCGTTCGGGCTGGGAGCGTGCTTTCCACGA
>sp|P04637|P53_HUMAN Cellular tumor antigen p53
MEEPQSDPSVEPPLSQETFSDLWKLLPENNVLSPLPSQAMDDLMLSPDDI
EQWFTEDPGPDEAPRMPEAAPPVAPAPAAPTPAAPAPAPSWPLSSSVPSQ
```

Structure

- `>` Header line (description)
- Sequence on following lines
- Multiple sequences allowed

File Extensions

- `.fasta`, `.fa`
- `.fna` (nucleotide)
- `.faa` (amino acid)

FASTA Header Conventions

Headers contain metadata, but format varies by source

```
>gi|8393948|ref|NM_000546.2| Homo sapiens p53, mRNA
>sp|P04637|P53_HUMAN Cellular tumor antigen p53 OS=Homo sapiens
>ENST00000269305.9 TP53-201 gene:ENSG00000141510.18
>chr1:1000-2000 extracted from hg38
>my_sequence description goes here
```

Common Patterns

- Accession number first
- Pipe-separated fields
- Species information
- Coordinates

⚠ Important

The sequence ID ends at the first space. Everything after is the description.

FASTQ Format

Raw sequencing reads with quality scores

FASTQ Format

FASTA + Quality scores — standard for sequencing data

```
@SEQ_ID_1 description
GATTTGGGGTTCAAAGCAGTATCGATCAAATAGTAAATCCATTTGTTCAACTCACAGTTT
+
!''*((((***+))%%%++)(%%%%).1***-+*''))**55CCF>>>>>>CCCCCCC65
@SEQ_ID_2
GCTAGCTACGATCGATCGATCGATCGATCGATCGATCGATCGATCGATCGATCGATCGAT
+
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
```

Four lines per read:

1. `@` Header line (read ID)
2. Sequence
3. + Separator (may repeat header)
4. Quality scores (ASCII-encoded)

Understanding Quality Scores

Phred Quality Score

$$Q = -10 \times \log_{10}(P)$$

where P = probability of error

Q	Error	Accuracy
10	1 in 10	90%
20	1 in 100	99%
30	1 in 1000	99.9%
40	1 in 10000	99.99%

ASCII Encoding (Phred+33)

- Character = Q + 33
- `!` = Q0 (33 in ASCII)
- `I` = Q40 (73 in ASCII)

Quality Interpretation

- `!` to `5`: Very poor
- `5` to `?`: Poor
- `?` to `I`: Good
- `I+`: Excellent

Key Concept

Q30 or higher is generally considered good quality for Illumina data.

FASTQ Naming Conventions

Illumina Read IDs contain metadata

```
@A00182:842:HVNLJDRX2:1:1101:1234:1000 1:N:0:ATCACG
```

Field	Meaning
A00182	Instrument name
842	Run number
HVNLJDRX2	Flowcell ID
1	Lane number
1101	Tile number
1234:1000	X:Y coordinates
1:N:0:ATCACG	Read number:filtered:control:index

Paired-end files: `sample_R1.fastq.gz` and `sample_R2.fastq.gz`

SAM/BAM Format

Aligned sequence reads

SAM/BAM Format Overview

Standard format for sequence alignments to a reference

```
@HD  VN:1.6   SO:coordinate
@SQ  SN:chr1 LN:248956422
@SQ  SN:chr2 LN:242193529
@RG  ID:sample1   SM:sample1   PL:ILLUMINA
read001   99    chr1   10000   60   100M     =   10200   300   AGCT...   IIII...   MD:Z:100
read001   147   chr1   10200   60   100M     =   10000   -300  TCGA...   IIII...   MD:Z:100
```

SAM = Text format

- Human-readable
- Large file sizes
- Header (@) + alignments

BAM = Binary format

- Compressed SAM
- 5-10x smaller
- Requires samtools to view

SAM Alignment Fields

Col	Field	Description	Example
1	QNAME	Read name	read001
2	FLAG	Bitwise flag	99
3	RNAME	Reference name	chr1
4	POS	Position (1-based)	10000
5	MAPQ	Mapping quality	60
6	CIGAR	Alignment description	100M
7	RNEXT	Mate reference	=
8	PNEXT	Mate position	10200
9	TLEN	Template length	300
10	SEQ	Sequence	AGCT...
11	QUAL	Quality scores	IIII...
12+	TAGS	Optional fields	MD:Z:100

Understanding SAM Flags

FLAG field encodes read properties as bits

Bit	Value	Meaning
0x1	1	Paired-end read
0x2	2	Proper pair (both mapped correctly)
0x4	4	Read unmapped
0x8	8	Mate unmapped
0x10	16	Read on reverse strand
0x20	32	Mate on reverse strand
0x40	64	First in pair (R1)
0x80	128	Second in pair (R2)
0x100	256	Secondary alignment
0x400	1024	PCR duplicate

Example: FLAG=99 = 64+32+2+1 = paired, proper pair, mate reverse, first read
Tool: `https://broadinstitute.github.io/picard/explain-flags.html`

CIGAR Strings

Describes how the read aligns to reference

Op	Meaning
M	Match or mismatch
I	Insertion to reference
D	Deletion from reference
N	Skipped region (splicing)
S	Soft clipping (bases present)
H	Hard clipping (bases absent)

Examples:

- `100M`: 100 bases aligned
- `50M2I48M`: 50 match, 2bp insertion, 48 match
- `30M1000N70M`: Spliced alignment (intron)
- `5S95M`: 5bp soft-clipped at start

Working with BAM Files

samtools: The essential BAM toolkit

```
# View BAM as text
samtools view sample.bam | head

# View with header
samtools view -h sample.bam | head

# Get alignment statistics
samtools flagstat sample.bam

# Index BAM file (required for random access)
samtools index sample.bam

# Extract region
samtools view sample.bam chr1:1000-2000

# Count reads
samtools view -c sample.bam
```

VCF Format

Variant Call Format

VCF Format

Standard format for genetic variants

```
##fileformat=VCFv4.2
##INFO=<ID=DP,Number=1,Type=Integer,Description="Total Depth">
##FORMAT=<ID=GT,Number=1,Type=String,Description="Genotype">
##FORMAT=<ID=DP,Number=1,Type=Integer,Description="Read Depth">
#CHROM  POS      ID          REF ALT QUAL FILTER INFO        FORMAT  SAMPLE1
chr1    10177    rs367896724 A   AC  100  PASS   DP=50       GT:DP   0/1:25
chr1    10352    rs555500075 T   TA  100  PASS   DP=45       GT:DP   1/1:20
chr7    55259515 rs121434568 T   G   100  PASS   DP=100      GT:DP   0/1:50
```

Sections:

- **Meta-information** (##): Format definitions
- **Header** (#): Column names
- **Data**: One variant per line

VCF Fields Explained

Field	Description	Example
CHROM	Chromosome	chr1
POS	Position (1-based)	10177
ID	Variant identifier	rs367896724
REF	Reference allele	A
ALT	Alternate allele(s)	AC
QUAL	Quality score	100
FILTER	Filter status	PASS
INFO	Variant annotations	DP=50
FORMAT	Sample field format	GT:DP
SAMPLE	Sample genotype data	0/1:25

Key Concept

Genotypes: 0/0 = homozygous reference, 0/1 = heterozygous, 1/1 = homozygous alternate

Working with VCF Files

bcftools: VCF manipulation toolkit

```
# View VCF
bcftools view variants.vcf.gz | head

# Get variant statistics
bcftools stats variants.vcf.gz

# Filter variants (PASS only, QUAL>30)
bcftools filter -i 'FILTER="PASS" && QUAL>30' variants.vcf.gz

# Extract region
bcftools view -r chr1:1000-2000 variants.vcf.gz

# Extract specific samples
bcftools view -s sample1,sample2 variants.vcf.gz

# Count variants
bcftools view -H variants.vcf.gz | wc -l
```

GTF/GFF Format

Gene annotations

GTF/GFF Format

Tab-separated format for genomic features

```
##gff-version 3
chr1   HAVANA   gene         11869   14409   .   +   .   gene_id "ENSG00000223972"; gene_name "DDX11L1";
chr1   HAVANA   transcript   11869   14409   .   +   .   gene_id "ENSG00000223972"; transcript_id "ENST00000456328";
chr1   HAVANA   exon         11869   12227   .   +   .   gene_id "ENSG00000223972"; exon_number "1";
chr1   HAVANA   exon         12613   12721   .   +   .   gene_id "ENSG00000223972"; exon_number "2";
chr1   HAVANA   CDS          12010   12057   .   +   0   gene_id "ENSG00000223972"; protein_id "ENSP00000450983";
```

Nine Columns:

1. Chromosome   2. Source   3. Feature type

2. Start (1-based)   5. End   6. Score

3. Strand (+/-)   8. Phase (CDS)   9. Attributes

GTF vs GFF3

GTF (Gene Transfer Format)

- Used by Ensembl, GENCODE
- Attributes: key "value";
- Required: gene_id, transcript_id
- Common for RNA-seq tools

GFF3

- More general format
- Attributes: key=value;
- Parent-child relationships
- Used by NCBI, many organisms

Important

GTF and GFF3 look similar but have different attribute formats! Check which your tools expect.

BED Format

Genomic intervals and regions

BED Format

Simple format for genomic intervals

```
chr1    11868    14409    DDX11L1    0    +
chr1    14403    29570    WASH7P     0    -
chr1    17368    17436    MIR6859-1  0    -
chr7    55019017  55211628  EGFR    0    +
```

Basic BED (3 columns)

1. Chromosome
2. Start (0-based!)
3. End

Extended BED (6+ columns)

4. Name
5. Score (0-1000)
6. Strand (+/-)

Important

BED uses 0-based, half-open coordinates! Position 1 in 1-based = position 0 in BED. The interval [0, 100) includes bases 0-99.

Working with BED Files

bedtools: Swiss army knife for genomic intervals

```
# Find overlapping regions
bedtools intersect -a peaks.bed -b genes.bed

# Regions in A but not B
bedtools subtract -a regions.bed -b exclude.bed

# Merge overlapping intervals
bedtools merge -i regions.bed

# Get sequences for regions
bedtools getfasta -fi genome.fa -bed regions.bed

# Compute coverage
bedtools coverage -a genes.bed -b reads.bam

# Closest feature
bedtools closest -a peaks.bed -b genes.bed
```

Coordinate Systems: The Eternal Confusion

Region: bases 3-6

Sequence:	A	T	C	G	A	T	C	G	A	T
1-based:	1	2	3	4	5	6	7	8	9	10
0-based:	0	1	2	3	4	5	6	7	8	9

Format	**System**	**Region 3-6**
SAM, VCF, GTF, GFF	1-based, closed	3-6
BED, BAM (internal)	0-based, half-open	2-6

Format Summary

Format	Content	Coords	Tool	Extension
FASTA	Sequences	N/A	seqkit	.fa, .fasta
FASTQ	Reads + quality	N/A	seqkit, fastp	.fq, .fastq
SAM/BAM	Alignments	1-based	samtools	.sam, .bam
VCF	Variants	1-based	bcftools	.vcf, .vcf.gz
GTF/GFF	Annotations	1-based	awk, grep	.gtf, .gff
BED	Intervals	0-based	bedtools	.bed

Key Concept

- Know your coordinate systems!
- Use appropriate tools for each format
- Most formats have compressed versions (.gz)
- Index files enable random access

Session Summary

Key Concept

Key File Formats

- **FASTA**: Reference sequences (simple, universal)
- **FASTQ**: Raw reads with quality scores
- **SAM/BAM**: Aligned reads (use samtools)
- **VCF**: Genetic variants (use bcftools)
- **GTF/GFF**: Gene annotations
- **BED**: Genomic intervals (use bedtools)

Remember: BED is 0-based, everything else is 1-based!

Next up: Hands-on with Genomic File Formats

Questions?

?