

Introduction to UNIX and bash shell

Jiawei Wang, PhD

Marie Curie Postdoctoral Fellow & EMBO Non-Stipendiary Fellow

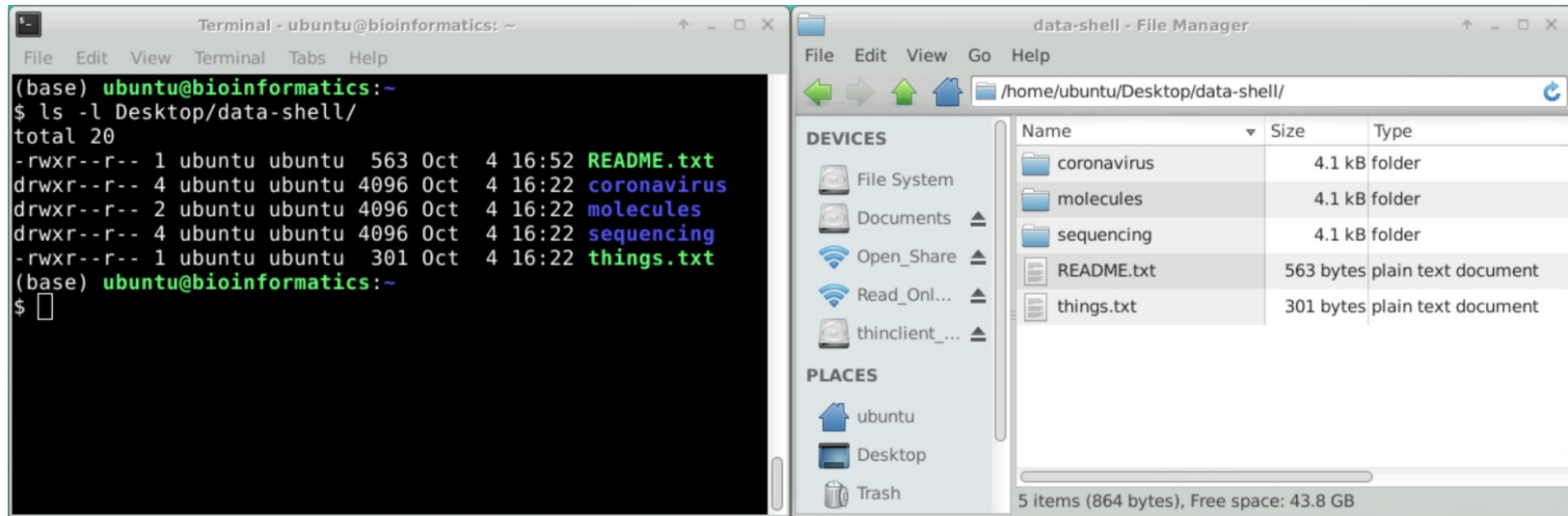
Finn & Marioni Groups

EMBL-EBI

jwang@ebi.ac.uk

2024/07/01

UNIX command line vs. Graphical user interface (GUI)



Unix command line

- Text-based interaction
- Powerful and flexible
- Scriptable
- Resource-Efficient

GUI

- Visual interaction
- User-friendly
- Less scriptable
- Resource-intensive

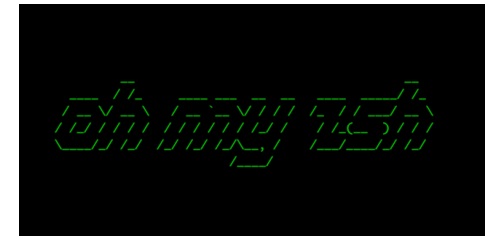
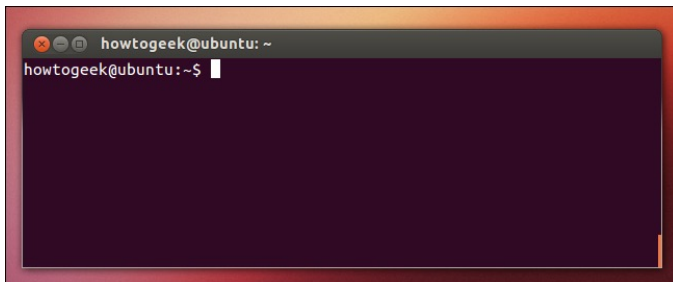
Nomenclature

- Unix - a family of operating systems that includes Linux and Mac OS
- Terminal - a program that allows interaction with the computer using text-based commands (i.e., a command-line interface)
- Unix Shell - the command-line interpreter that enables interaction with a Unix-like operating system
- Bash - a programming language used to interact with the Unix Shell

Don't worry too much about the terminology!

People interchangeably use terms like “shell”, “command line”, “bash”, “terminal”, “unix”, “linux”, “ubuntu” to mean broadly the same thing:

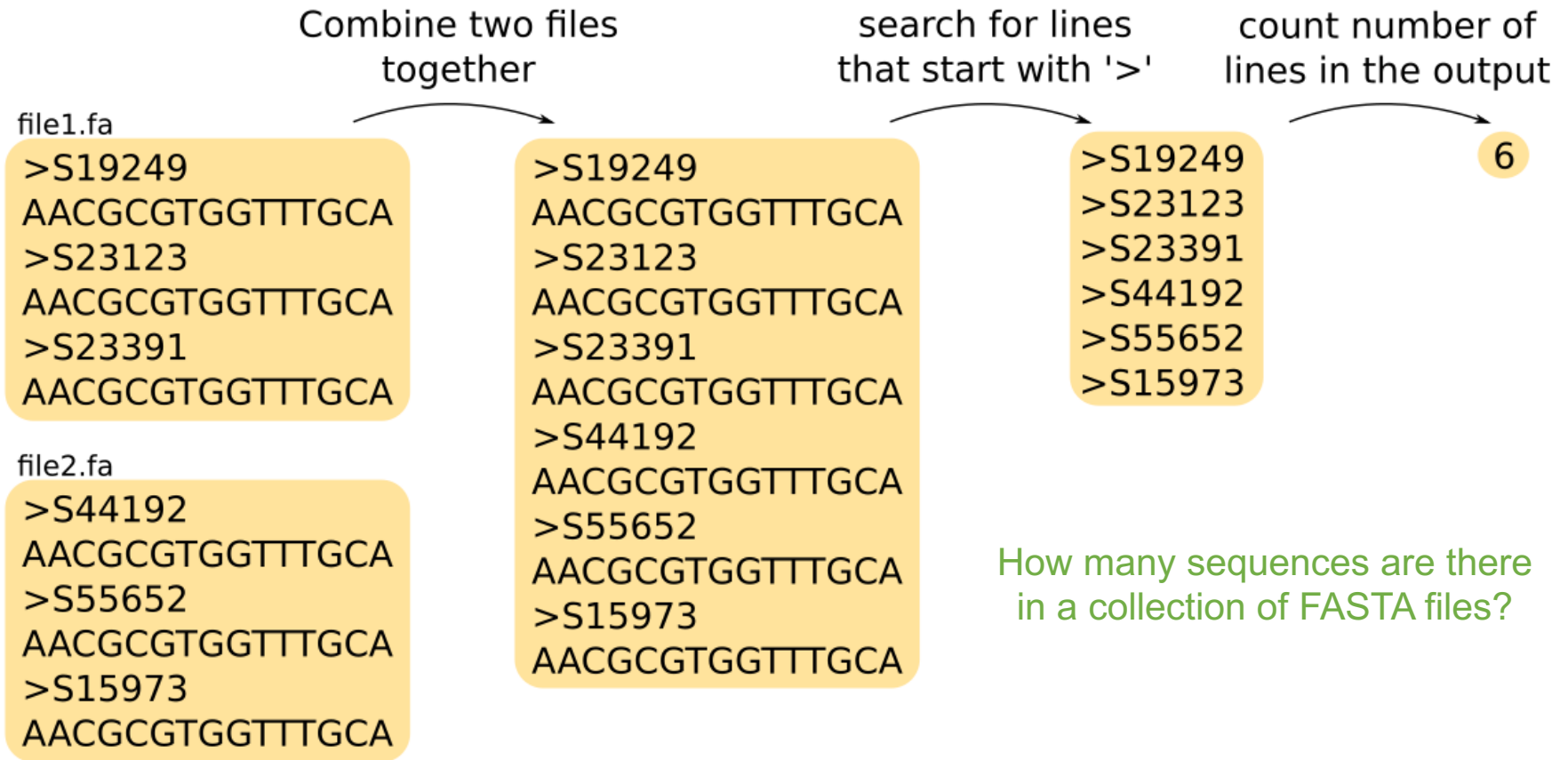
a (non-Windows) command line



Why do we need to learn UNIX and bash shell?

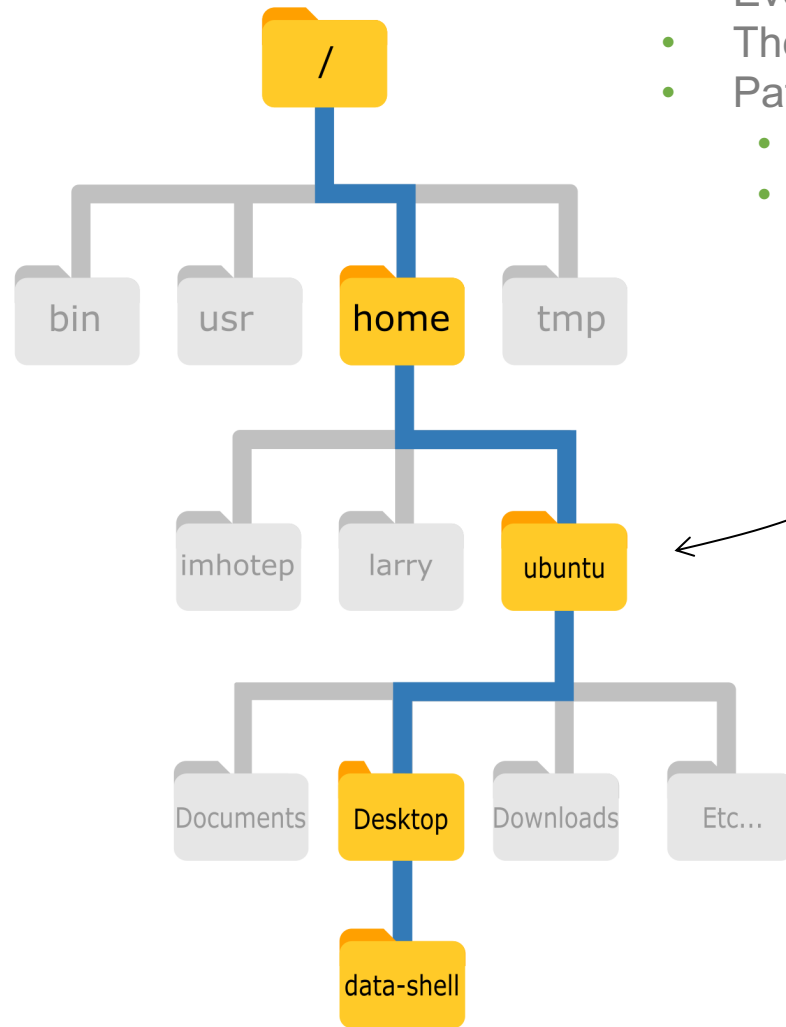
- Ubiquitous in computing applications and used in remote servers (like HPC)
- Most bioinformatics software is written for Linux
- Wide range of commands allow complex file manipulations, such as finding/replacing text patterns → very useful in bioinformatics
- Commands can be saved in script files → automation and reproducibility

Motivating example



```
cat *.fa | grep ">" | wc -l
```

Files, directories and paths




- Every user has a “home” directory
- The location of files or directories is specified with a **path**
- Paths can be **absolute** or **relative**
 - **Absolute:** `/home/ubuntu/Desktop/data-shell/`
 - **Relative:** `Desktop/data-shell/`

Current directory

Absolute paths always start with
/
“the root/start of the filesystem”

Terminal layout

```
robin@computer-name:~$ pwd  
/home/robin  
  
robin@computer-name:~$ cd Desktop  
  
robin@computer-name:~/Desktop$ cd Desktop  
  
robin@computer-name:~/Desktop$
```



pwd → “print working directory”
~ → “short for home directory”

Terminal layout

```
robin@computer-name:~$ pwd  
/home/robin
```



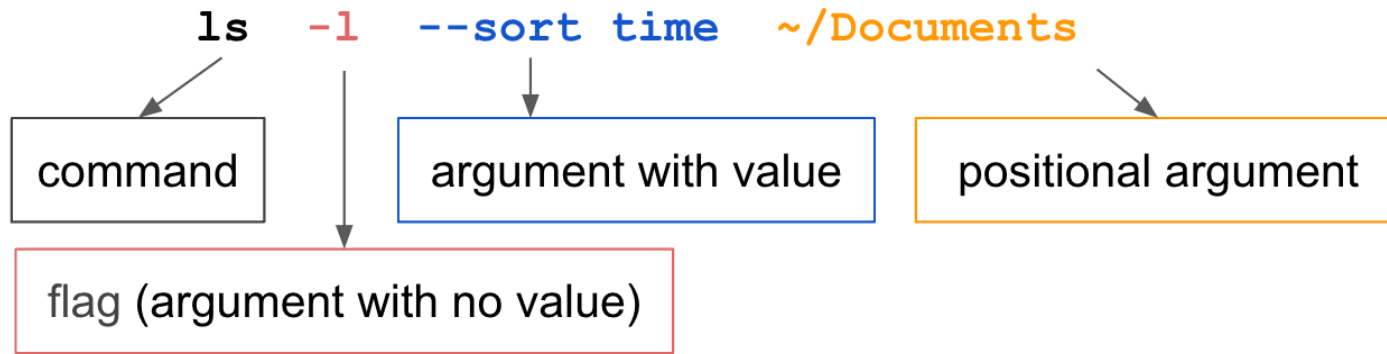
```
robin@computer-name:~$ cd Desktop
```

```
robin@computer-name:~/Desktop$ cd Desktop
```

```
robin@computer-name:~/Desktop$
```

cd → “change directory”

Basic command structure



List the files
in a long format
sorted by time of creation/modification
in the directory Documents

Documentation

man ls

NAME

ls - list directory contents

SYNOPSIS

ls [OPTION]... [FILE]...

DESCRIPTION

List information about the FILES (the current directory by default). Sort entries alphabetically if none of -cftuvSUX nor --sort is specified.

Mandatory arguments to long options are mandatory for short options too.

-a, --all
do not ignore entries starting with .

-A, --almost-all
do not list implied . and ..

--author
with -l, print the author of each file

...

-l use a long listing format

...

--sort=WORD
sort by WORD instead of name: none (-U), size (-S), time (-t),
version (-v), extension (-X)

Combining commands using pipe |

```
cat example.csv | cut -d "," -f 3
```

```
cat example.csv | cut -d "," -f 3 | sort
```

```
cat example.csv | cut -d "," -f 3 | sort | uniq
```

What is a “for loop”?

- A programming language technique which allows a piece of code to be repeated multiple times
- A “for loop” statement is usually called an “iterative statement”
- A typical example of a repetitive task is to read and process a list of files

```
$ bash count_atoms.sh cubane.pdb  
$ bash count_atoms.sh methane.pdb  
$ bash count_atoms.sh ethane.pdb  
$ bash count_atoms.sh pentane.pdb
```

```
for variable in item1 item2 item3  
do  
    some_command_using ${variable}  
done
```

What is a shell script?

```
count_input_loop.sh
```

```
#!/bin/bash
```

```
pattern=$1
```

```
for filename in example_sub1.csv example_sub2.csv example_sub3.csv
```

```
do
```

```
    bash count_input.sh $pattern $filename
```

```
done
```

You don't need to remember everything...



Bash & zsh Shell Terminal Basics

Learn Bash online at [www.DataCamp.com](https://www.datacamp.com)

> What are Bash & zsh Terminals?

Shell terminals, such as Bash and zsh, are text-based user interfaces for interacting with an operating system. They allow you to input commands through a command line, offering direct communication with the system for tasks like file manipulation, program execution, and system control. Bash is common on Linux systems and zsh is the default on MacOS systems.

Definitions

The **working directory** is the directory that commands are executed from. By default, commands will read and write files to this directory.

The **root directory** is the top of the file system. All other directories are contained within the hierarchy of this directory.

An **absolute path** starts from the root directory. Think of it like latitude and longitude - the values for a location don't change wherever you are.

A **relative path** starts from the working directory. Think of it like directions from where you are, like "20 kilometers West from here".

A **glob pattern** is a way of specifying multiple files at once.

A **regular expression** is a more complex way of specifying text fragments. Learn more in DataCamp's [Regular Expressions Cheat Sheet](#).

> Getting help

Display the manual for a command with `man`

```
man head
```

> File System Navigation

Print the current working directory with `pwd`

```
pwd
```

Change the current working directory with `cd`

```
cd data/raw # Go to raw dir inside data dir inside current dir
```

Absolute paths start with the root directory, /

```
cd /none
```

Relative paths can start with the current working directory, .

```
cd ../images
```

Move up to the parent directory with `..` (can be used repeatedly)

```
cd ../.. # Go to grandparent directory
```

List files and folders in the current working directory with `ls`

```
ls
```

List all files and folders, including hidden ones (names starting .) with `ls -a`

```
ls -a
```

List files and folders in a human-readable format with `ls -lh`

```
ls -lh
```

List files and folders matching a glob pattern with `ls` and pattern

```
ls *.csv # Returns all CSV files
```

Recursively list all files below the current working directory with `ls -R`

```
ls -R
```

List estimated disk usage of files and folders in a human-readable format with `du -ah`

```
du -ah
```

Find files by name in the current directory & its subdirectories with `find -type f -name pattern`

```
find . -type f -name *.ipynb # Find Jupyter notebooks
```

> Displaying Files

Display the whole file with `cat`

```
cat README.txt
```

Display a whole file, a page at a time with `less`

```
less README.txt
```

Display the first few lines of a file with `head`

```
head -n 10 filename sales.csv # Get first 10 lines of sales.csv
```

Display the last few lines of a file with `tail`

```
tail -n 10 filename sales.csv # Get last 10 lines of sales.csv
```

Display columns of a CSV file with `cut`

```
cut -d , -f 2-5, 8 sales.csv # Using comma delimiter, select fields 2 to 5 & 8
```

Display the lines of a file containing text matching a regular expression with `grep`

```
grep [0-9]+ sales.csv # Matches lines containing numbers
```

Display the names of files with filenames containing text matching a regular expression with `grep -r`

```
grep -r sales[0-9]+\.csv # Matches filenames with "sales", numbers, dot "csv"
```

Get the line word & character count of a file with `wc`

```
wc README.txt
```

> Copying, Moving and Removing Files

Copy (and paste) a file to a new directory with `cp`

```
cp sales.csv data/sales-2023.csv # Copy to data dir and rename
```

Copy files matching a glob pattern with `cp` pattern `newdir`

```
cp *.csv data/ # Copy all CSV files to data dir
```

Move (cut and paste) a file to a new directory with `mv`

```
mv sales.csv data/sales-2023.csv # Move to data dir and rename
```

Rename a file by moving it into the current directory

```
mv sales.csv sales-2023.csv
```

Move files matching a glob pattern with `mv` pattern `newdir`

```
mv *.csv data/ # Move all CSV files to data dir
```

Prevent overwriting existing files with `mv -n`

```
mv -n sales.csv data/sales-2023.csv # Rename unless new filename exists
```

Remove (delete) a file with `rm`

```
rm bad_data.json
```

Remove a directory with `rmdir`

```
rmdir temp_results
```

> Combining commands

Redirect the output from a command to a file with `>`

```
head -n 5 sales.csv > top_sales.csv
```

Pipe the output from a command to another command with `|`

```
head -n 5 sales.csv | tail -n 1
```

Redirect the input to a command with `<`

```
head -n 5 < sales.csv
```

> Glob Patterns

Match one or more character with `*`

```
*.txt # Match all txt files
```

Match a single character with `?`

```
sales202?.csv # Match this decade's sales data files
```

Match any character in the square brackets with `[...]`

```
head -n 5 < sales.csv
```

Match any patterns listed in the curly braces with `{...}`

```
{*.csv *.tsv} # Matches all CSV and TSV files
```

> Manipulating File Contents

Sort lines of a file with `sort`

```
sort random_order.txt
```

Sort in descending order using `sort -r`

```
sort -r random_order.txt
```

Combine cut and sort using a pipe to sort a column of a CSV file

```
cut -d , -f 2 | sort
```

Remove adjacent duplicate lines with `uniq`

```
uniq sales.csv
```

Get counts of (adjacent) duplicate lines with `uniq -c`

```
uniq -c sales.csv
```

Combine sort and uniq using a pipe to remove duplicate lines

```
sort random_order.txt | uniq
```

> Loops and Flow Control

Execute a command for multiple values with for variable in values; do command; done

```
datafiles=*.csv
for file in datafiles; do echo $file; done
```

Spread loops over multiple lines for increased readability

```
datafiles=*.csv
for file in datafiles
do
    echo $file
done
```

Conditionally execute code with `if [condn]; then command; else alt_command; fi`

```
x=99;
if [ $x > 50 ]; then
    echo "too high";
elif [ $x < 50 ]; then
    echo "too low";
else
    echo "spot on";
fi
```

> Reusing Commands

See your previous commands with history

```
history
```

Save commands in a shell file (extension .sh) and run them with bash or zsh

```
bash mycommands.sh
zsh mycommands.sh
```

Learn Power BI Online at
[www.DataCamp.com](https://www.datacamp.com)

datacamp

Reference

Most of the content presented here is based on the following references with a few modifications, from the [Bioinformatics Training Facility, University of Cambridge](#).

- https://docs.google.com/presentation/d/1q9Wy77wg_QrN5iP_wYD_TpX7iurs2eFYliOld1owj1A/edit#slide=id.g149f6c2cfa3_0_0
- <https://github.com/cambiotraining/unix-shell/tree/main/materials>