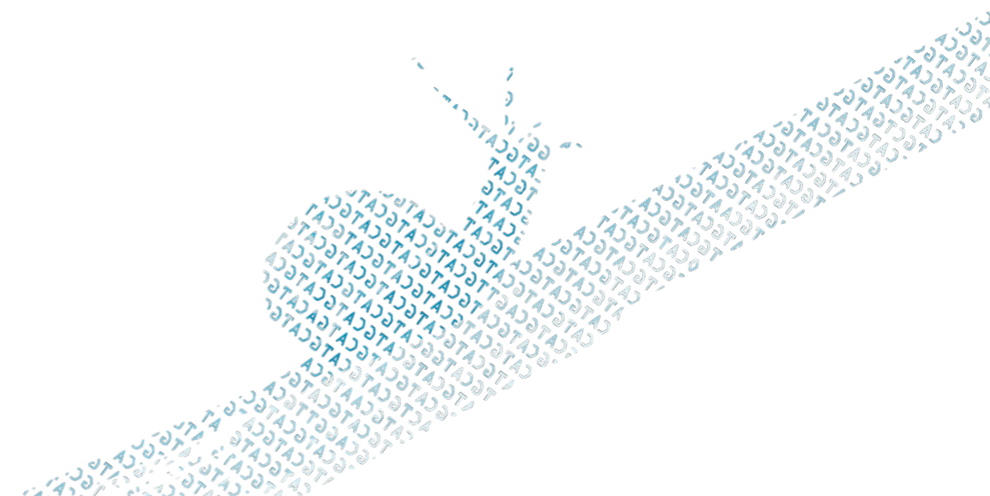


Terminal / Command Line

Tuesday, June 17, 2025



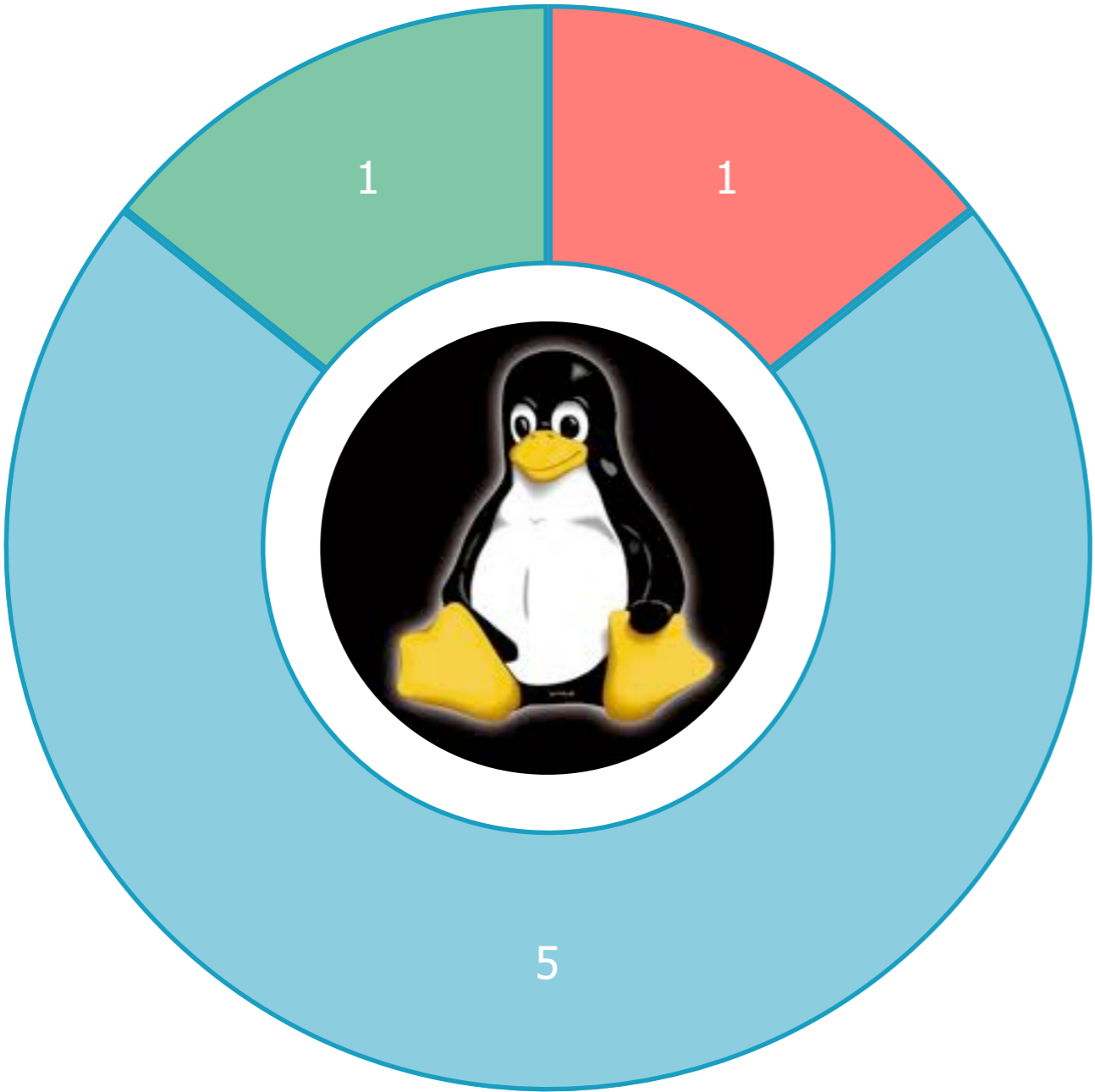
Learning Objectives

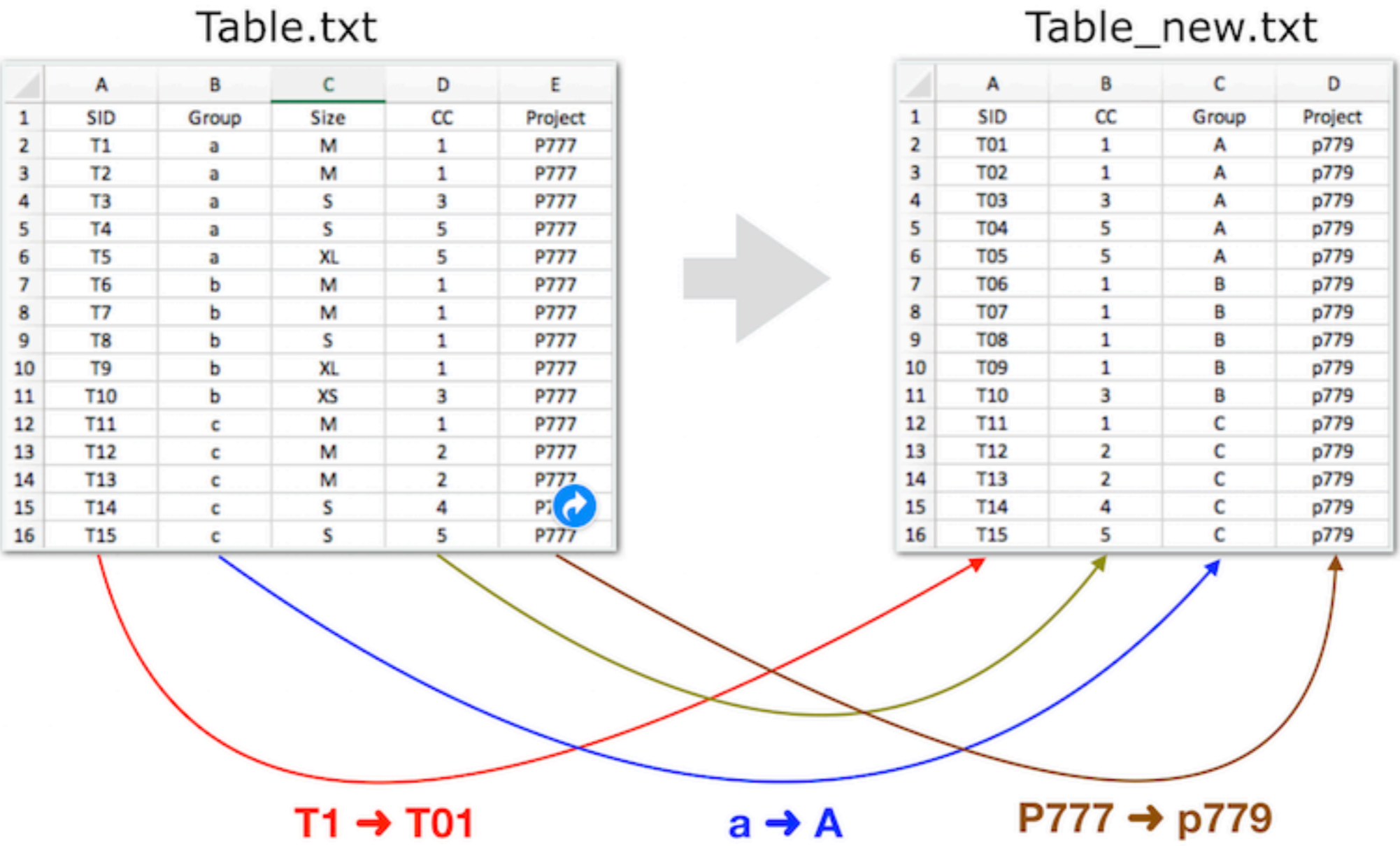
Main

- ◇ Understand the basic principles and differences between massive-parallel sequencing (MPS) platforms.
- ◇ Understand sample indexing and sequence (read) types.

Minor

- ◇ Understand the importance of a read data archive and be able to access such data.
- ◇ Understand the advantages, limitations and applications of MPS in e.g. genomics, transcriptomics or metabarcoding.





```
## Get Table
curl -O https://www.gdc-docs.ethz.ch/GeneticDiversityAnalysis/GDA/data/Table.txt

## Show Table
cat Table.txt
```

```
# SID Group Size CC Project
# T1 a M 1 P777
# T2 a M 1 P777
# T3 a S 3 P777
# T4 a S 5 P777
# T5 a XL 5 P777
# T6 b M 1 P777
# T7 b M 1 P777
# T8 b S 1 P777
# T9 b XL 1 P777
# T10 b XS 3 P777
# T11 c M 1 P777
# T12 c M 2 P777
# T13 c M 2 P777
# T14 c S 4 P777
# T15 c S 5 P777
```

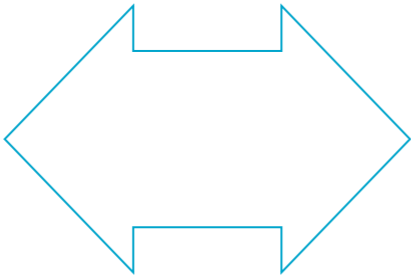


Table.txt

	A	B	C	D	E
1	SID	Group	Size	CC	Project
2	T1	a	M	1	P777
3	T2	a	M	1	P777
4	T3	a	S	3	P777
5	T4	a	S	5	P777
6	T5	a	XL	5	P777
7	T6	b	M	1	P777
8	T7	b	M	1	P777
9	T8	b	S	1	P777
10	T9	b	XL	1	P777
11	T10	b	XS	3	P777
12	T11	c	M	1	P777
13	T12	c	M	2	P777
14	T13	c	M	2	P777
15	T14	c	S	4	P777
16	T15	c	S	5	P777

```
## Transform Table
```

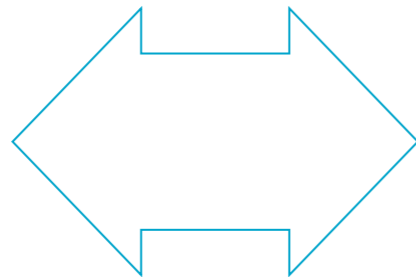
```
awk '{
    if(NR==1) print $1,$4,$2,$5;           # exclude header line
    else if(length($1)>2) print $1,$4,toupper($2),"p779"; # change project number
    else print "T0"substr($1,2,3),$4,toupper($2),"p779" # rename sample < 10
}' Table.txt > Table_new.txt
```

```
## Show NEW Table
```

```
cat Table_new.txt
```

```
# SID CC Group Project
```

```
# T01 1 A p779
# T02 1 A p779
# T03 3 A p779
# T04 5 A p779
# T05 5 A p779
# T06 1 B p779
# T07 1 B p779
# T08 1 B p779
# T09 1 B p779
# T10 3 B p779
# T11 1 C p779
# T12 2 C p779
# T13 2 C p779
# T14 4 C p779
# T15 5 C p779
```



Table_new.txt

	A	B	C	D
1	SID	CC	Group	Project
2	T01	1	A	p779
3	T02	1	A	p779
4	T03	3	A	p779
5	T04	5	A	p779
6	T05	5	A	p779
7	T06	1	B	p779
8	T07	1	B	p779
9	T08	1	B	p779
10	T09	1	B	p779
11	T10	3	B	p779
12	T11	1	C	p779
13	T12	2	C	p779
14	T13	2	C	p779
15	T14	4	C	p779
16	T15	5	C	p779

Alternative solution providing a bash script:

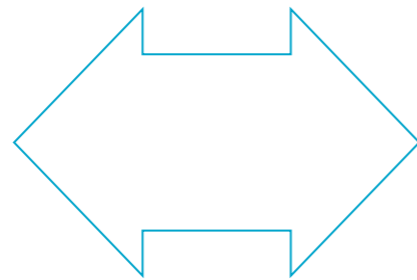
```
## Get Table
curl --verbose -O https://gdc-docs.ethz.ch/GeneticDiversityAnalysis/GDA/scripts/reformat_table.sh

# Make the script executable
chmod a+x reformat_table.sh

# run the script but make sure the Table.txt file is in the same directory
./reformat_table.sh Table.txt
```

```
## Show NEW Table
```

```
cat Table_new.txt
# SID CC Group Project
# T01 1 A p779
# T02 1 A p779
# T03 3 A p779
# T04 5 A p779
# T05 5 A p779
# T06 1 B p779
# T07 1 B p779
# T08 1 B p779
# T09 1 B p779
# T10 3 B p779
# T11 1 C p779
# T12 2 C p779
# T13 2 C p779
# T14 4 C p779
# T15 5 C p779
```



Table_new.txt

	A	B	C	D
1	SID	CC	Group	Project
2	T01	1	A	p779
3	T02	1	A	p779
4	T03	3	A	p779
5	T04	5	A	p779
6	T05	5	A	p779
7	T06	1	B	p779
8	T07	1	B	p779
9	T08	1	B	p779
10	T09	1	B	p779
11	T10	3	B	p779
12	T11	1	C	p779
13	T12	2	C	p779
14	T13	2	C	p779
15	T14	4	C	p779
16	T15	5	C	p779

Different OSs and Versions



Mac



Linux

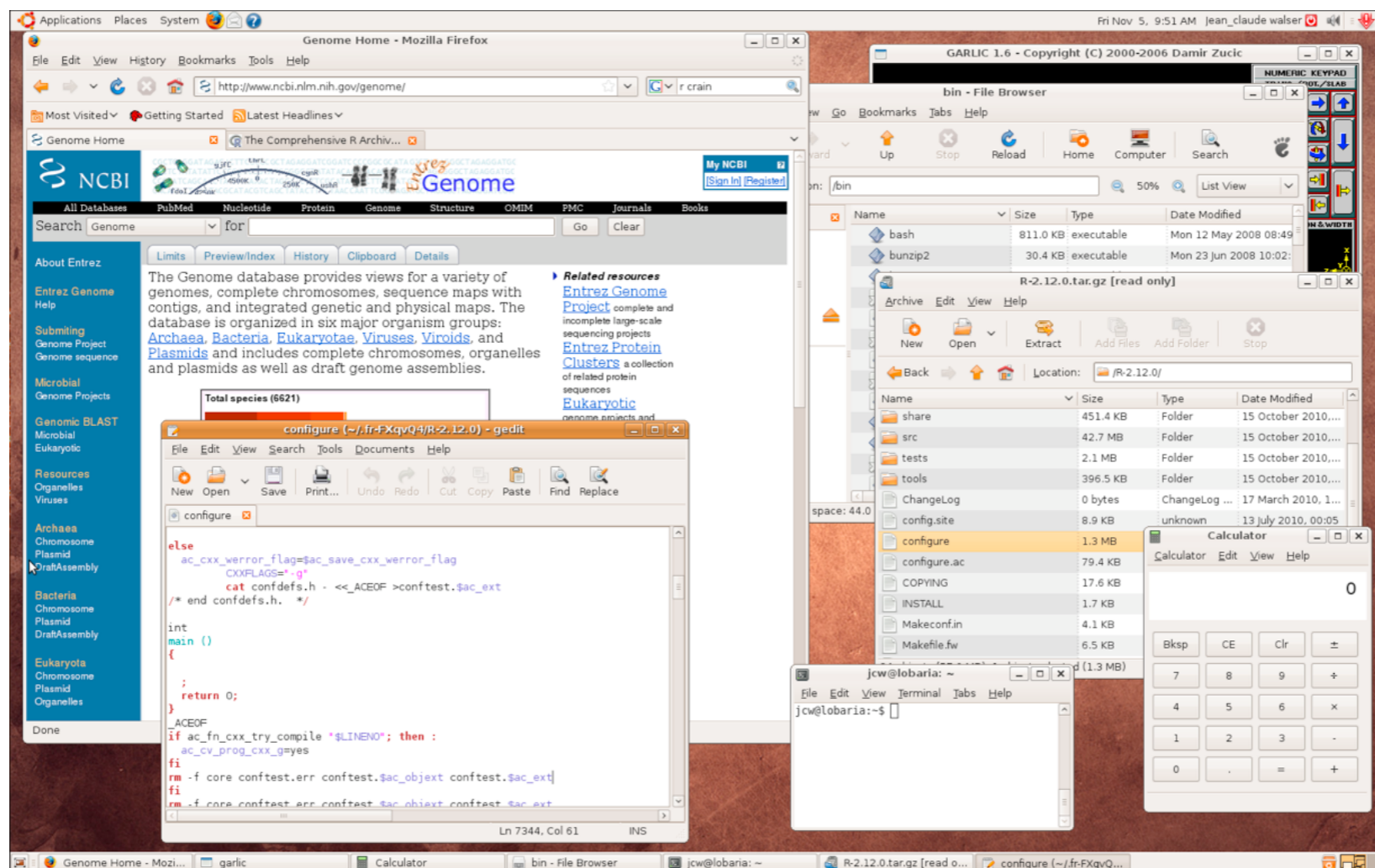


Windows



© Scott Adams, Inc./Dist. by UFS, Inc.

A **graphical user interface (GUI)** - often pronounced gooey - an interface that allows the user (you) to interact with programs in more ways than typing.



GUIs are nice but limited. Don't be afraid to use the terminal!

“Where there is a shell, there is a way.”

GUIs were introduced in reaction to the steep learning curve of **command-line interfaces (CLI)**, which require commands to be typed on the keyboard. Since the commands available in command line interfaces can be numerous, complicated operations can be completed using a short sequence of words and symbols. **This allows for greater efficiency, productivity once many commands are learned, and better reproducibility.**

Once upon a time, computers spoke only in cryptic spells typed by brave command-line wizards. These early users, armed with nothing but a blinking cursor and pure willpower, tamed machines by uttering arcane incantations like `grep`, `awk`, and `chmod 775`. It was powerful—but only if you had the patience of a monk and the memory of an elephant.

Then came the Graphical User Interface (GUI), a shiny new toy meant to rescue humanity from mistyping `rm -rf /` and accidentally deleting the universe. GUIs introduced buttons, windows, and icons - tools for the rest of us, who prefer dragging and clicking over memorizing which flag does what in a 12-word command.

Yet, ironically, while GUIs made computers more approachable, the command line didn't go away. Why? Because once you master its secret language, you can perform sorcery at lightning speed: batch-renaming files, deploying software, or bending entire servers to your will - often in just a single line. It's like riding a bicycle that can also fly, once you stop falling off.

So yes, GUIs are friendly. But CLIs? They're fast, efficient, and reproducible. Just... maybe keep a cheat sheet handy.

Shell - Terminal



Shell is a UNIX term for the interactive user interface with an operating system. The shell is the layer of programming that understands and executes the commands a user enters.

Bourne-Shell (sh)

Korn-Shell (ksh)

C-Shell (csh)

TC-Shell (tcsh)

Bourne-Again-Shell (bash)

Debian Almquist Shell (dash)

Z-Shell (zsh)

A-Shell (ash)

PowerShell / cmd.exe

What do I have?
\$> echo \${SHELL}

```
Bash > zcat file.gz
```

```
Zsh > zcat < file.gz
```

```
Bash > sed -i 's/d/D/2' file.txt
```

```
Zsh > sed -i"" "s/d/D/2" file.txt
```

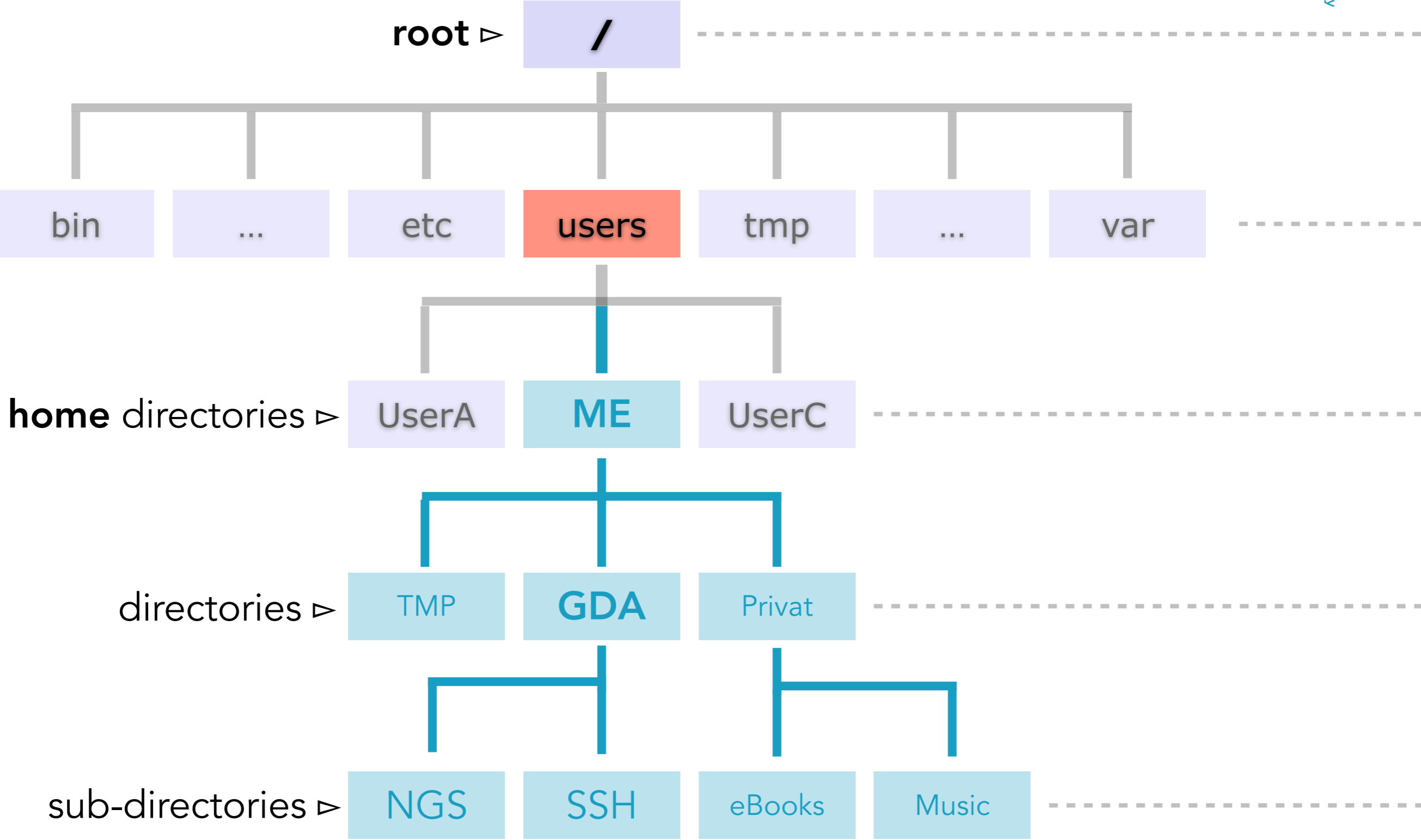
Change Shell

```
echo ${SHELL}
#which zsh
#which bash

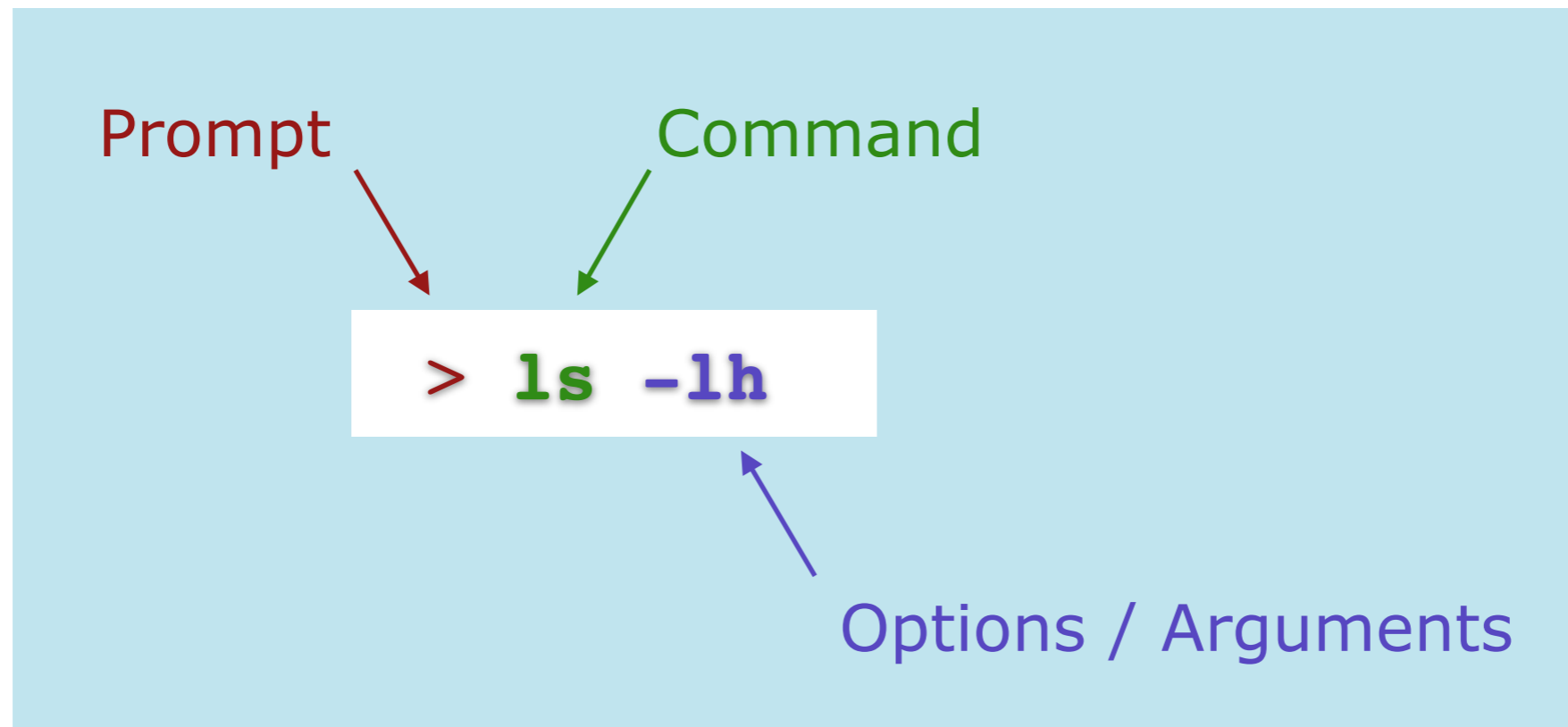
## Bash > Zsh
chsh -s /bin/bash

## Zsh > Bash
chsh -s /bin/zsh
```

Which one is better? Both shells will get the job done. The bash is a bit outdated (version 3.2 - 2006). Many of the conveniences provided by zsh can be made available in bash. It seems zsh is more helpful to newer shell user.



Command - Line



Built-in Help

```
> info <command>
```

```
> info ls
```

```
> man <command>
```

```
> man ls
```

* press **Q** to leave info or manual

JULIA EVANS

MEGA TERMINAL CHEAT SHEET

keyboard shortcuts

editing text (always works)
(almost)

backspace
Ctrl+W delete word
Ctrl+U delete line
(except in text editors)

editing text
(these often work in a
readline-like situation)

arrow keys
Ctrl+A or Home beginning of line
Ctrl+E or End end of line
Ctrl+arrow keys left/right a word
(or sometimes Alt+arrow keys
or Option+arrow keys
or Alt+b /Alt+f)

Ctrl+K delete forward
Ctrl+Y paste
Ctrl+H might help if
Backspace doesn't
work

also many shells have a "vi mode"
if that's your jam

in less

/ search
n/N next/prev match
space/PageDn scroll down

fixing broken output

fix messed up terminal

clear the screen:
clear

ask the terminal emulator &
OS to reset everything:
reset
(or just open a new tab!)

ways to quit

Ctrl+C quit (SIGINT)
Ctrl+Z stop process (SIGSTP)
(resume with fg or bg
or kill with kill)
Ctrl+D quit (in a REPL)
q quit (in a TUI)
Enter ~ exit frozen
SSH session
Ctrl+\ send SIGQUIT
(quit and core dump)

other useful stuff

Ctrl+L clear screen
Ctrl+R search history
Ctrl+Q unfreeze screen (that
you accidentally froze
with Ctrl+S)

copy and paste

in your terminal emulator,
it's usually:
Ctrl+Shift+C/V
or Cmd+C/V
if selecting text doesn't work,
try holding Shift

mouse stuff that
might work

Option+click place cursor
or Alt+click
scroll wheel scroll

force program not to
buffer its output

make its output a TTY with:
unbuffer CMD

core tasks

glob expansion

expand filenames with *
ls *.png
expand in all subdirectories:
wc **/*.txt
(not all shells support ** though)
include files starting with a dot:
ls .*

switch users

run a command as root with sudo
sudo cmd
or start a new shell as root:
su

change directories

go to home directory:
cd
go to prev directory:
cd -
go to /some/dir:
cd /some/dir

set environment variables

set a variable for one command:
env NAME=val cmd
NAME=val cmd
(they do almost the same
thing but I prefer the env
version because it behaves
the same in every shell)
set it for the whole session:
export NAME=val
(also what you put in your
bash/zsh config file to set
it permanently, in fish it's
set -x NAME val)
view environment:
env
search environment:
env | grep MAIL

pipes & redirects

redirect to a file:
cmd > file.txt

append to a file:
cmd >> file.txt

send a file to stdin:
cmd < file.txt

redirect stderr to a file:
cmd 2> file.txt

redirect stdout AND stderr:
cmd > file.txt 2>&1
-or-
cmd &> file.txt

pipe stdout:
cmd1 | cmd2

pipe stdout AND stderr:
cmd1 2>&1 | cmd2

managing processes

killing programs

send a SIGTERM:
kill PID
send a SIGKILL (can't be ignored):
kill -9 PID
kill all processes with python3
anywhere in its arguments:
pkill python3
kill all processes named python3:
killall python3
see what's using all your CPU:
top or htop
find a process's PID to kill it:
\$ ps aux | grep python3
bork(7213)... python3
the PID

job control

running in foreground
paused
running in background
list jobs:
jobs
stop a program from closing
when the terminal tab is closed:
nohup CMD
disown %1

& more

history

look at history:
history
search history:
Ctrl+R
rerun prev command as root:
sudo !!
rerun line 33 from history:
!33

add a directory to
your PATH

at the end:
export PATH=\$PATH:/my/dir
at the beginning:
export PATH=/my/dir:\$PATH
in fish the syntax is
different, like:
set PATH \$PATH /my/dir

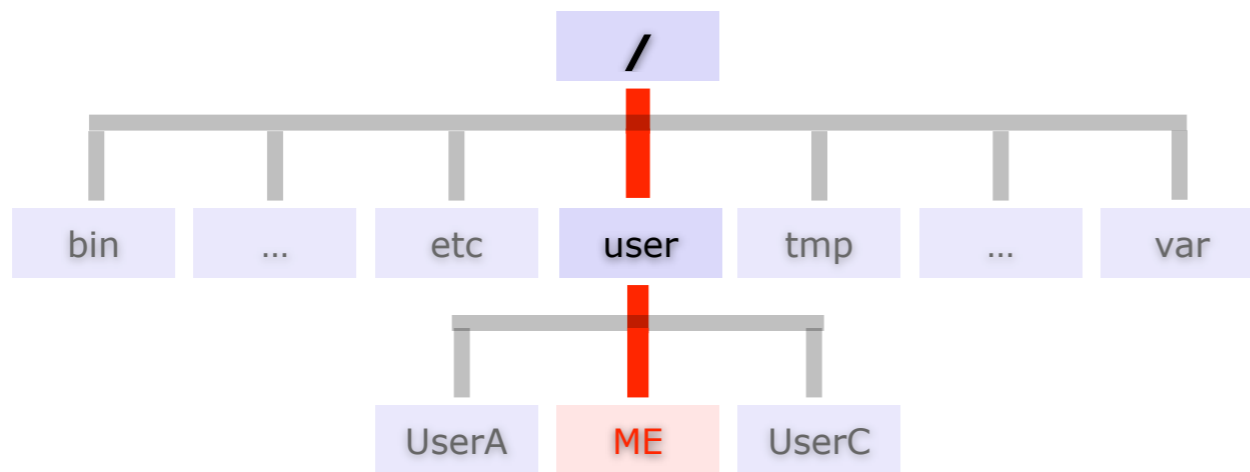
PATH debugging

show each PATH entry on its
own line:
echo "\$PATH" | tr ':' '\n'
find the first time a program
appears in PATH:
which CMD
find all instances of program
in PATH:
which -a CMD
ask your shell if CMD is a
function or alias or program:
type CMD

aliases

alias ll="ls -l"

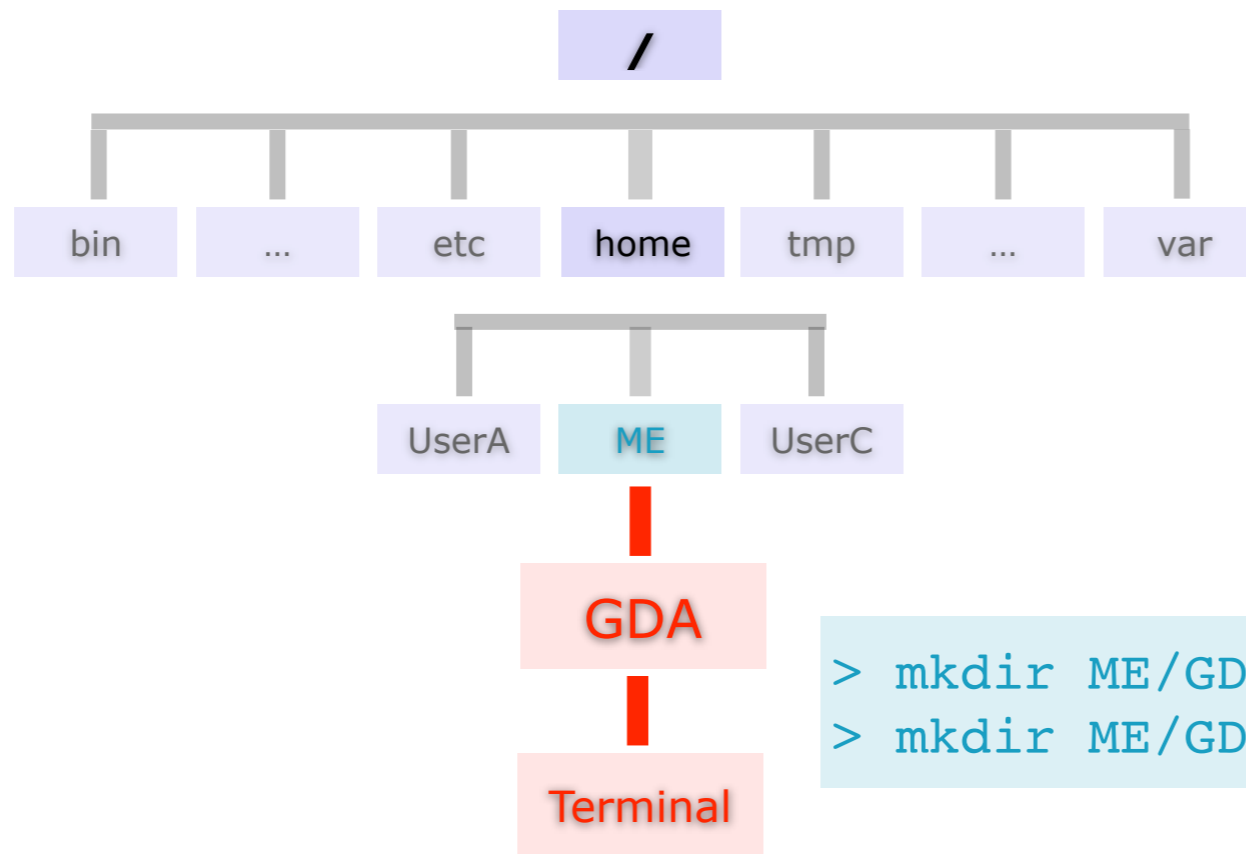
pwd - print name of current/working directory



```
> pwd  
/home/ME
```

```
> pwd  
/home/ME  
> cd ${HOME}  
> pwd  
/home/ME  
> cd ~  
> pwd  
/home/ME  
> cd  
/home/ME
```

mkdir - creating/making directories

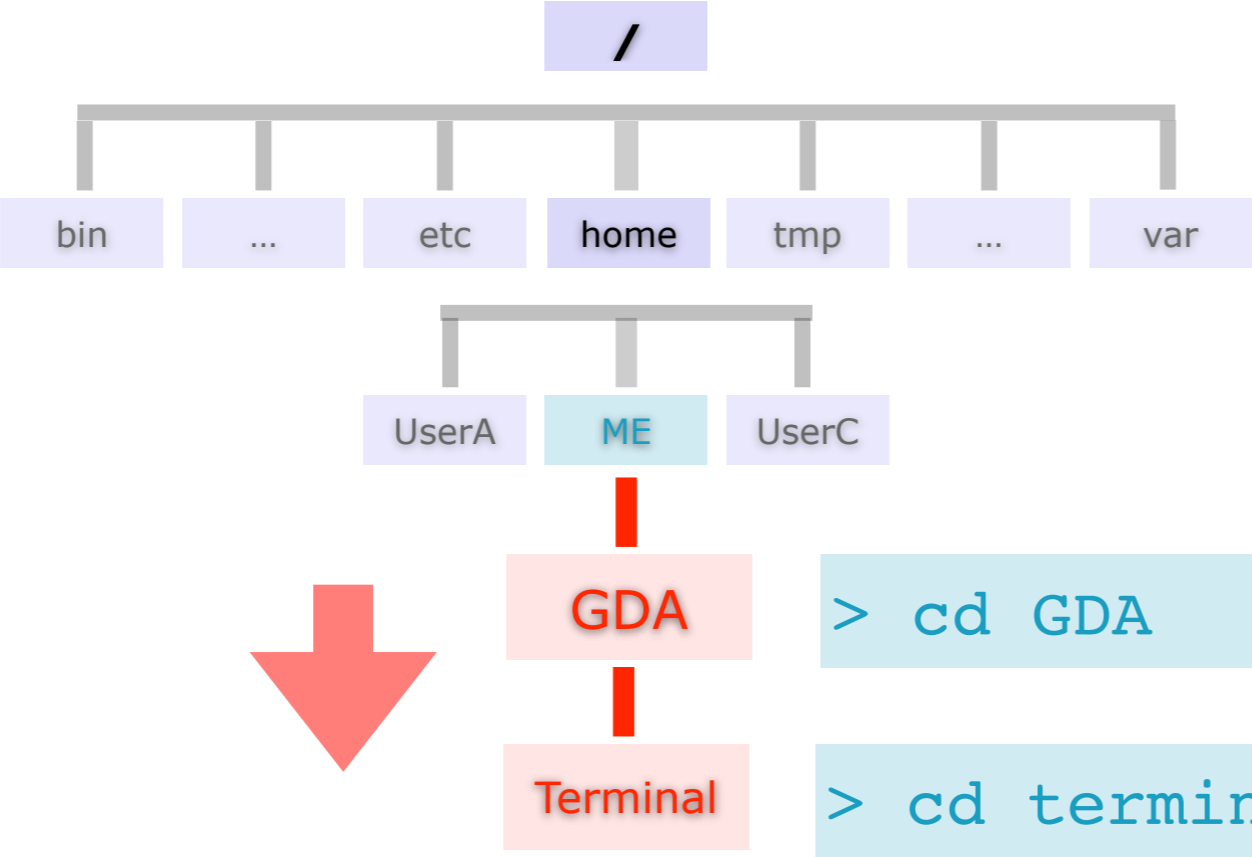


```
> mkdir ME/GDA  
> mkdir ME/GDA/Terminal
```

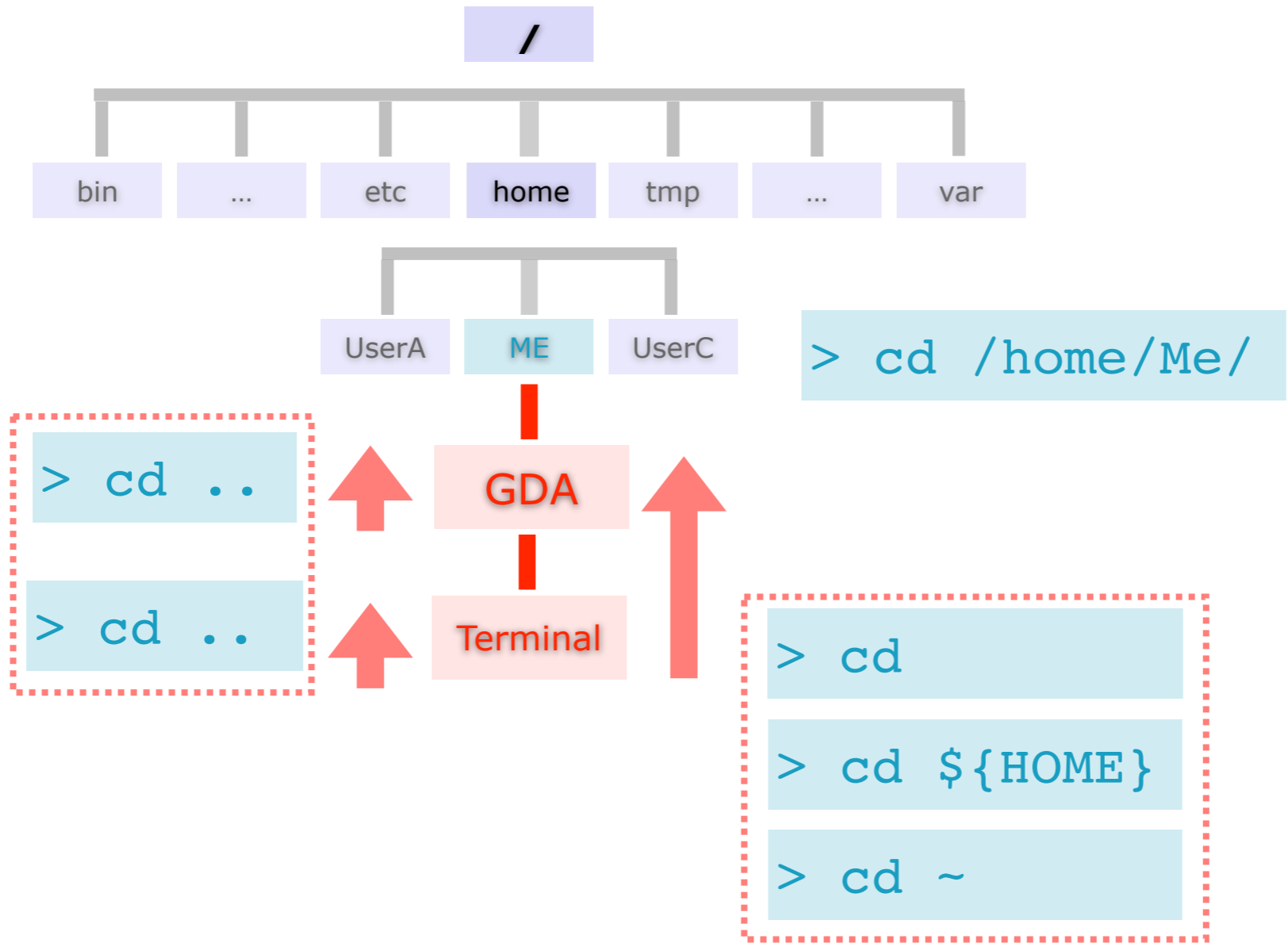
Alternatively, use option **p** to create directories and sub-directories

```
> mkdir -p ME/GDA/Terminal
```

cd - change directory



ways to go home





Local
Software
Management

Software
Dependencies

Parallel
Versions



Package, **dependency and environment management** for any language—Python, R, Ruby, Lua, Scala, Java, JavaScript, C/ C++, FORTRAN. Conda as a package manager helps you find and install packages. If you need a package that requires a different version of Python, you do not need to switch to a different environment manager, because conda is also an environment manager. With just a few commands, you can set up a totally separate environment to run that different version of Python, while continuing to run your usual version of Python in your normal environment.

BIOCONDA[®]

Bioconda is a **channel for the conda package manager** specialising in bioinformatics software. The conda package manager makes installing software a vastly more streamlined process. Conda is a combination of other package managers you may have encountered, such as pip, CPAN, CRAN, Bioconductor, apt-get, and homebrew. Conda is both language- and OS-agnostic, and can be used to install C/C++, Fortran, Go, R, Python, Java etc programs on Linux, Mac OSX, and Windows.

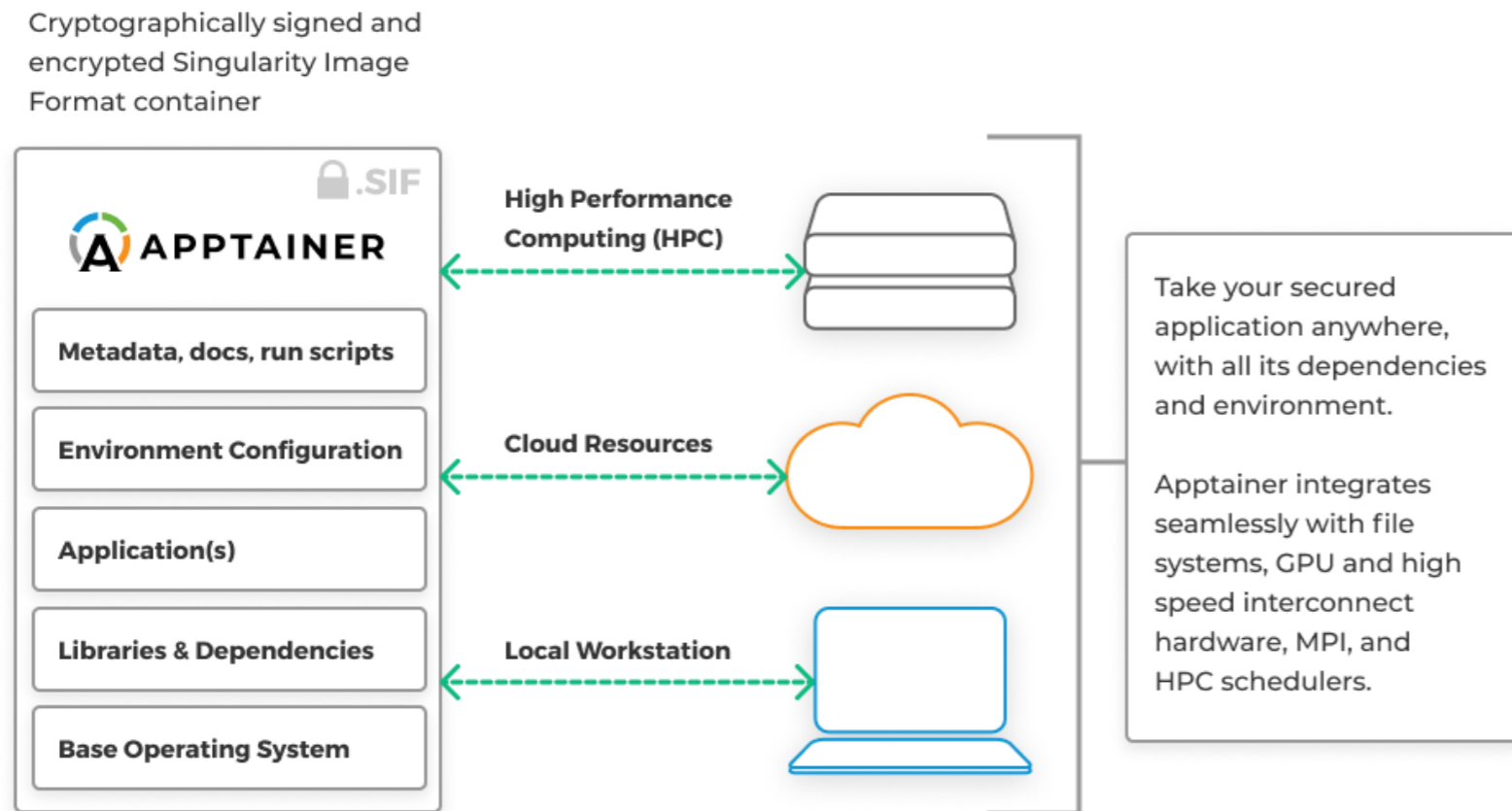
```
python --version
# Python 2.7.15
bwa
# -bash: bwa: command not found
blast -help
# -bash: blast: command not found
```

```
conda info --envs
source activate aligners
conda info --envs
python --version
# Python 3.6.7
bwa
blastn -help
```



1. **Portability:** Containers are portable and can be deployed across different systems without modifications. They encapsulate the application and its dependencies, ensuring consistent behaviour regardless of the underlying infrastructure.
2. **Scalability:** Containers enable applications to be easily scaled horizontally by running multiple instances of the same container across a cluster of machines. This allows for efficient resource utilisation and the ability to handle increased workload demands.
3. **Efficiency:** Containers share the host operating system's kernel and only require the necessary dependencies, making them lightweight and resource-efficient compared to traditional virtual machines.
4. **Isolation:** Containers provide a level of isolation between the application and the host operating system, as well as between different containers. This isolation helps prevent conflicts between applications and improves security by limiting the impact of any potential vulnerabilities.
5. **Rapid deployment:** Containers simplify the deployment process by providing a consistent environment. They can be quickly started, stopped, and updated, allowing for fast and efficient application deployment and rollbacks.

Containerisation is a technique used in software development and deployment that **allows applications and their dependencies to be packaged together into self-contained units called containers**. Containers provide a lightweight and isolated runtime environment that can run consistently across different computing environments, such as laptops, servers, or cloud platforms.



1. **Reproducibility:** Singularity enables researchers to create and distribute containers that encapsulate their entire software stack, including the application, libraries, and dependencies. This ensures that the environment remains consistent and reproducible across different systems.
2. **Compatibility with HPC:** Singularity is designed to seamlessly integrate with HPC clusters and resource managers commonly used in scientific computing. It allows users to leverage specialised hardware resources, such as GPUs and high-performance interconnects, while still providing isolation and security.
3. **Security and user isolation:** Singularity employs a security model that focuses on user isolation and containment. It ensures that users can run containers securely without requiring escalated privileges, making it suitable for multi-user HPC environments.
4. **Mobility:** Singularity containers are portable and can be easily shared, distributed, and run on different systems, including local workstations, clusters, and cloud platforms.
5. **Interoperability:** Singularity supports a wide range of container image formats, including Docker, making it possible to leverage existing container images and workflows.

Singularity (Apptainer) is an open-source **containerisation platform** that focuses on providing secure and reproducible environments for scientific and **high-performance computing (HPC)** workloads. It was developed specifically to address the challenges faced by researchers and scientists when working with complex software stacks and specialized computing resources.

EXTRAS

```
#!/usr/bin/perl -w
use strict;

my $input_fasta=$ARGV[0];
open(IN,"<$input_fasta") || die ("Error opening $input_fasta $!");

my $line = <IN>;
print $line;

while ($line = <IN>)
{
    chomp $line;
    if ($line=~m/^>gi/) { print "\n",$line,"\n"; }
    else { print $line; }
}

print "\n";
```

```
perl fa2one.pl multiple.fa > one.fa
```

```
#!/usr/bin/perl -w
use strict;

# Perl script to reformat fasta files
# Usage: perl fa2one.pl multiple.fa > one.fa

my $input_fasta=$ARGV[0];
open(IN,"<$input_fasta") || die ("Error opening $input_fasta $!");

my $line = <IN>;
print $line;

while ($line = <IN>)
{
    chomp $line;          ✓ limiting factor
    if ($line=~m/^\>gi/) { print "\n",$line,"\n"; }
    else { print $line; }
}

print "\n";
```

```
perl fa2one.pl multiple.fa > one.fa
```