

Introduction to UNIX

Georgette Femerling
UC Davis
09/28/2023



About me

- Mexican, born and raised in the Northeast of Mexico.
- **BSc in Genomic Sciences by the National Autonomous University of Mexico (UNAM)**
- Research Intern, Bioinformatician at the National Institute for Genomic Medicine at Mexico City
- Research Intern for a year at The Globe Institute in Copenhagen University
- MSc student in Human Genetics at McGill University
 - Visiting Researcher at UC Davis, visiting the Henn Lab
 - Been leading workshops for over a year as part of the MiCM

My research interest are focused on statistical and computational population genetics and method development.



Workshop outline

1

Introduction to the UNIX operating system
Files and processes
Directory Structure
The Terminal

2

Basic Commands
Directory Management
File Management
Redirecting output
File permissions

3

Text processing
Pipes

4

Variables
Connecting to a remote server
How to transfer files to and from a server

This is an interactive workshop!



- . If possible try to follow along in your own computer.
- . Feel free to interrupt or raise your hand to ask questions.
- . We can check error messages if I ask if there are questions or during the breaks
- . **I can also help/explain in Spanish, feel free to ask in Spanish if needed!**

Workshop material is available at:

<https://github.com/georgette-femerling/UCDavis-IntroUnix-Fall2023>

Please go to the page and download the material.

Part I: What is UNIX?

1

Introduction to the UNIX operating system
Files and processes
Directory Structure
The Terminal



Dennis Ritchie (standing) and Ken Thompson (seating) at Bell Labs, 1960's

UNIX Operating system

- Multi-user, multi-tasking and interactive system.

Characterized by:

- Resource-efficient
- Hierarchical file system
- Compatibility (files, devices, processes)
- Plain files to store data



Parts

The Kernel

- The core of the operating system
- Manages processes' time, memory and resources
- Handles file storage
- Responds to system calls

The shell

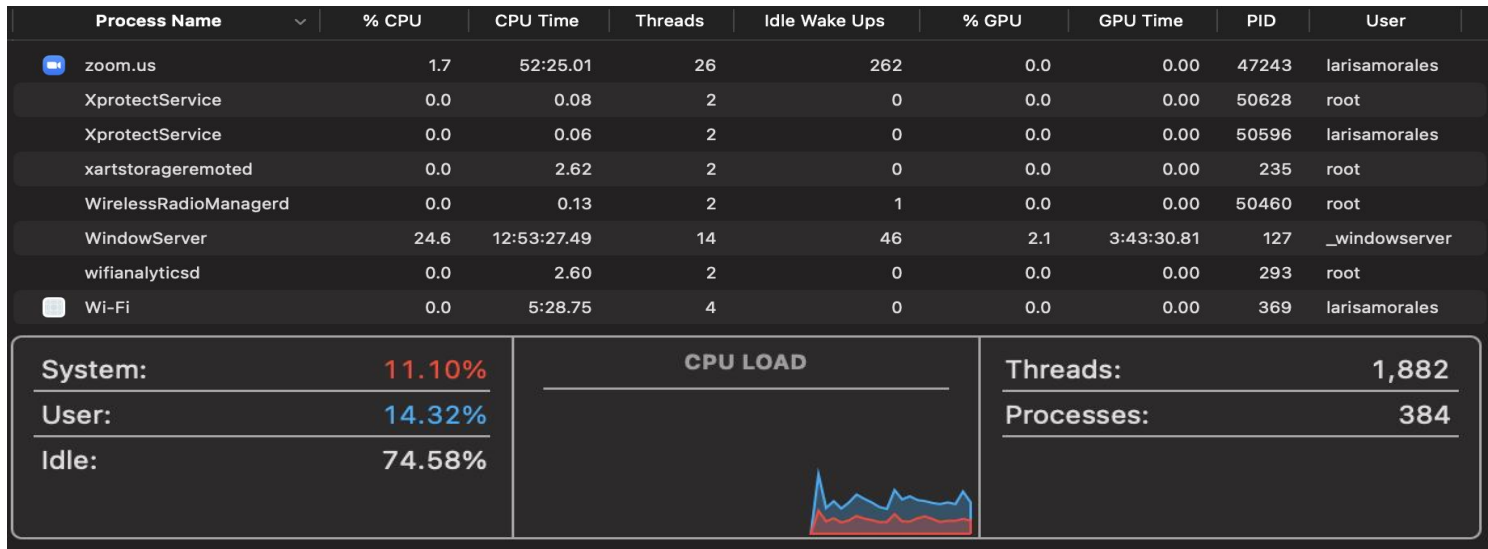
- Let's the user communicate with the kernel.
- Command line interpreter (CLI)
- Executes the instructions requested by the user (commands/programs)

The programs

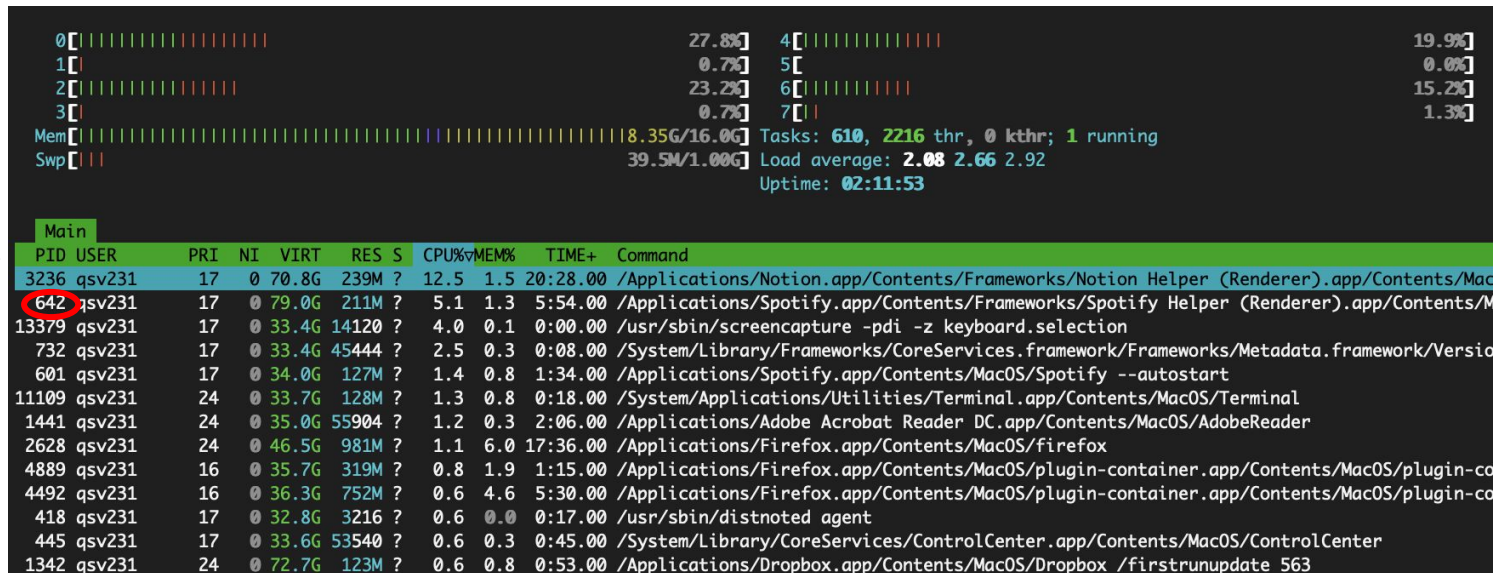
- Let's the user navigate the operating system
- And perform specific actions

**In UNIX
everything is a file or a
process!**







Processes



PID (process identifier)



Files

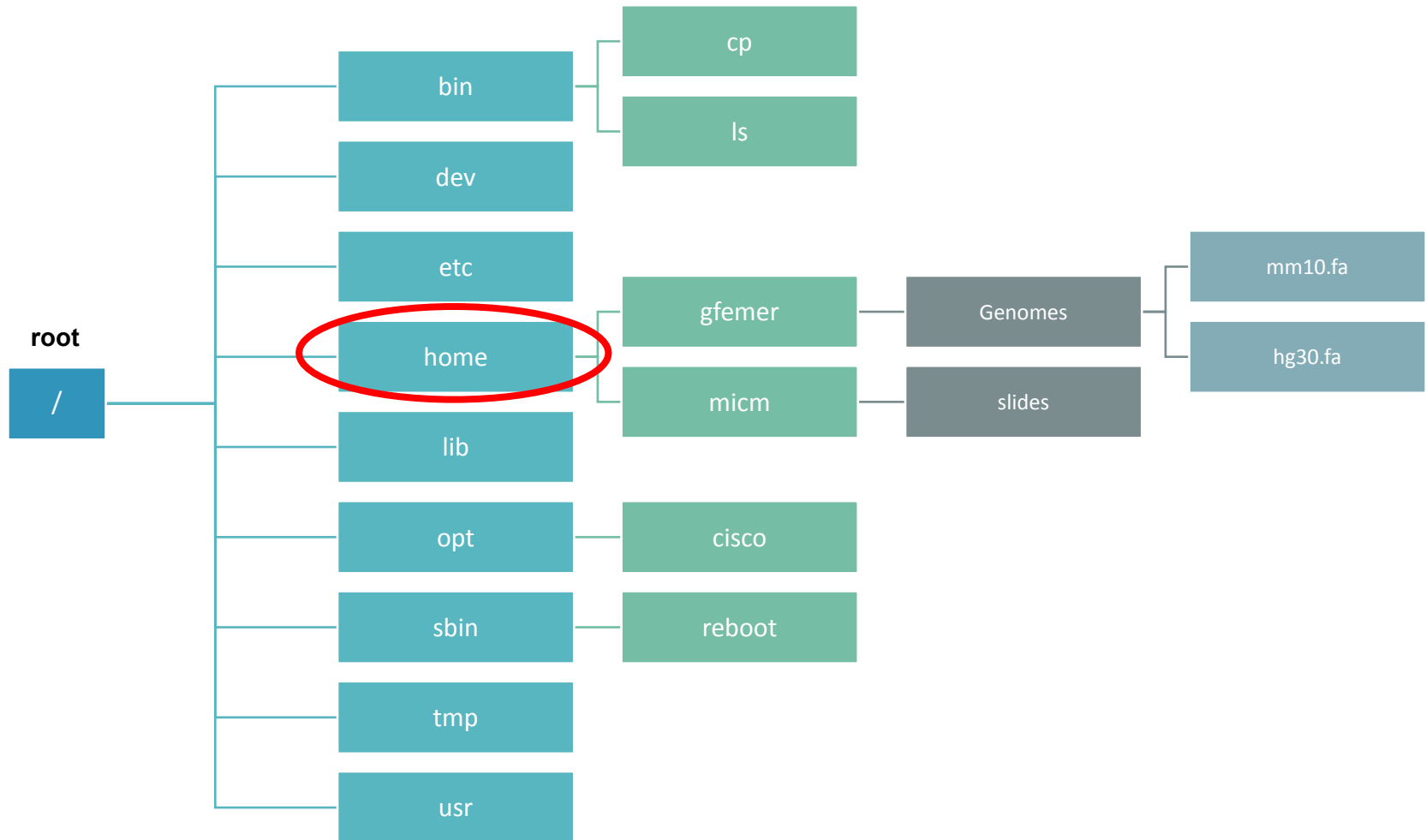
Name	^	Date Modified	Size	Kind
 SRR957824.bam		8 Jul 2022 at 11:58 AM	98,1 MB	BAM As...bly File
 SRR957824.fixmate.bam		8 Jul 2022 at 11:58 AM	102,4 MB	BAM As...bly File
 SRR957824.fixmate.sorted.bam		8 Jul 2022 at 11:58 AM	72,7 MB	BAM As...bly File
 SRR957824.markdup.bam		8 Jul 2022 at 11:58 AM	72,6 MB	BAM As...bly File
 SRR957824.markdup.bam.bai		8 Jul 2022 at 11:58 AM	17 KB	Document
 SRR957824.stats		4 Jul 2022 at 4:17 PM	102 KB	Document

```
total 675592
-rw-r--r--  1 qsv231  staff    101763 Jul  4 16:17 SRR957824.stats
-rw-r--r--  1 qsv231  staff   98077266 Jul  8 11:58 SRR957824.bam
-rw-r--r--  1 qsv231  staff  102389120 Jul  8 11:58 SRR957824.fixmate.bam
-rw-r--r--  1 qsv231  staff   72677432 Jul  8 11:58 SRR957824.fixmate.sorted.bam
-rw-r--r--  1 qsv231  staff   72627930 Jul  8 11:58 SRR957824.markdup.bam
-rw-r--r--  1 qsv231  staff    16664 Jul  8 11:58 SRR957824.markdup.bam.bai
```

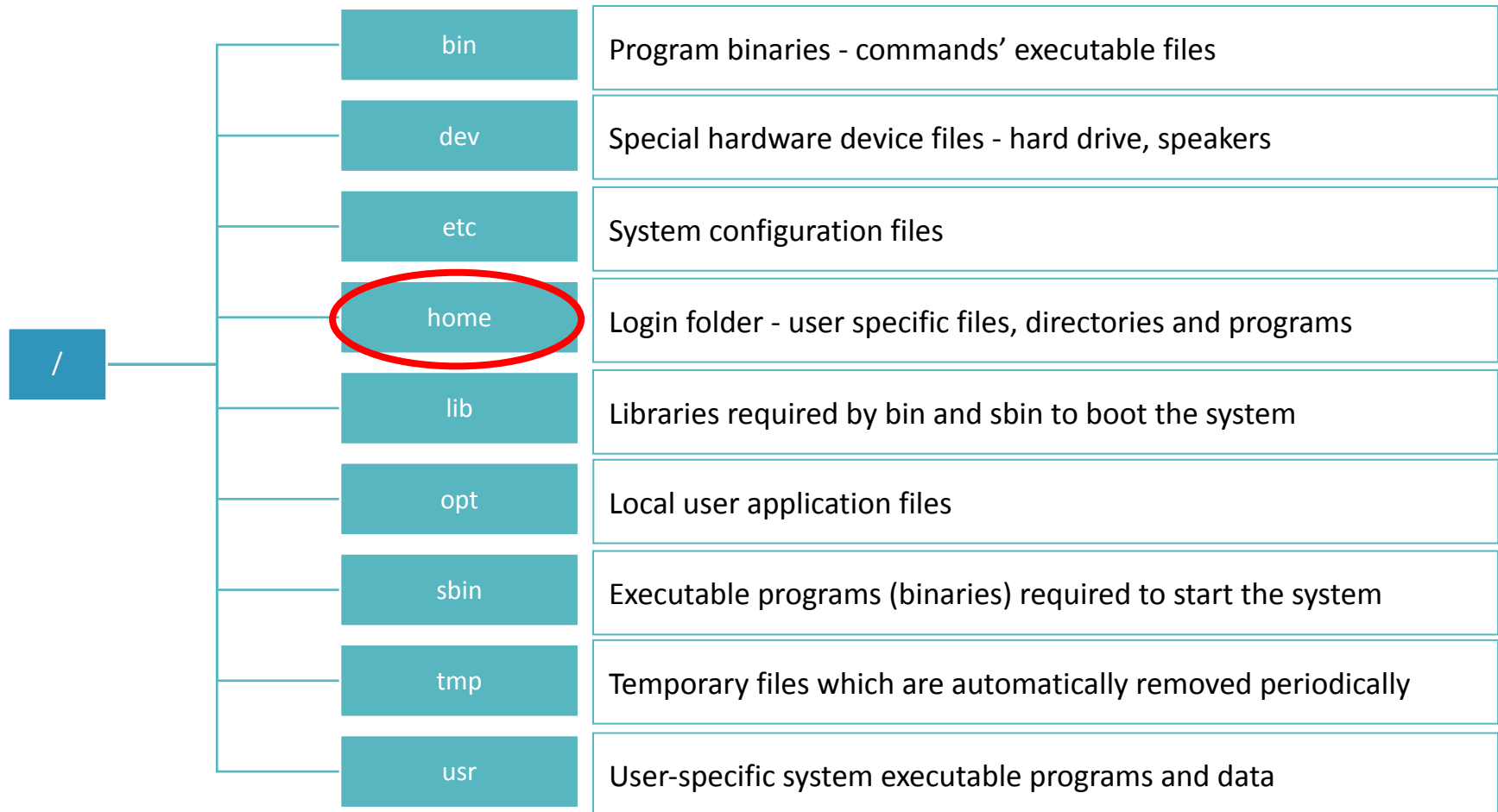
Examples of files:

- Any document (report, essay, presentation, sequence data file pdf, etc.)
- A computer program
- Binary Files
- A directory (folder)

File system structure



File system structure

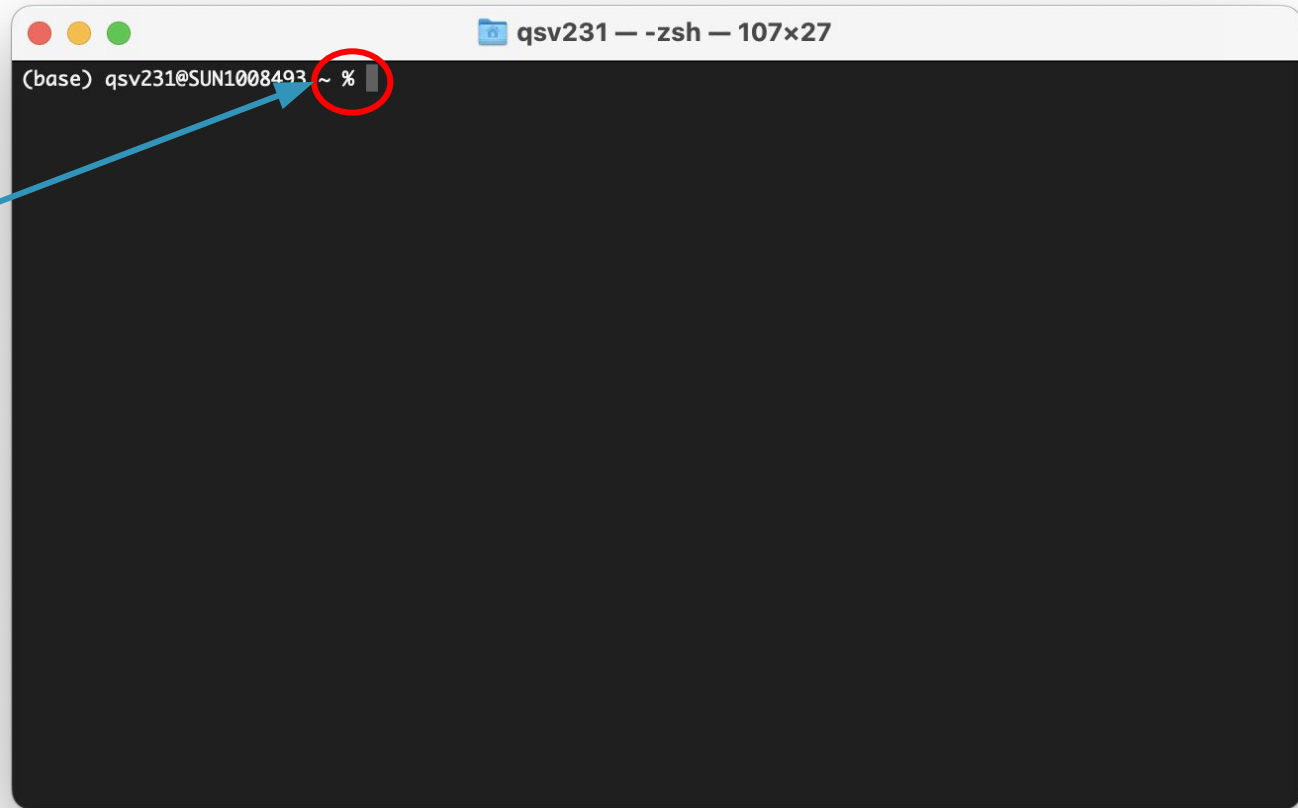


The terminal



The prompt

- % or \$



A terminal is a **text input and output environment** that provides a command-line interface using a shell program.

What is a UNIX Command?

- A UNIX command is a **program** inbuilt with UNIX OS that can be invoked in several ways.
- UNIX commands work interactively from a UNIX terminal.
- The commands are used for several purposes including getting information, manipulating data, displaying and linking files, network communication, and more.
- Not all UNIX Commands are universal.

Syntax of a UNIX command



- **The command** tells the computer what sort of operation is being done.
- **The options** assign more specific functions to the command, or add additional actions to the command.
 - Usually single letters
 - Options are preceded with a hyphen
 - Multiple options can be combined with no spaces or separators
- **The arguments** are any piece of additional information that might be necessary to execute the command.
 - If more than one argument, they must be separated by a space.

Other concepts

- Data streams
 - STDIN: Standard input. It takes text as input.
 - STDOUT: Standard output. The text output of a command is stored in the stdout stream.
 - STDERR: Standard error. Whenever a command faces an error, the error message is stored here.
- stdin takes as input text typed in in the terminal and stdout and stderr are normally printed out in the terminal.
- In UNIX, folders are called **directories**

Questions / Issues with installations?

Part II: Basic commands and working with files and directories

2

Basic Commands
Directory Management
File Management
Redirecting output
File permissions
Hands on 1 exercise

Keyboard shortcuts

Ctrl+A

Go to the
start of the
line

Ctrl+E

Go to the
end of the
line

Ctrl+C

Stop the
current
process

q

Exit a child
process
(less, more,
etc)

[Tab]

Autocomplete
file/dirname or
command

Basic commands

top

See active processes and the resources they're using

% top

man

Shows the manual page of a command

% man ls
% man cd
% man htop

history

List your previous commands

% history

clear

Clear your terminal window

% clear

Working with directories

ls

Prints the
content of the
current
directory

% ls

pwd

Shows the
current
directory

% pwd

mkdir

Creates a new
directory

% mkdir test1

cd

Access a
directory

% cd test1

Special characters

.

Current
directory

% ls .

..

Parent
directory

% ls ..

~

Home directory

% ls ~

*

Match any and
all characters

% ls *

Understanding path names

- A path is a unique location to a file or a folder in a file system of an OS.
- A path to a file is a combination of / and alpha-numeric characters.

Absolute path

An **absolute path** is defined as specifying the location of a file or directory from the root directory(/).

To write an absolute path-name:

- Start at the root directory (/) and work down
- Write a slash (/) after every directory name (last one is optional)
- You can use the shortcut ~ to reference the path to the home directory.

Relative path

A **relative path** is defined as the path related to the present working directory(pwd).

- It starts at your current directory and **never starts with a /**
- You can use the shortcuts for the current (.) or parent (..) directory as reference and specifies the path relative to it.

Working with files

cp

Copy a file

```
% cp f1.txt  
f1_copy.txt
```

mv

Move a file or
rename it

```
% mv  
f1_copy.txt  
f1.txt
```

rm

Removes
file(s)

```
% rm cars.csv
```

rmdir

Removes an
empty
directory

```
% rmdir data/
```

ln

Creates link
to a file or
directory
(symbolic or
hard)

```
% ln -s  
cars.csv  
cars_sl.csv
```

Display contents of a file

head

tail

more

less

Print the first N
lines of a file

Print the last N
lines of a file

View contents
of a file

View contents
of a file

```
% head -3  
cars.csv
```

```
% tail -3 cars.csv
```

```
% more cars.csv
```

```
% less cars.csv
```

File Management

cat

Concatenate
files and print
the contents of
a file(s) to the
terminal

```
% cat  
cars_aa.txt  
cars_bb.txt
```

nano

text editor
if arguments
given open a
file to
edit/create.

```
% nano f2.txt
```

touch

Creates a new
empty file

```
% touch f1.txt
```

wc

Count words,
characters lines
and bytes

```
% wc cars.csv
```

Compressed File management

gzip

Compress a file

```
% gzip cars.csv
```

gunzip

Decompress a file

```
% gunzip *
```

tar

Bundle files with compression (optional)

```
% tar -cvzf cars.tgz *csv
```

zcat

Print the contents of a zipped file(s) to the terminal

```
% zcat f1.txt.gz
```

Redirect output

>

Will send the
output of the
command to a
NEW file

```
% ls folder1 >  
files.txt
```

>>

Will APPENND
the output of
the command
to a file

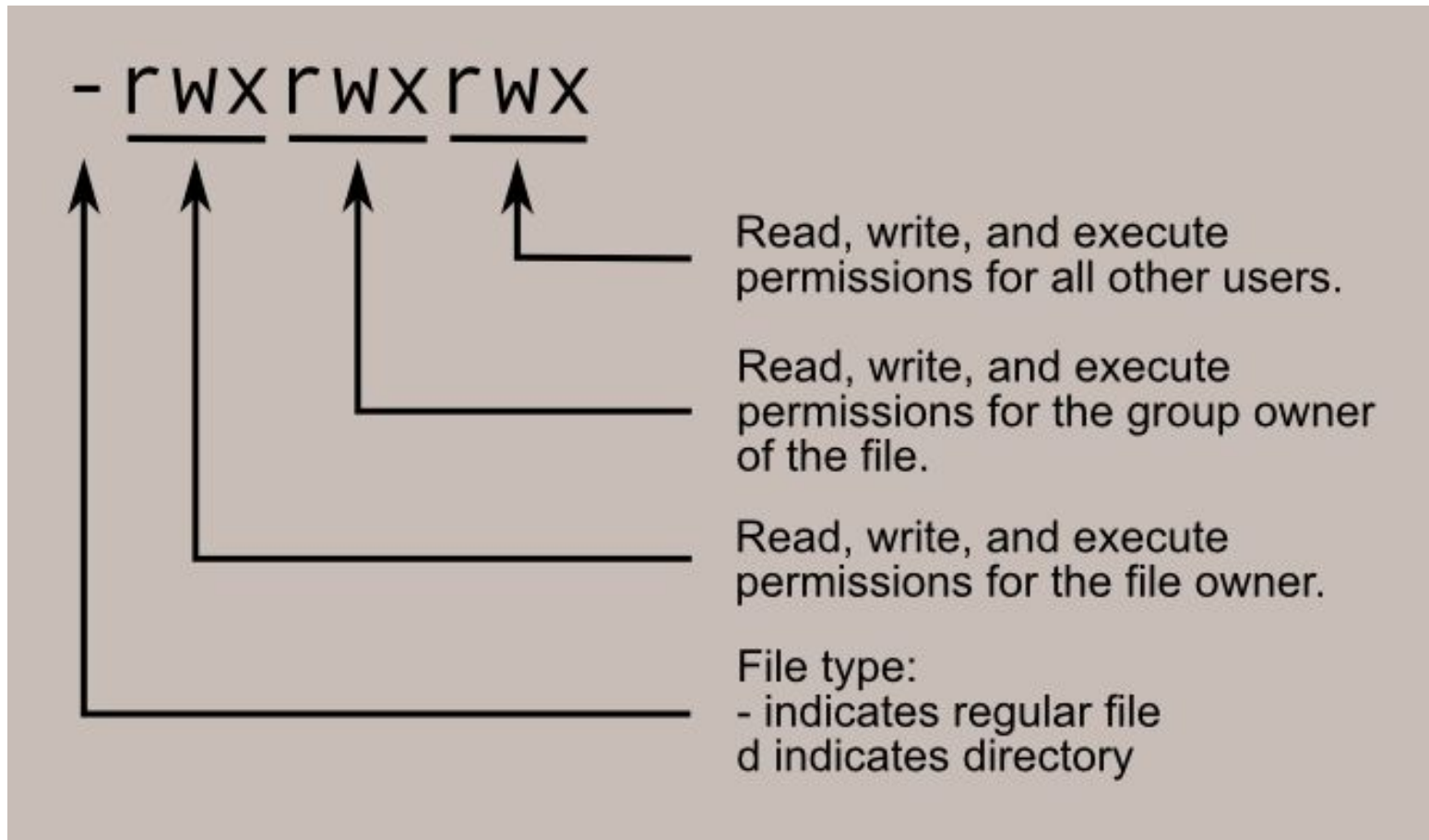
```
% ls folder1 >>  
files.txt
```

<

Redirect
output of one
command as
input to
another
command

```
% head -1 <(cat  
files.txt)
```


File permissions



Managing permissions

- To display information about the permissions of a file we use **ls -l**

```
% ls -l cars.tgz
-rwxr-xr-x 1 user group file_size day month year cars.tgz
```

- To change the permissions we use the **chmod** command.
- There are two ways of using chmod: the symbolic and the absolute.

Symbolic

- u, g, o, and a
- + (to add)
- - (to remove)
- = (to set)

```
% chmod u = rwx,go-wx file.txt
```

Absolute

- one digit from 0-7 per user type (u,g,o)
- 0: no permission
- 7: all the permissions

```
% chmod 744 file.txt
```

chmod

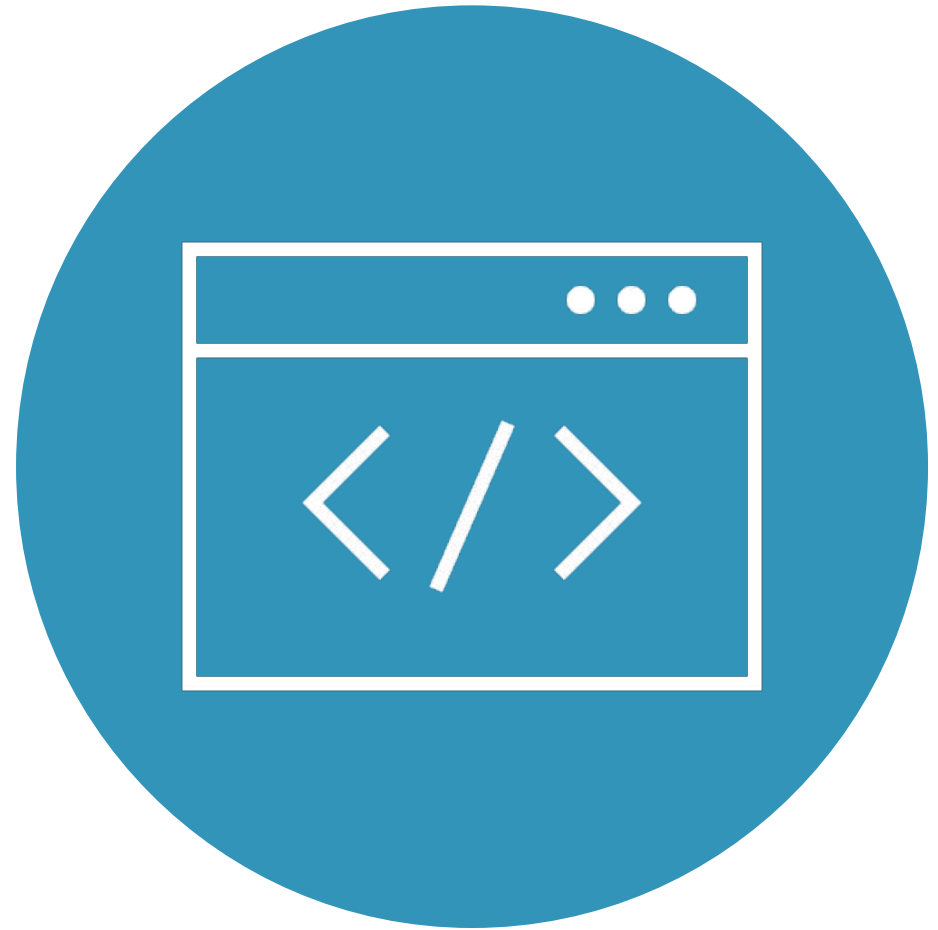
Change file(s)
permissions

```
% chmod o-x
cars.csv
```

Absolute permissions

Number	Octal Permission Representation	Ref
0	No permission	---
1	Execute permission	--X
2	Write permission	-W-
3	Execute and write permission: 1 (execute) + 2 (write) = 3	-WX
4	Read permission	r--
5	Read and execute permission: 4 (read) + 1 (execute) = 5	r-X
6	Read and write permission: 4 (read) + 2 (write) = 6	rw-
7	All permissions: 4 (read) + 2 (write) + 1 (execute) = 7	rwX

Hands on 1



1. Go to the exercises directory inside the Intro to Unix folder you downloaded.
2. Create a directory called `folder1` under `data/ho1`
3. Access the directory you just created
4. Create two files: `f1.txt` and `.f2.txt`
5. Write the numbers from 1 to 10 in `f1.txt` (one number per line)
6. Write the following sequence in `.f2.txt` (one letter per line)
 - **a a b b b c c c c d d d d d**
7. List all the contents of the directory in the long format – hint look at the manual of `ls`
8. Change the name of `f2.txt` to `f2.v2.txt`
9. Write the first 10 lines of `f1.txt` and all the lines in `f2.v2.txt` to a new file `f3.txt`
10. Change the permissions of `f1.txt` so that only the user can read and write the file
11. Count the number of lines in file `f3.txt`
12. Go back to the data directory and create a symbolic link to `folder1`

Part III: Text processing

Workshop outline

3

Text processing

Pipes

Hands on 2 exercise

Pipes

- A way to connect the end of something with the start of something else.
- Pipes will take the output of a command and give it to the following command as input.
- Pipes are very useful for text processing and filtering.
- They are specified with the control operator “ | ”

```
% cat cars.csv | head -10
```


Text processing commands

cut

Cut textfiles on
specific
columns

```
% cut -f1  
happiness.complete  
.tsv > col1.txt
```

paste

Paste two files
column-wise

```
% paste col1.txt  
happiness.complete  
.tsv
```

sort

Order contents
of file

```
% sort  
happiness.dup.csv
```

uniq

Get unique
entries

```
% sort  
happiness.dup.csv |  
uniq -c
```

Pattern matching with grep

grep

Search a
pattern in a file

```
% grep Male  
happiness.csv
```

Country Gender Mean
N=

```
AT Male 7.3 471  
AT Female 7.3 570  
AT Both 7.3 1041  
BE Male 7.8 468  
BE Female 7.8 542  
BE Both 7.8 1010  
BG Male 5.8 416  
BG Female 5.8 555  
BG Both 5.8 971
```

Some useful options:

- -v (reverse)
- -i (case insensitive)
- -c (outputs the count)
- -E (Regular expressions)
- -f (from file)
- -w (the exact word)

Some pattern hacks:

- ^ (line starts with)
- \$ (line ends with)

Examples

- All lines that start with a letter

```
% grep ^S countries.txt
```

- Lines that do not start with a letter

```
% grep -v ^S countries.txt
```

- Lines with a country from the list

```
% grep -f countries.txt happiness.complete.csv
```

Pattern substitution with sed

Command or streamline editor with multiple text processing functionalities.

sed

Search a
pattern in a file
and substitutes
with another

```
% sed s/Male/M/g  
happiness.csv
```

> Basic syntax to replace a pattern:

sed 's/search/replace/' file

- /g – replace all occurrences
- /1,/2,... - specifying which occurrence to replace
- /I – Ignore case

> Sed can also delete lines

sed 'nd' file

Options

- -e to run multiple commands
% sed -e 's/a/A/' -e 's/b/B/' file.txt

Examples

- Delete the first line

```
% ls -l | sed 1d
```

- Replace capital A and B for lowercase a and b

```
% sed -e 's/A/a/g' -e 's/B/b/g' happiness.complete.txt | head
```

- Convert csv to tab separated

```
% sed 's/,/\\t/g' happiness.csv | head
```

Hands on 2



1. Go to the directory [data/ho2](#) and list the files in it in long format.
2. Count how many lines start with a “,” in the file [happiness.csv](#)
3. Convert the file [happiness.complete.csv](#) into a tab-delimited file and store it as [happiness.complete.tsv](#)
4. Using the file [happiness.complete.tsv](#)
 - a. Count how many unique countries are listed
 - b. Count how many entries of each gender are
 - c. Replace all spaces with “_” within the same file
 - d. Replace “**Female**” for “**F**”, “Male for “**M**” and “**Both**” for **B** and look at the first 10 lines
 - e. Write a new file with all the “**Both**” entries from the countries in the file [countries.txt](#)

Part IV: Variables

Variables

- A variable is a character string to which we assign a value.
- Used to store information
 - number, text, file name, etc
- The variable name can only contain:
 - Letters (a to z or A to Z)
 - Numbers (0 to 9) *not at the beginning
 - Underscore (“_”)

Defining variables

- Variables are defined as:

```
% variable_name=variable_value
```

```
% NAME="Zara Ali"
```

```
% VAR1="Zara Ali"
```

```
% VAR2=100
```

- Variables of this type are called **scalar variables**. A scalar variable can hold only one value at a time.

Access variables

- To access the value stored in a variable, prefix its name with the dollar sign (\$)

```
% NAME="Zara Ali"  
% echo $NAME
```

echo

- You can access variables inside a string ("")

```
% echo "My name is $NAME"
```

Prints
something to
the terminal
(stdout)

- You can use variables as arguments to commands:

```
% data_dir="exercises/data"  
% cd $data_dir
```

```
% echo "Hello  
World"
```

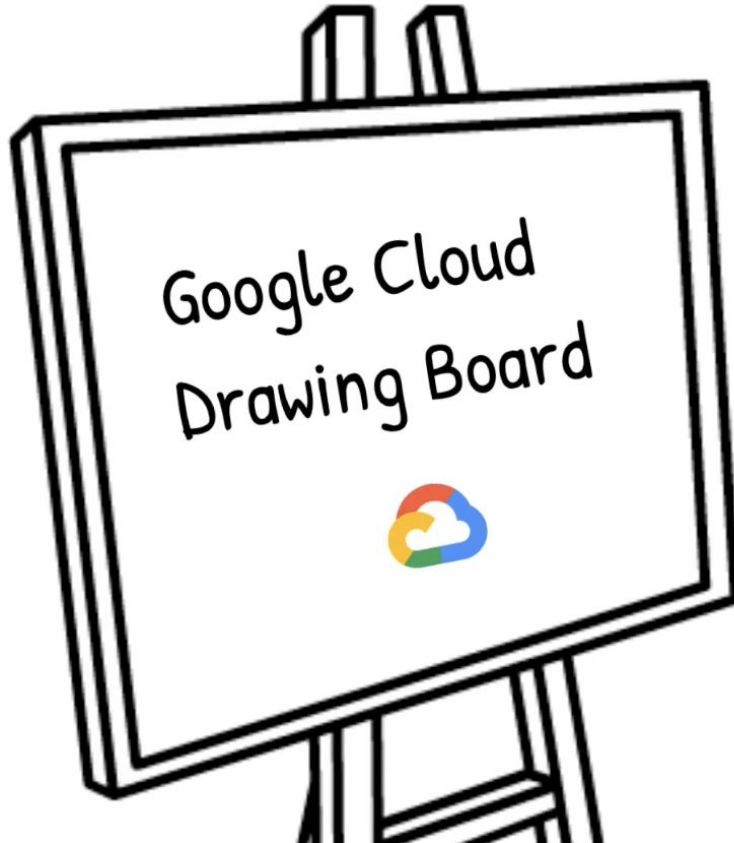
Part IV: Accessing a HPC server

Workshop outline

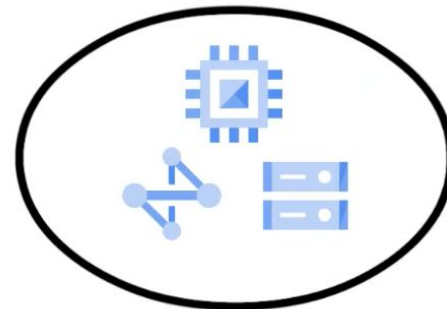
4

How to connect to a server
Running and monitoring jobs
How to copy files to and from a server

HPC servers



What is High
Performance Computing
(HPC)?



 @pvergadia



Connect to a cluster

ssh user@clusterdomain

```
gfemerling — gfemer@narval2:~ — ssh gfemer@narval.computecanada.ca — 105x29
```

Last login: Wed Sep 27 18:41:05 on console
[gfemerling@Maritima ~ % ssh gfemer@narval.computecanada.ca
[(gfemer@narval.computecanada.ca) Password:

 _ |
_	/ \		_	_	\ / \|_	_			
			()	_	_	\ v / ()	_
-		_	__,_-	_	_	_/__,_-	_	_	

Bienvenue sur Narval / Welcome to Narval

Aide/Support: support@tech.alliancecan.ca
Globus Collection: Compute Canada - Narval
Documentation: docs.alliancecan.ca

[gfemer@narval2 ~]\$ █

Jobs

The clusters use SLURM workload manager to handle to submissions and monitoring.

00_index.slurm

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

```
#SBATCH --job-name="salmon_index"
#SBATCH --account=rrg-hsn
#SBATCH --err=logs/err.index.%j.log
#SBATCH --output=logs/out.index.%j.log
#SBATCH --mem=64gb
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=12
#SBATCH --time=6:00:00
```

```
DIR=/project/6007998/lmoral7/references/gencode_GRCh37.p13
GENTR=${DIR}/GRCh37.p13.gentrome.fa
DECOY=${DIR}/decoys.txt
```

```
module load StdEnv/2020 gcc/9.3.0 openmpi/4.0.3 salmon/1.3.0
```

```
salmon index -t ${GENTR} \
              -d ${DECOY} \
              -p 12 \
              -i ${DIR}/salmon_genomeIndex \
              --tmpdir _indextmp \
              --gencode
```


Job submission and monitoring

```
sbatch job_script.slurm
```

Submit a job using a job script

```
squeue -u ${USER}
```

List all the processes of \$USER

```
scancel [PID]
```

Cancel a process using its process ID (PID)

```
scancel -u ${USER}
```

Cancel all processes that belong to \$USER

One way to transfer files to and from a server

```
% scp [source] [target]
```

scp

We need to tell scp to connect to the source/target computer and where to find the file(s)

```
user@cluster.computecanada.ca:[absolute path to file(s)]
```

copies files
between hosts
on a network

- Not needed for the computer you are running it from.

```
% scp gfemer@narval.computecanada.ca:/home/gfemer/storage/testcopyfile.txt .
```

```
% scp mytestscript.sh gfemer@narval.computecanada.ca:/home/gfemer/storage/
```

In summary

- HPC can help solve problems that your personal computer could never do or would take too long
- It can reduce the computing time by running jobs in parallel
- As part of McGill you can have access to Compute Canada's resources.
- Processes run as jobs that need to be submitted with a script
- You can transfer files to and from a server using scp.

What we have learned

1

Introduction to the UNIX operating system
Files and processes
Directory Structure
The Terminal

2

Basic Commands
Directory Management
File Management
Redirecting output
File permissions

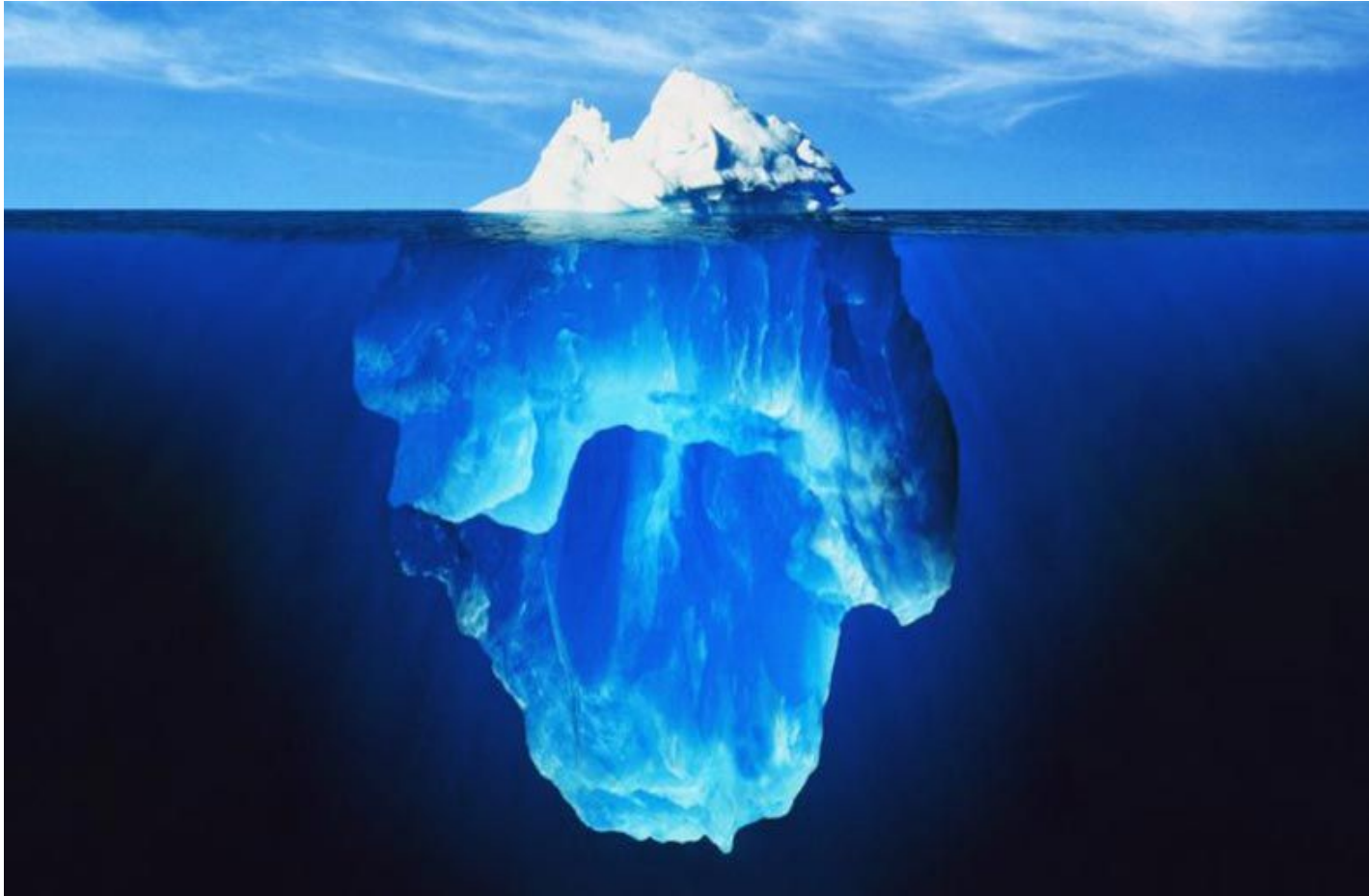
3

Text processing
Pipes

4

Variables
Connecting to a remote server
How to transfer files to and from a server

This is really just the tip of the iceberg!



Big thanks to:

- Alma Poceros Coba
- The MGSA team

**Thanks for
your
attention!**

Contact:

gfemerling@ucdavis.edu

maria.femerlingromero@mail.mcgill.ca



Datasets references

Hands on 1 – Cars dataset

<https://perso.telecom-paristech.fr/eagan/class/igr204/datasets>

Hands on 2 – Happiness surveys dataset

<https://perso.telecom-paristech.fr/eagan/class/igr204/datasets>

Extra Material - Unix variables

Types of variables

- When the shell is running, 3 types of variables are present:

Local

Variable that is present within the current instance of the shell. It is not available to programs that are started by the shell. They are set at the command prompt.

Environment

Available to any child process of the shell. Some programs need environment variables in order to function correctly.

Shell

A special variable that is set by the shell and is required by the shell in order to function correctly.

Shell variables

- Shell environment variables required by the shell to function.

\$USER

Stores the name
of the current
user

\$HOME

Has the path to
the user's home

\$PATH

Contains all the
directories
where
executable files
are stored

\$PWD

Contains the
path of the
current directory

Environment variables

- Variables that apply to both the current shell and to any subshells that it creates

```
% export MY_NAME=Georgette
```

```
% echo $MY_NAME
```

Local variables

- Are only available to the current process (current shell, current script, etc.)
- They disappear when the process is done

```
% MY_NAME_LOCAL=Georgette
```

```
% echo $MY_NAME_LOCAL
```

Environment vs local variables

```
% export MY_NAME=Maria  
% MY_NAME_LOCAL=maria  
% nano my_name.sh
```

```
#!/bin/sh  
echo "My name is ${MY_NAME}"  
echo "My name is ${MY_NAME_LOCAL}"
```

```
% chmod u+x my_name.sh  
% ./my_name.sh
```

Variables

- **Shell variables** are needed for the correct functioning of the shell. Be careful about creating a variable with the same name (in uppercase letters).
- **Environment variables** only apply to the current shell and subprocesses (scripts, programs, etc), when you open a new shell it will no longer exist. They are rarely used.
- **Local variables** exist only in the context that you created them. Useful for bash scripts to make it look more readable and when you need to use something multiple times in the same shell.