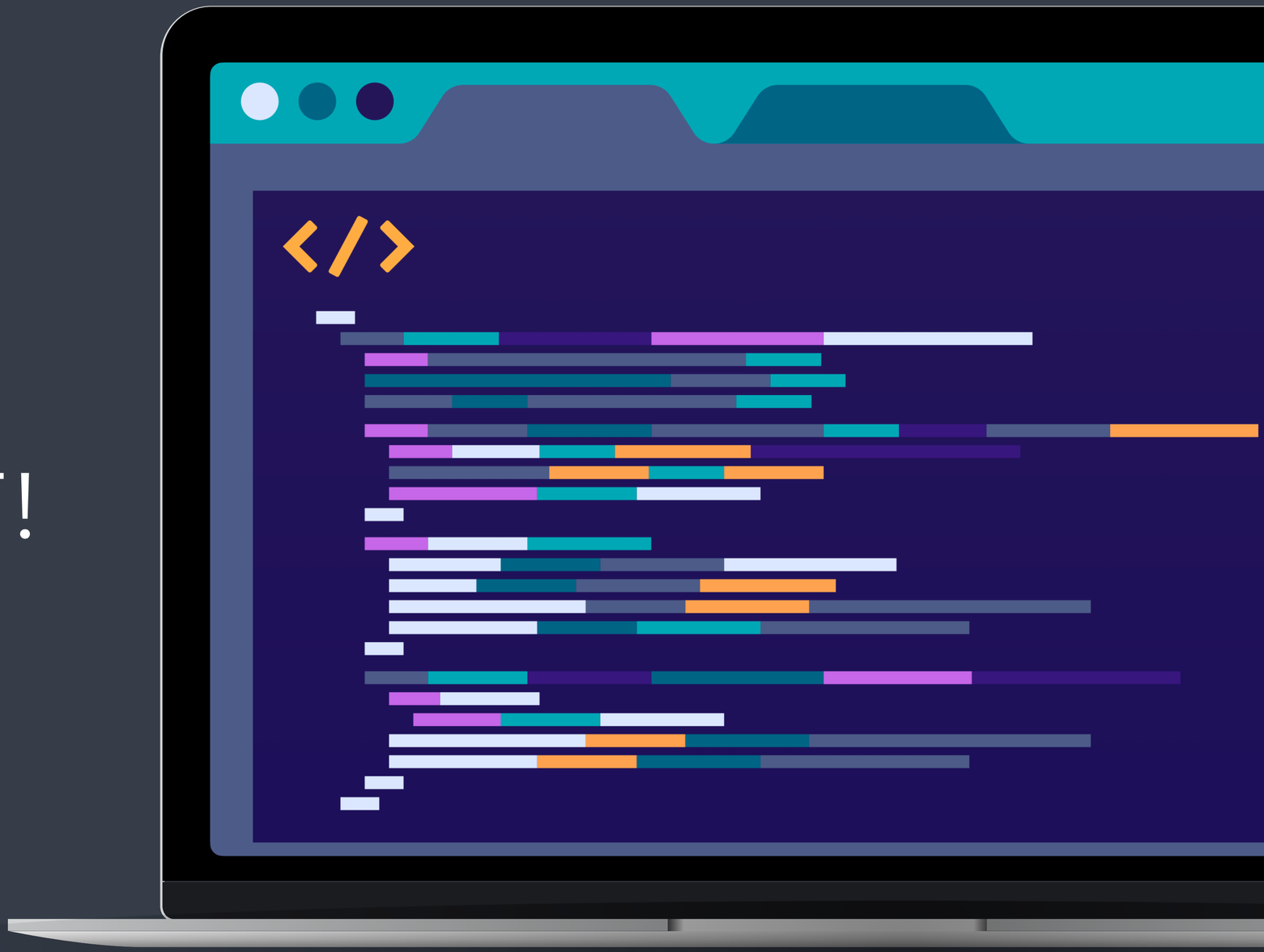
JUST BASH IT!



HEADS (SUND) DATA LAB

WEBSITE: https://heads.ku.dk/

- Center for Health Data Science (HeaDS)
- Data Lab Supports Researchers at the Faculty
- Consultation, Commission & Collaboration:
  - Data science and bioinformatics analyses
- Teaching: Courses & Workshops, Seminars.

Henrike Zschach

Thilde Terkelsen

KUB DATA LAB

WEBSITE: https://kub.kb.dk/datalab

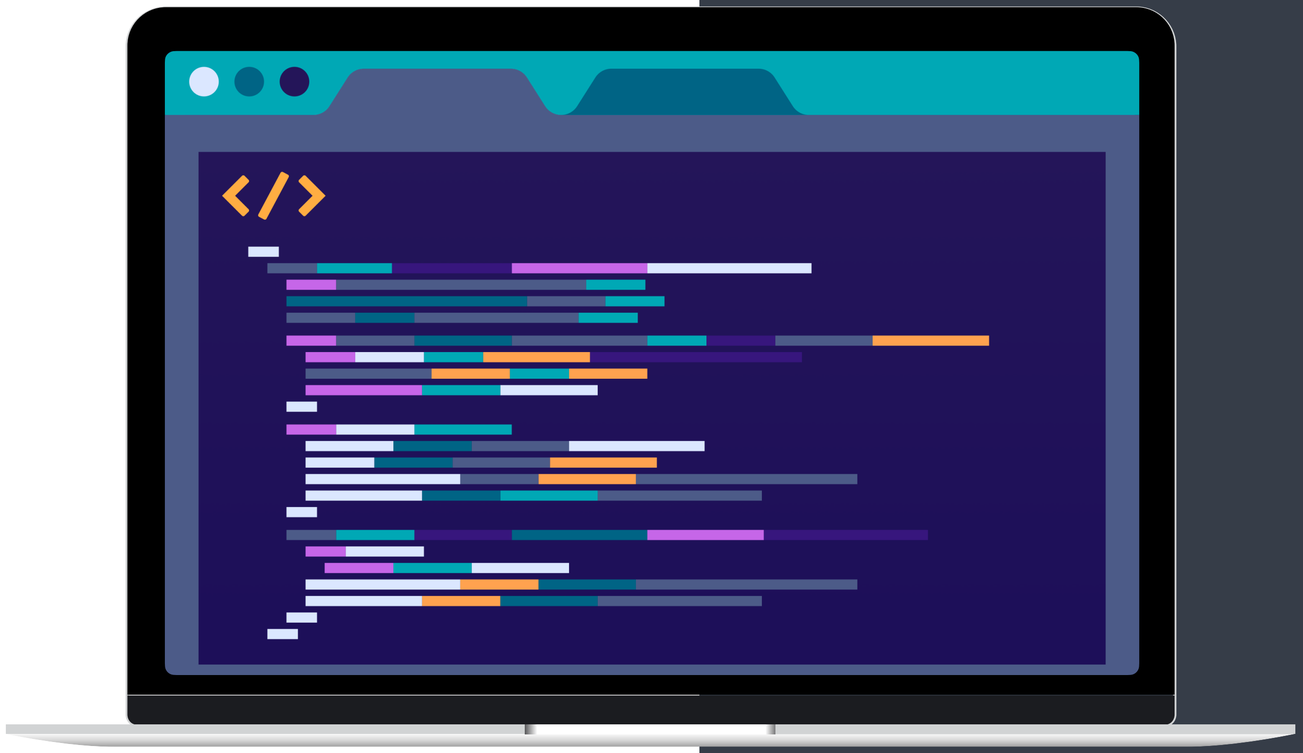
- KUB Datalab is an open space for learning - both KU students and teachers.
- Data wrangling: harvest, clean, analyse, and visualise data (novice and experienced)
- Free courses and learning activities

Ene Rammer Nielsen

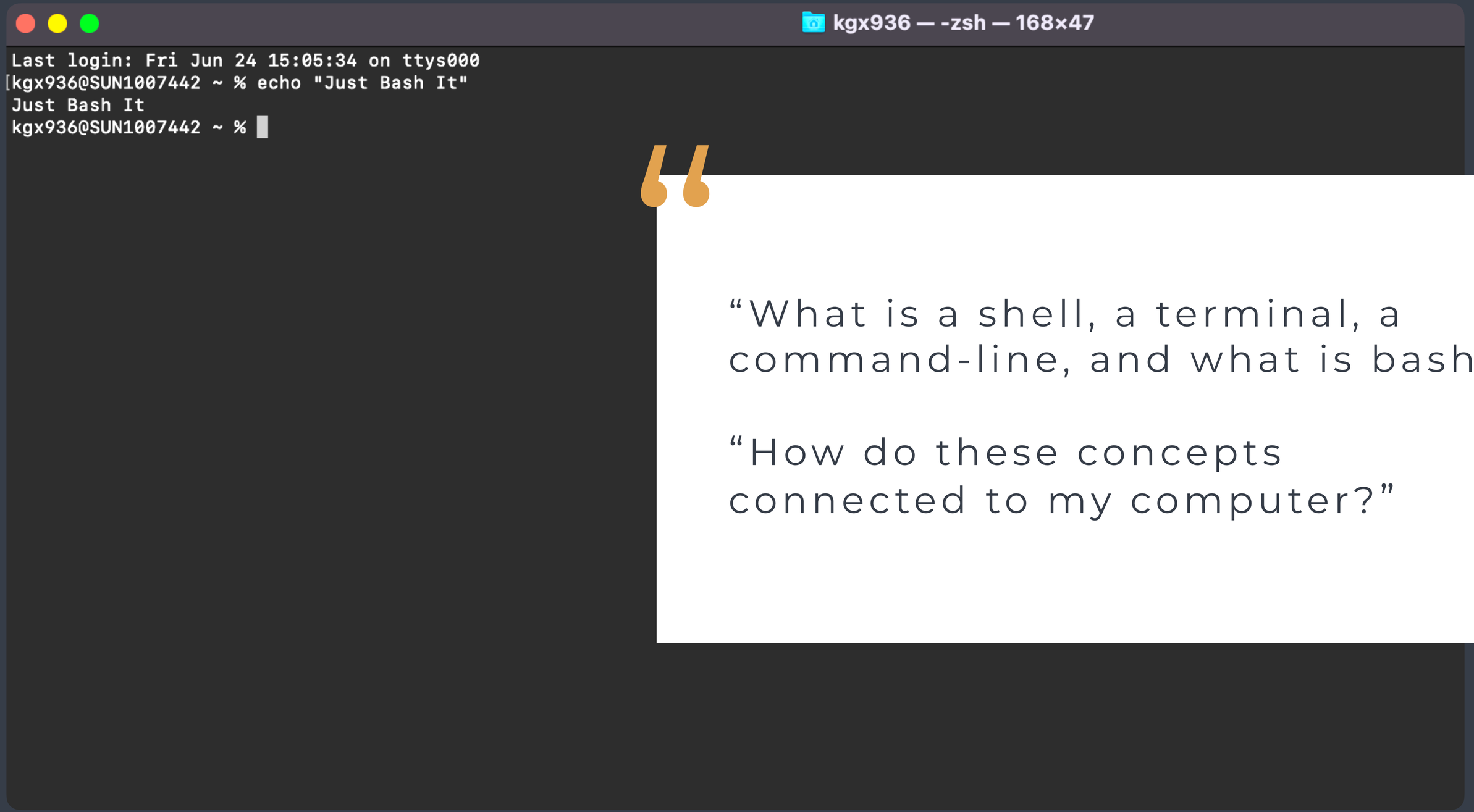
Daniel Pryn

PROGRAM

09:00 - INTRODUCTION TO COMMANDLINE
09:15 - NAVIGATING FILES & DIRECTORIES
09:40 - EXERCISE 1
10:00 - PROJECT ORGANIZATION & BACKUP
10:20 - COFFEE BREAK
10:30 - EXERCISE 2
10:50 - WORKING WITH FILES & DIRECTORIES
11:10 - EXERCISE 3
11:30 - MORE BASH COMMANDS
12:00 - LUNCH
13:00 - EXERCISE 4 (part 1 + 2)
14:00 - REDIRECTION & PIPES
14:20 - EXERCISE 5
14:50 - COFFEE BREAK
15:05 - SHELL SCRIPTS & LOOPS
15:20 - EXERCISE (part 1 + 2)
15:50 - SOFTWARE INSTALLATION UPKEEP & MORE
16:10 - EXERCISE 7



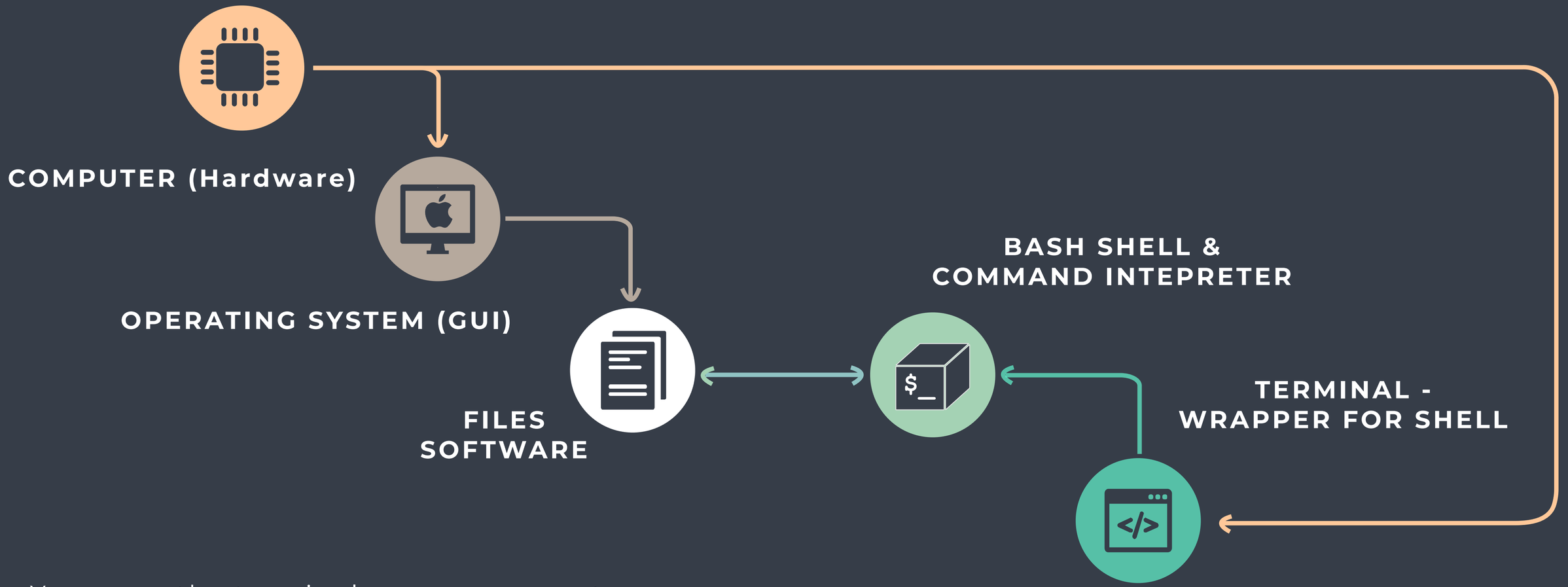
COURSE MATERIALS:
https://github.com/Center-for-Health-Data-Science/Just-Bash-It



“What is a shell, a terminal, a command-line, and what is bash?”

“How do these concepts connected to my computer?”

TERMINOLOGY



- You open the terminal on your computer
- You type bash commands
- Commands are interpreted and fed to the OS

SELL IT TO ME!

Benefits of bash & command line:

- Your analysis will be reproducible, automated and parallelised.
- Fast processes, less computationally intensive.
- Command line softwares.
- Power to handle big data and heavy computations.

A COUPLE OF EXAMPLES

REGISTRY DATA

Drug adverse effects and mortality:

Person sensitive
Huge files, many columns
Different formats

SEQUENCING DATA

Single Cell RNA from Cystic Fibrosis:

Paired-end sequencing
300 patients
Two fastq files per patient

DATABASES

Drug Databases - molecular structure & interaction:

Multiple large databases
Different formats
Maybe not downloadable

HOW DO I GET A BASH SHELL?

YEAH!
YOU HAVE A BASH SHELL & TERMINAL ALREADY.

Search for terminal on your laptop and open it.

OS X or UBUNTU

>= WINDOWS 10 - Windows Subsystem for Linux:
(https://adamtheautomator.com/windows-subsystem-for-linux/).

OR INSTALL A BASH SHELL/TERMINAL:

- **MobaXterm:** https://mobaxterm.mobatek.net/download.html
- **Cygwin:** https://www.cygwin.com/index.html
- **Cmder:** https://cmder.net/
- **PuTTY:** https://www.putty.org/

WINDOWS

In this course Windows users will be working on **MobaXterm**.
You should have **installed this shell** via. instructions in the introduction email.

WHAT WILL I LEARN TODAY?

The BASICS OF BASH

- The terminology
- Navigation w. bash
- Read, Edit, Copy

1

ORGANIZATION & STRUCTURE

- Directory Structure
- Paths & Permission
- Reproducibility

2

MORE BASH COMMANDS

- Manipulate files
- Subset, Count
- Print, Sort, Match
- Cut, Paste, Split

3

SCRIPTS & AUTOMATIZATION

- Stdin & Stdout
- Piping
- Loops & Scripts
- Pipelines

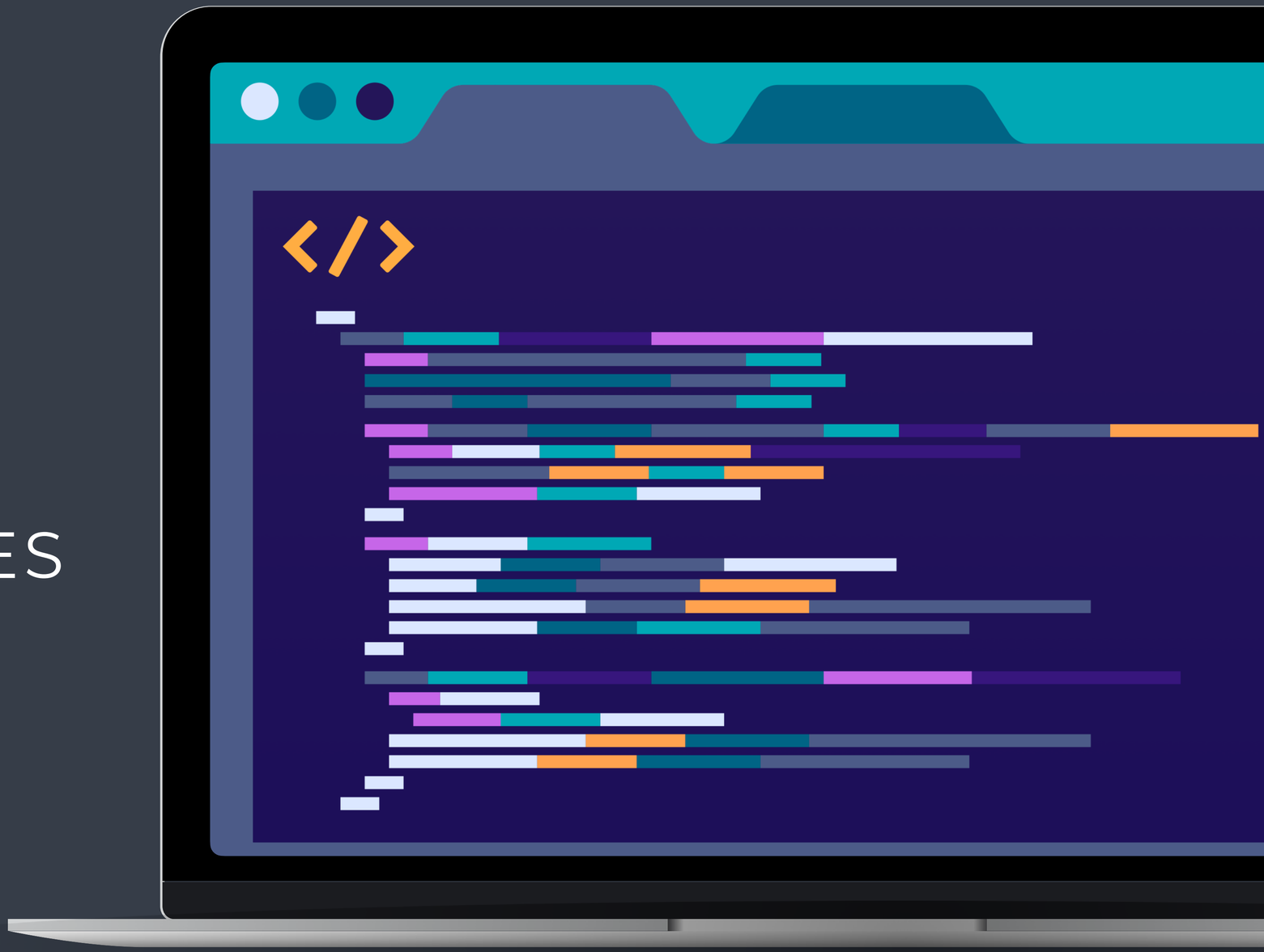
4

STORAGE, BACKUP & MORE...

- Manage Software
- Workflows
- Backups
- Computing Power

5

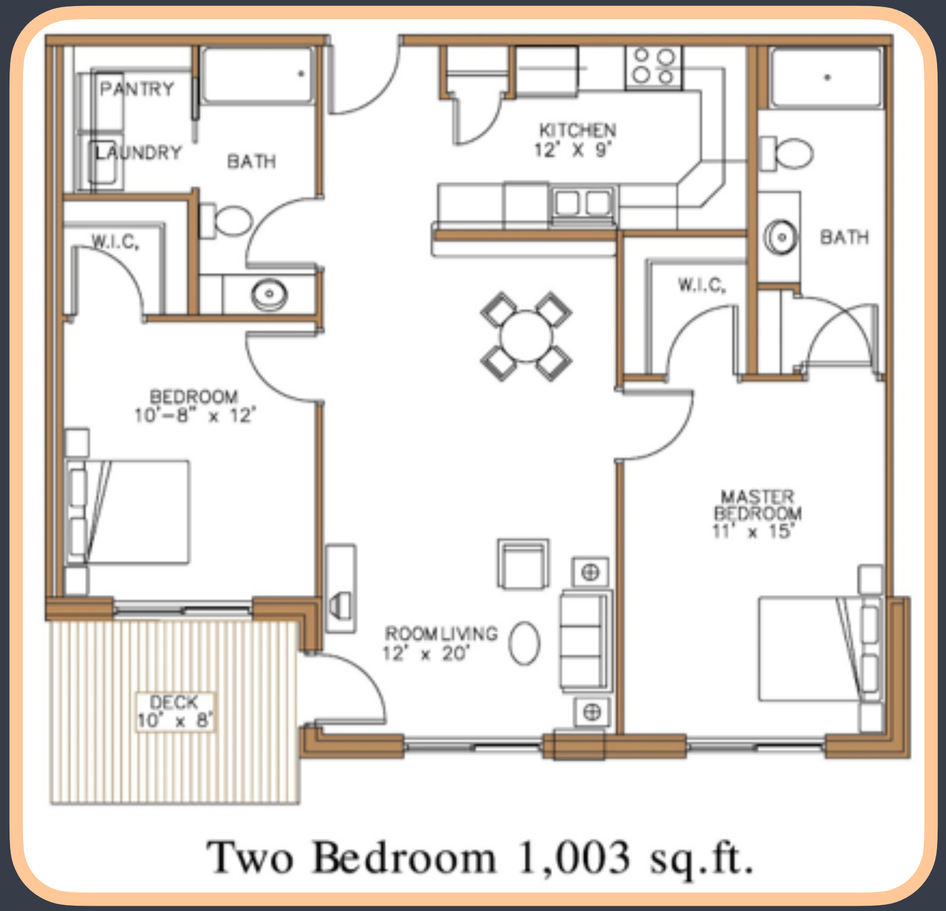
1. NAVIGATING FILES & DIRECTORIES



THE DIRECTORY TREE

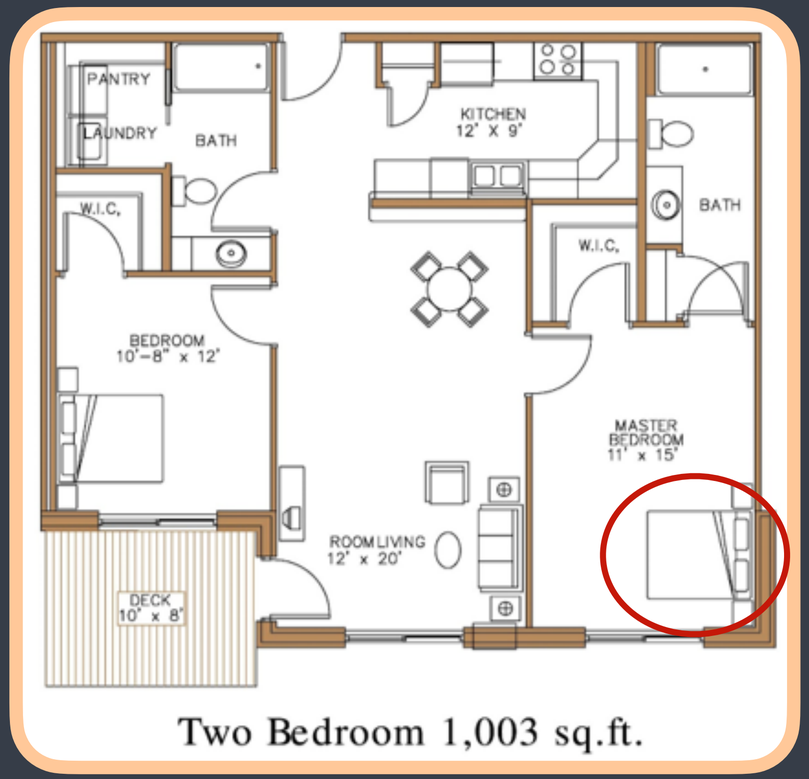
Your computer is **like a house**:

- It has different rooms, i.e. **directories** which contain items, i.e. **files**.
- You cannot be in two different rooms at the same time.
- Your files are items in the house and they are in specific places. Your bed is in your bedroom.
- In order to interact with your item **you need to know where it is.**
- You can move items from one room to another.

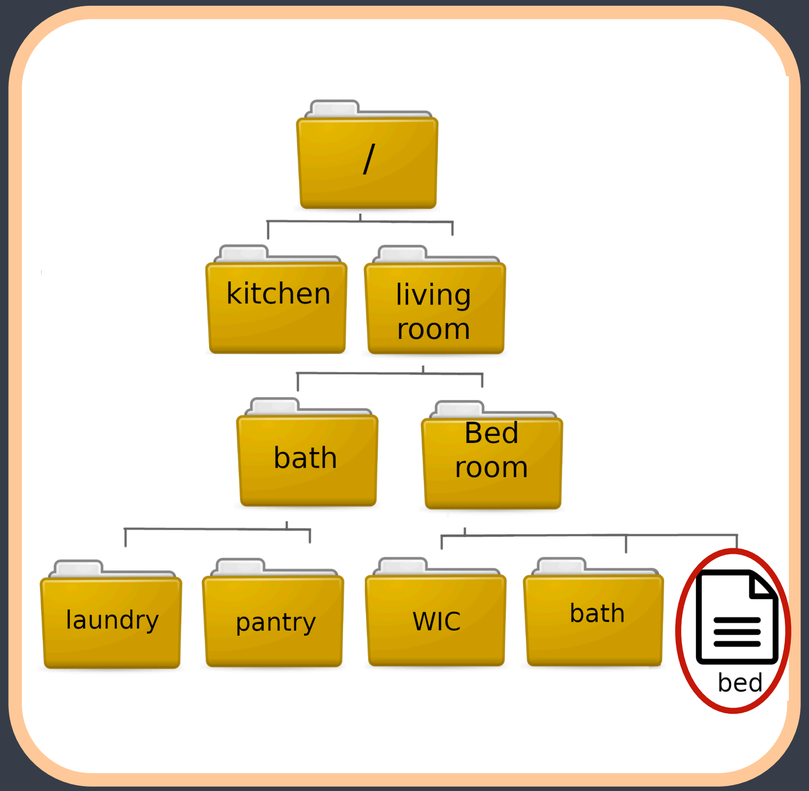


THE DIRECTORY TREE

A loose translation of the floor plan into a directory tree:



The **directory tree** is hierarchical and starts at the 'root' = '/'

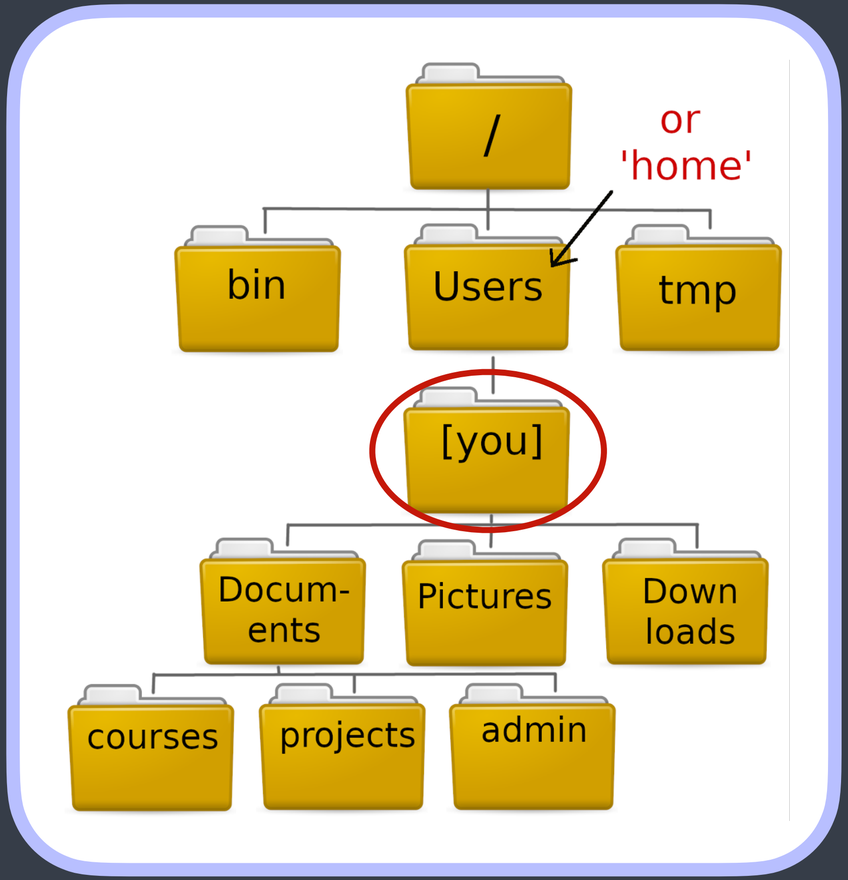


THE HOME DIRECTORY

- When you open your terminal you are in your home directory.
- The directory you are currently in is called your **working directory (wd)** - you can **check it**:

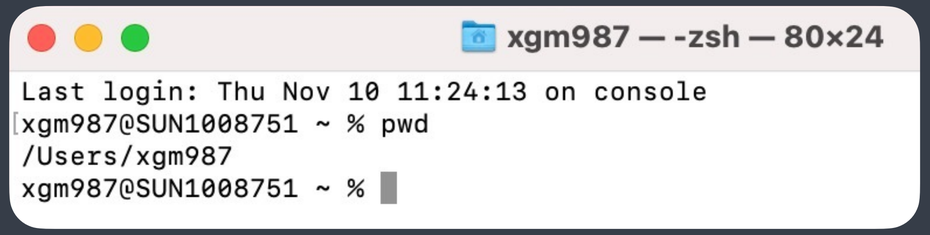
```
$ pwd
```

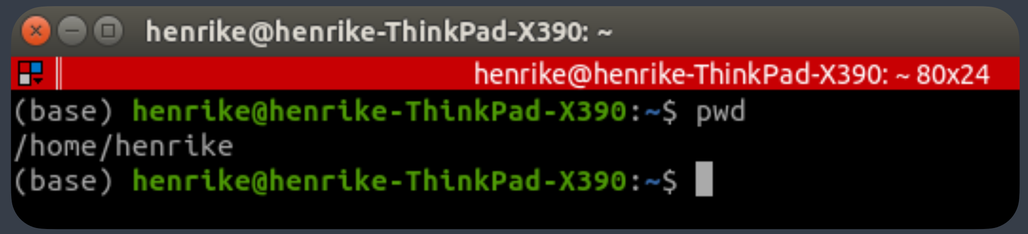
- **Open your terminal and check where you are!**

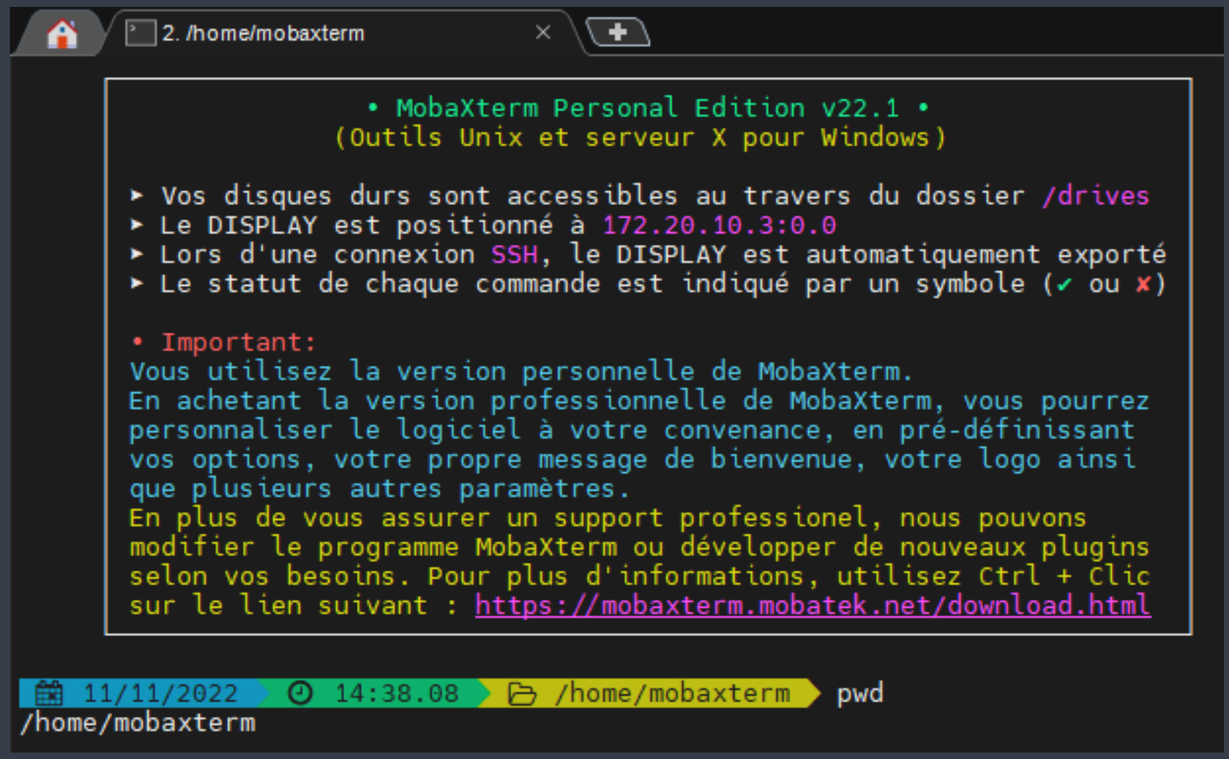


THE HOME DIRECTORY

What the home directory is called can differ between computers! Do not be too bothered by that. All home directories are beautiful <3!



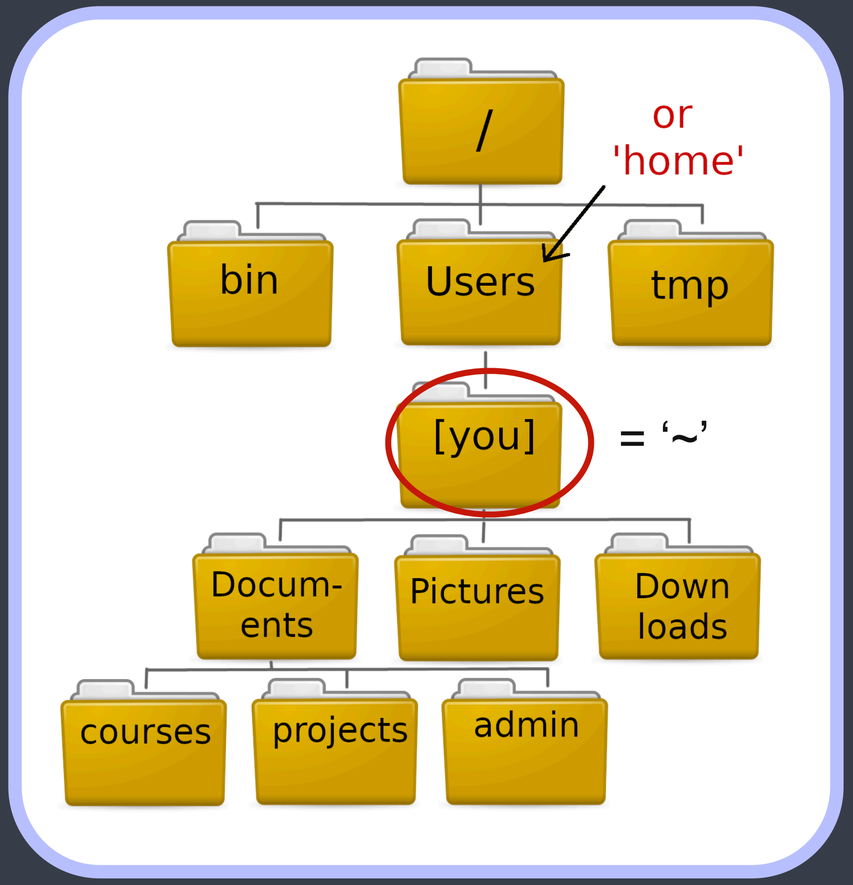




As a rule the home directory is generally two steps from the root '/' and it is personal.
When there are several users on the same machine each has their own home directory.

THE HOME DIRECTORY

- Your home directory is also represented by the symbol **‘~' (tilde)**.
- You may notice this symbol later when looking at your current directory or changing to another directory.



LISTING CONTENT

- To **list all contents of a directory**, including other directories, use the command:

```
$ ls
```

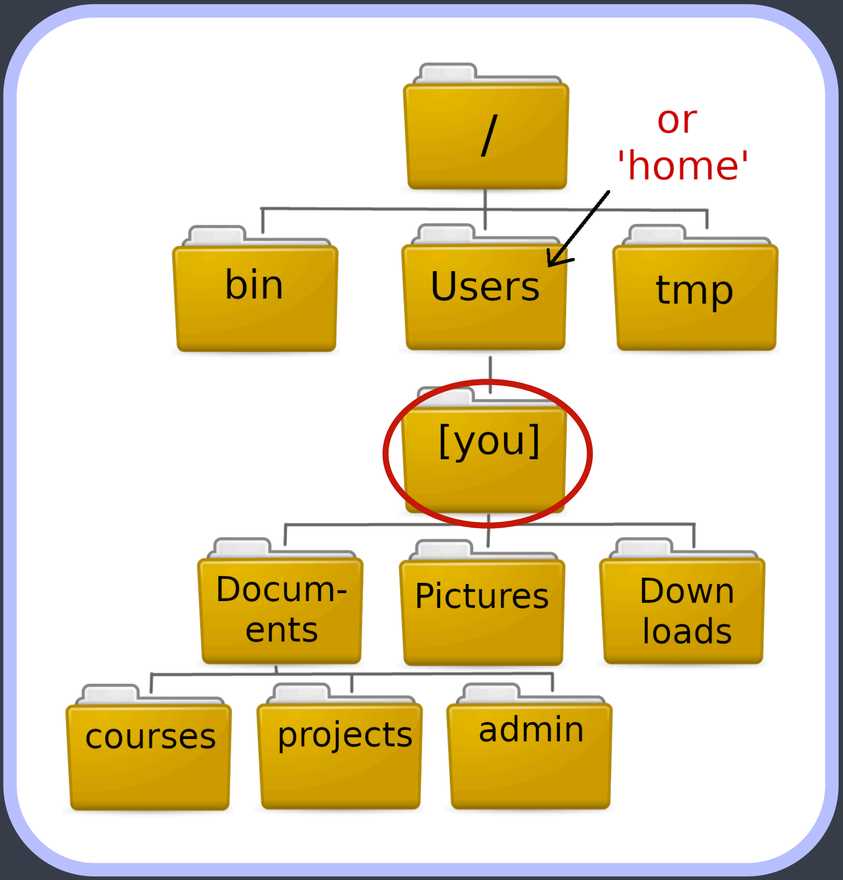
- ls will list what is in your current **wd**.

- You can list other directories by specifying the **path** to them:

```
$ ls /home
```

```
$ ls /Users
```

- ls has many useful options such as **–l, -t** and **–h, -F**. Try it out!



CHANGING DIRECTORIES

- From your **wd**, you can easily move into any subdirectory:

```
$ cd Documents
```

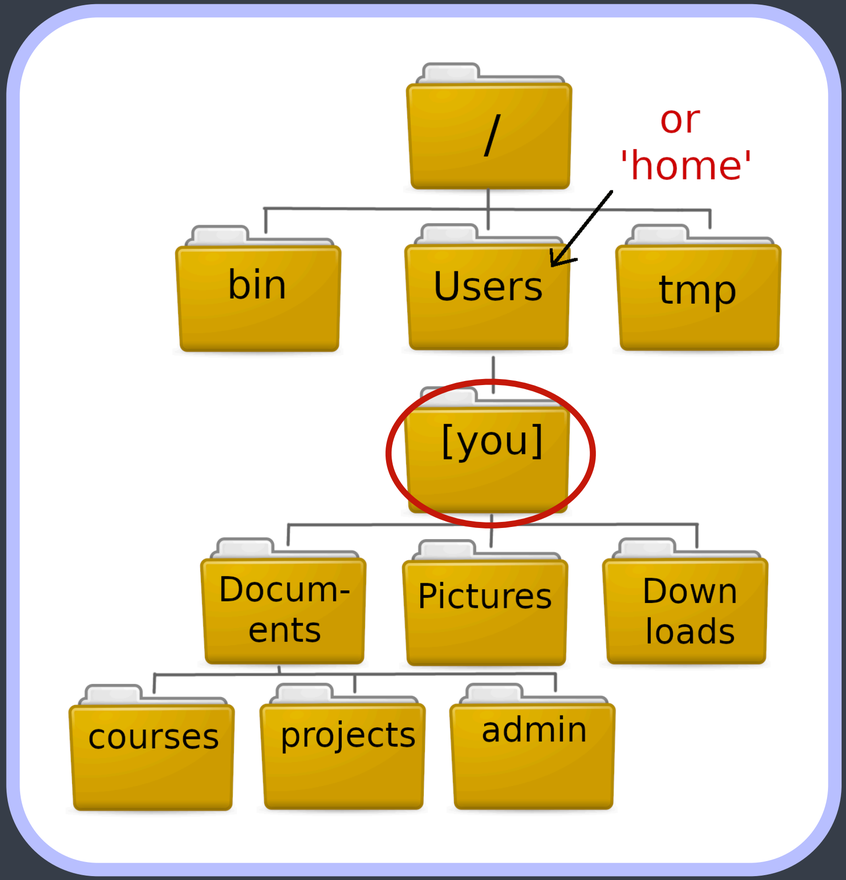
- You can also move several directories down:

```
$ cd Documents/courses
```

- To move ‘up' in the directory tree you need to use '../' . Each time you write this, you go one level up.

```
$ cd ../tmp
```

- 'cd' without any arguments brings you back to your home directory. Try it out!



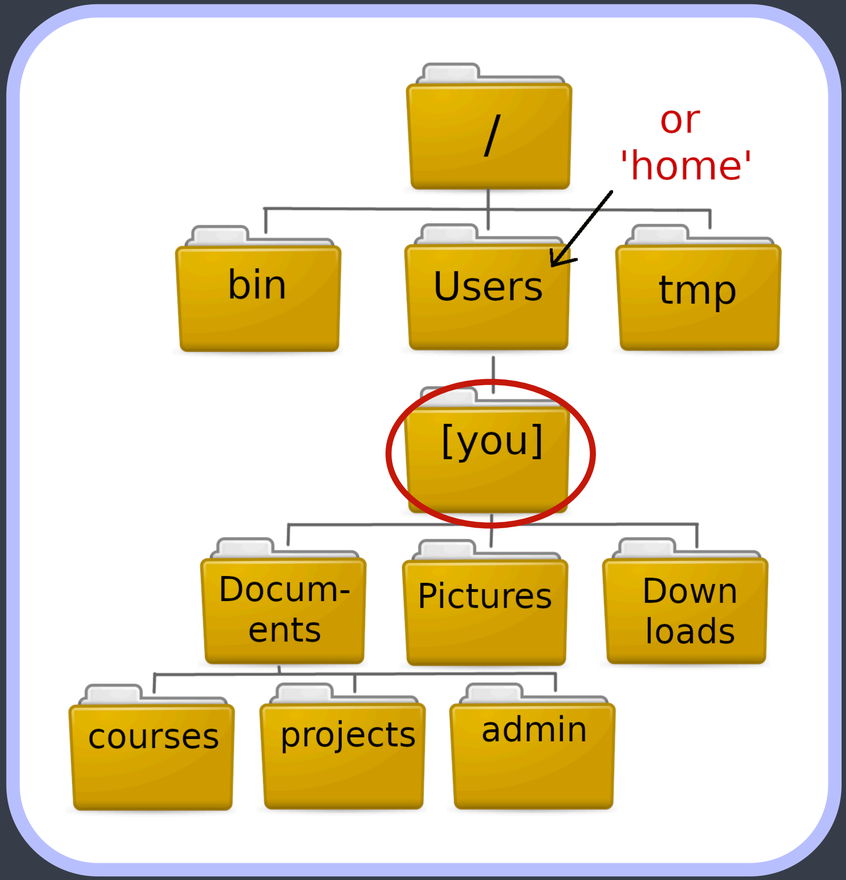
CHANGING DIRECTORIES

- cd without any arguments will always bring you to your home directory, no matter what your current directory is:

```
$ cd
```

- cd with a minus will bring you back to the last directory you came from:

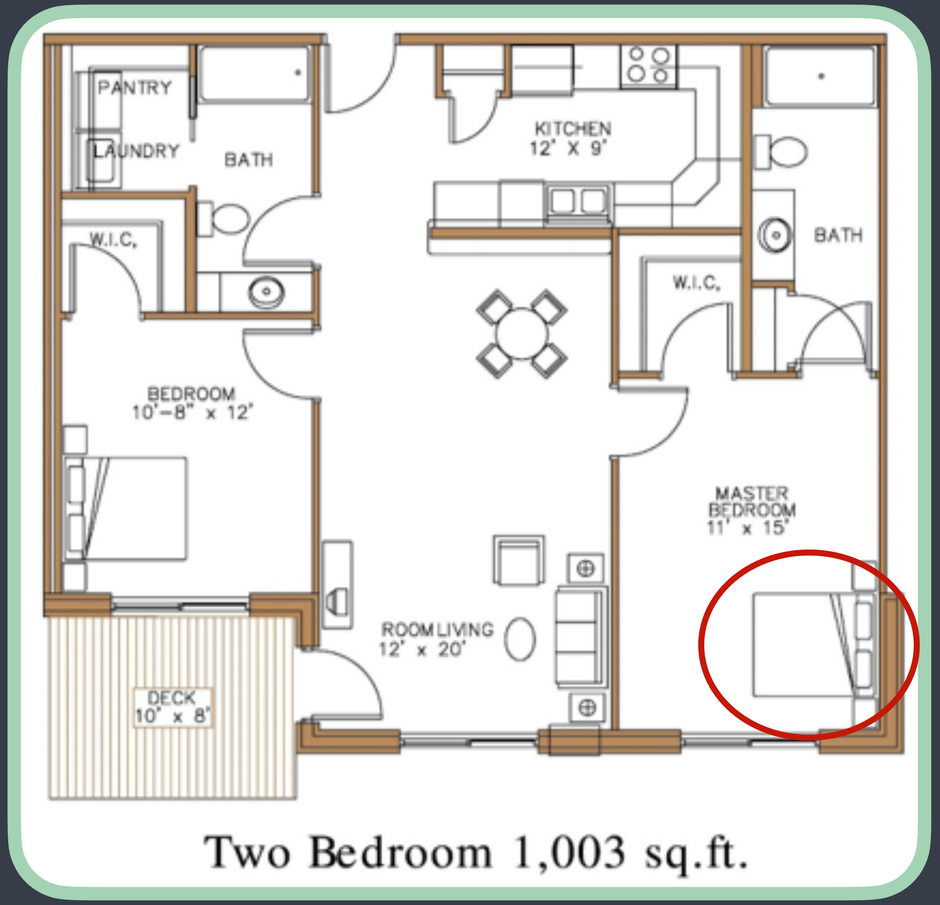
```
$ cd -
```



PATHS

- Each file and directory exist in one specific place on your computer. One could say they have an address.
- This ‘address’ is called their path. It is an instructions how to find the file or directory.
- Following the example from earlier, the path to you bed would be:

```
/entrance/living_room/Bedroom/bed.txt
```



PATHS

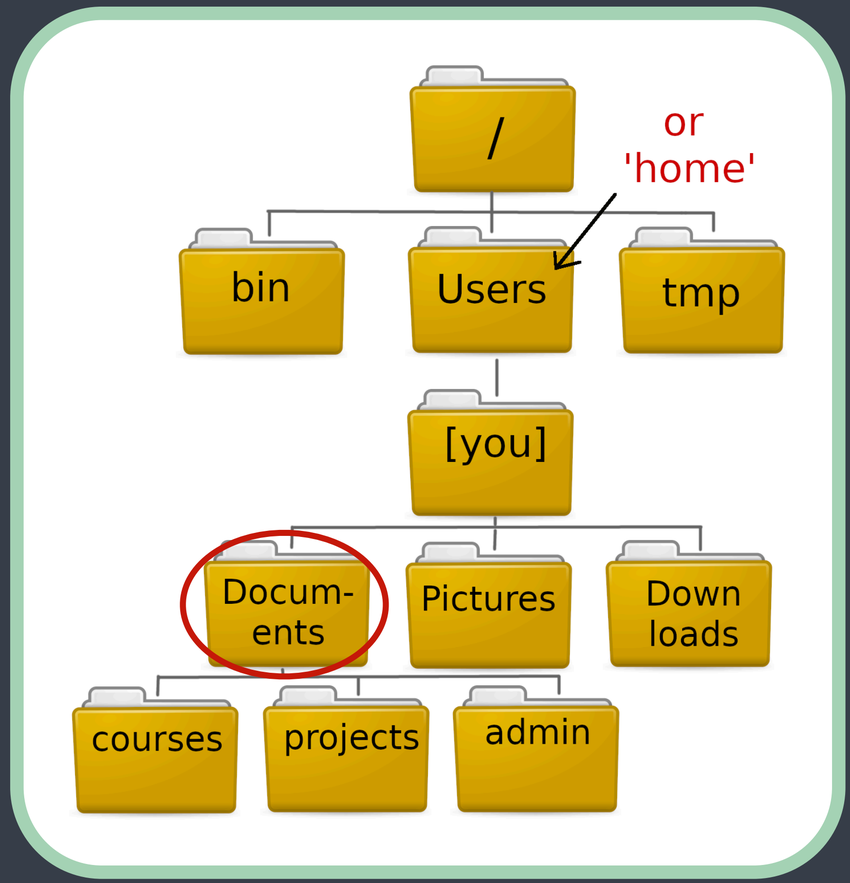
- Paths can be full/absolute or relative.
- Full paths always start at the root. Since ~ represent the path to your home directory it naturally includes the root. So

```
/home/[you]/Documents
```

and

```
~/Documents
```

- Are the same place and the same path.
- Full paths are always unambiguous since they describe the entire 'journey' starting from the root.



PATHS

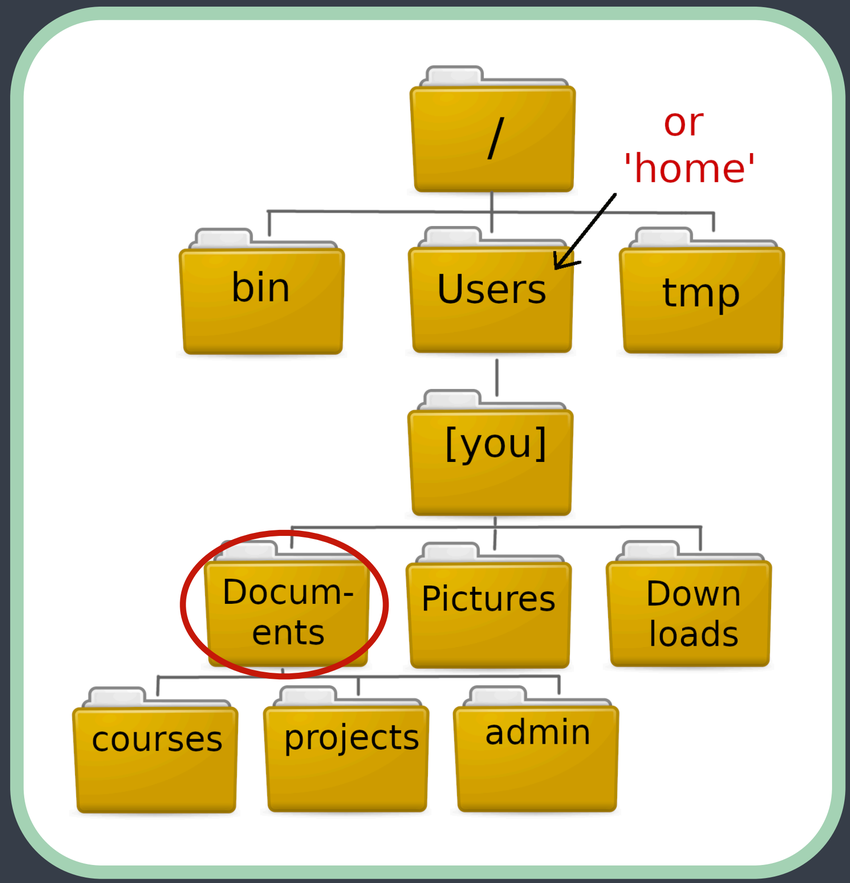
- A relative paths is an address relative to the current working directory.
- If my working directory is ‘Documents’, I can address this presentation as:

```
courses/Just-Bash-It-main/Commandline_Workshop.pptx
```

- Relative paths can include going up the directory tree:

```
../Pictures/summer2022/Bornholm
```

- This notation does not make sense for full paths as they start from the root and go strictly downwards.



MOVING FILES AND DIRS

- For this part, lets go into the '*Examples*' folder of the github repo.

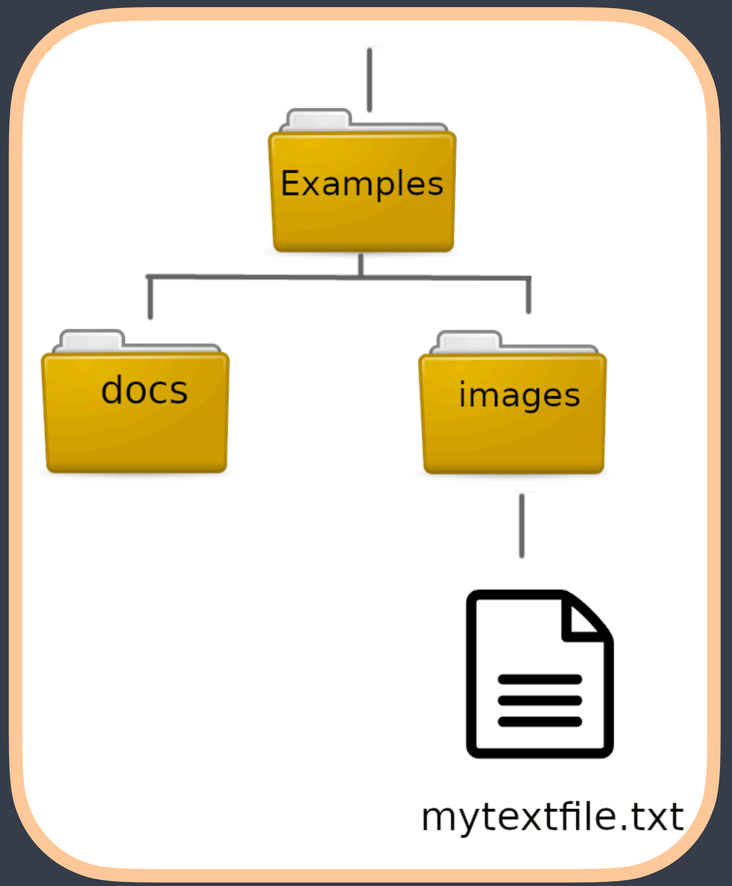
- You can move files (and directories) from one place into another with **mv**:

```
$ mv mytextfile.txt ../docs
```

- **mv** can also be used for renaming:

```
$ mv mytextfile.txt my_text_file.txt
```

- Do not try to move or remove your home directory or other system critical dirs. This may have unintended consequences.



COPY AND REMOVE

- Files can also be duplicated with the **cp** command:

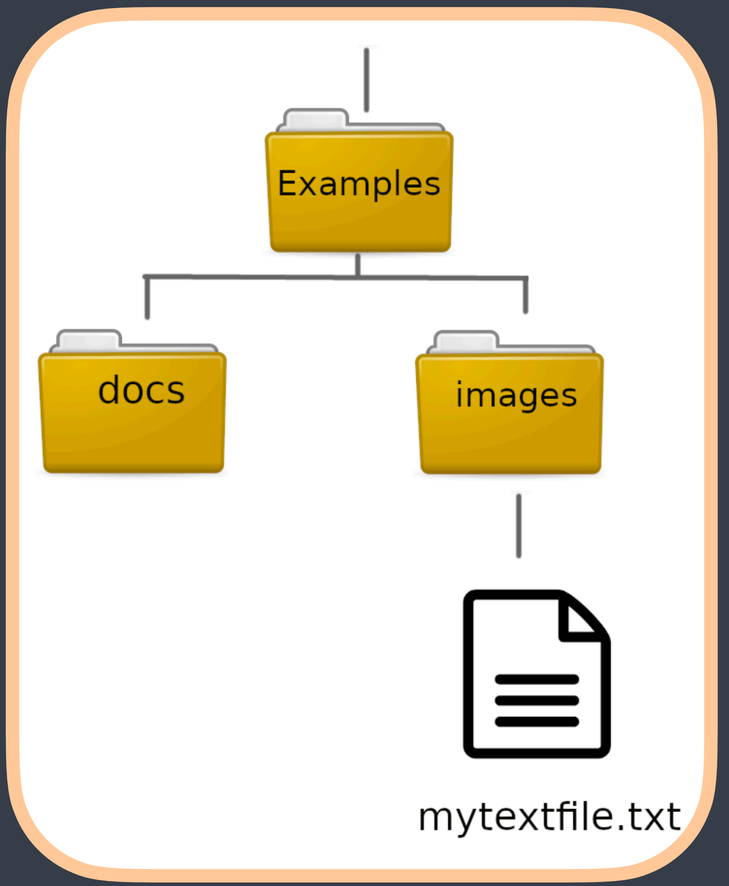
```
$ cp my_text_file.txt text_copy.txt
```

- To copy a directory (and all its contents!) you need the **–r** flag

- To remove files and dirs, use **rm**

```
$ rm text_copy.txt
```

- Unlike in many graphical file managers, **rm is permanent!** You cannot recover a removed file and you will not be asked whether you are sure you want it gone.



COMMANDS

We interact with the computer by issuing **commands** to it. You have already tried some like **cd**, **ls**, **pwd**.

Some commands can stand by themselves like **pwd**, but often they have arguments and options/flags:

The command:
List contents

The options/ flags:
Long format &
Human-readable size

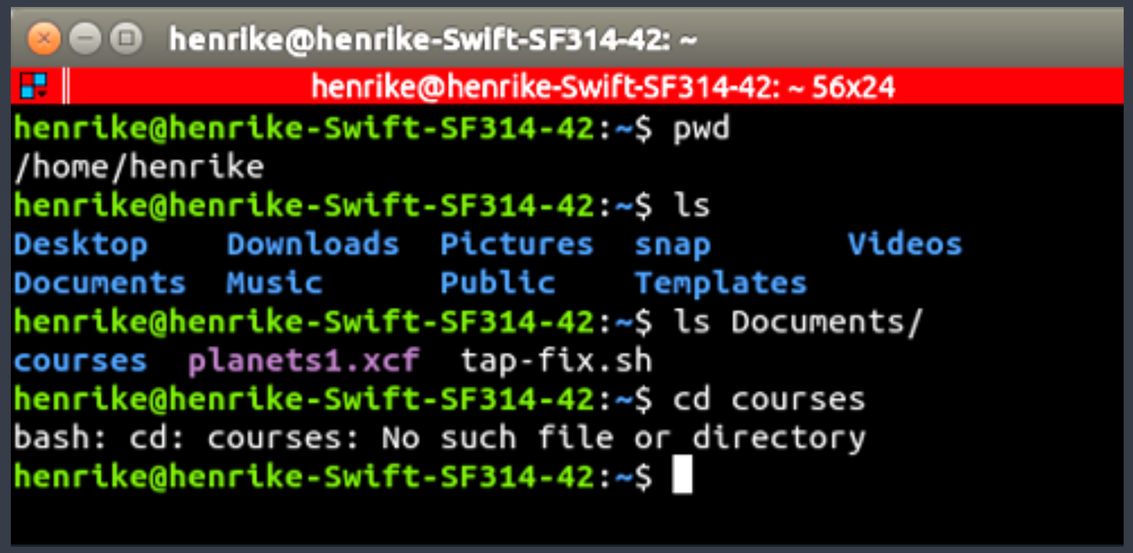
The argument:
Path to the directory
to work on

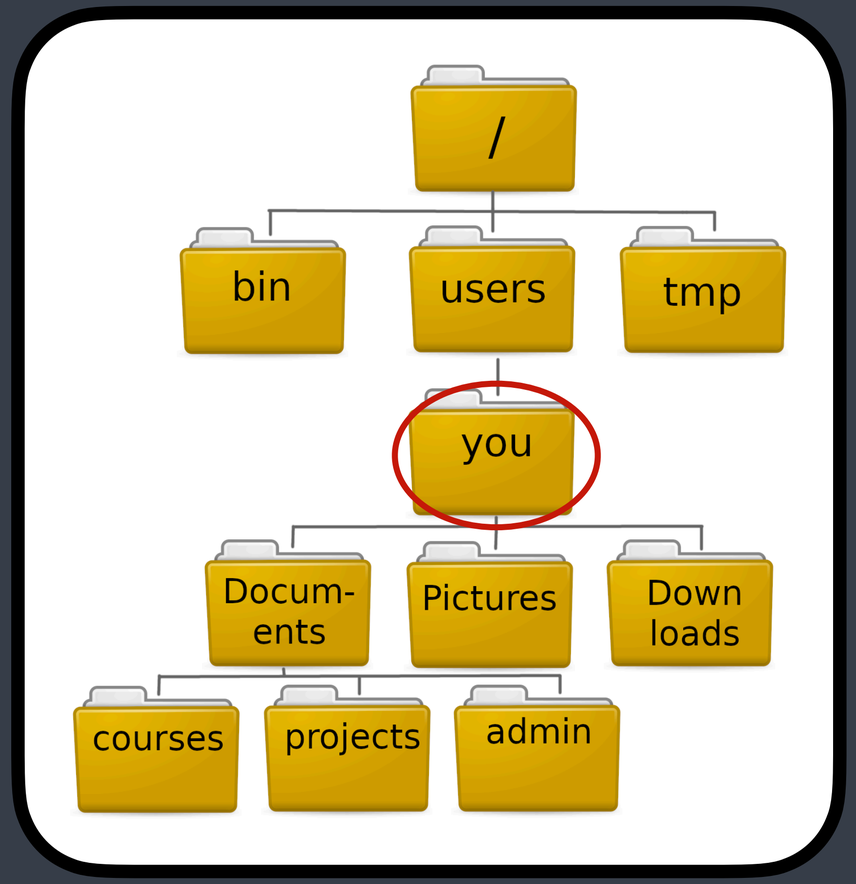
WHEN THINGS GO WRONG

Assuming I am in my home directory and I write:

```
$ cd courses
```

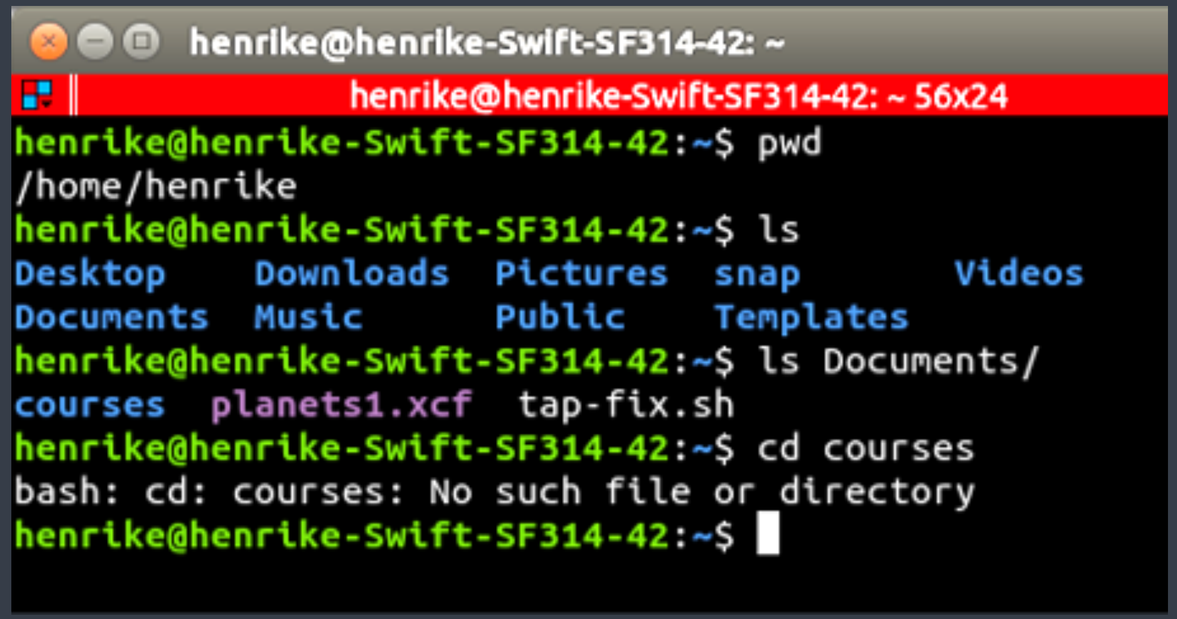
What will happen?





WHEN THINGS GO WRONG

1	Don’t panic!
2	Read the error message
3	Try to correct your error
4	If lost, google or read the help page



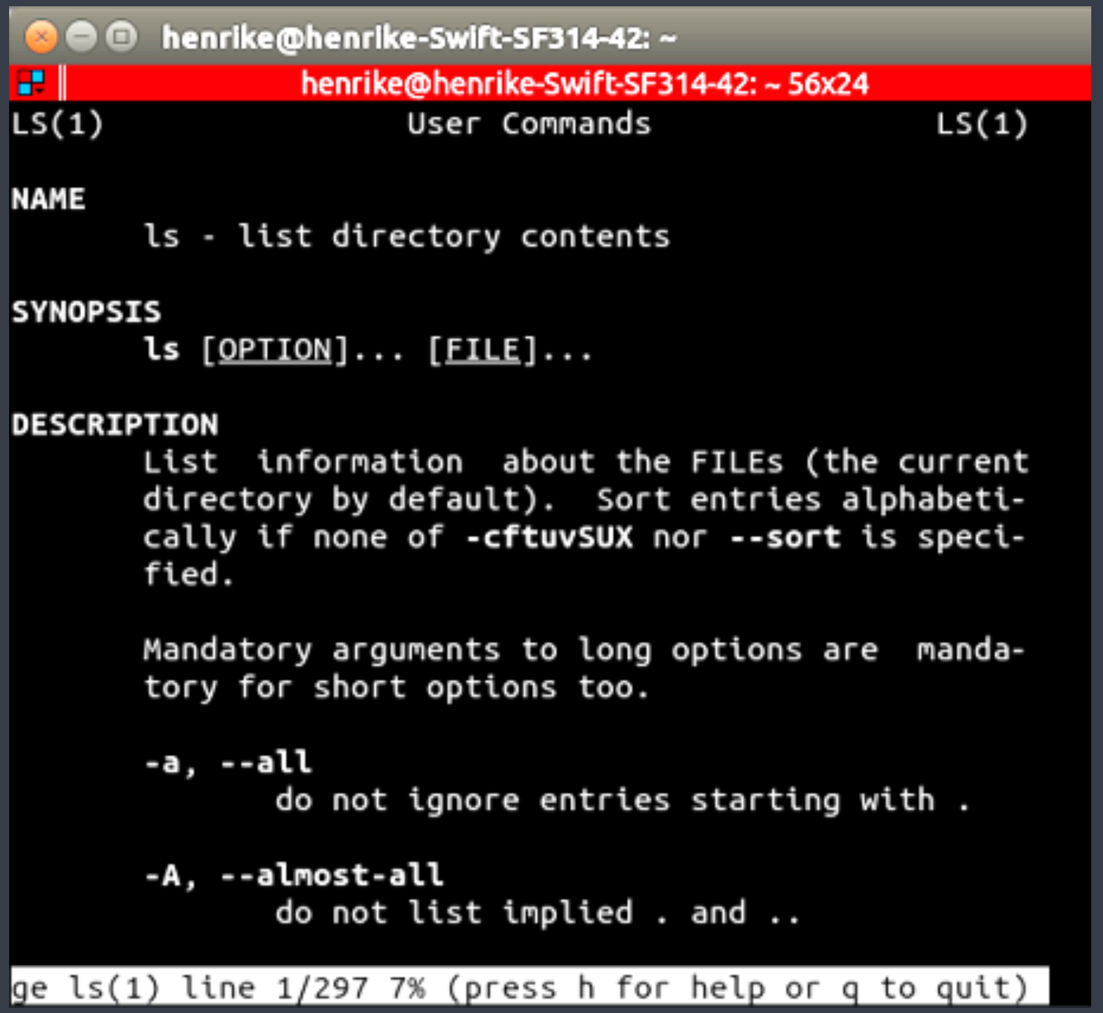
GETTING HELP

- Many commands have a help page, i.e. a **manual**. You can call it with **man**:

```
$ man ls
```

- The **man** page includes:
  - A description of what the command does
  - Explanation for arguments
  - Usage examples (the syntax of the command, what goes where)

- Commands that do not have a man page often have a help instead, option **(-)-help or (-)-h**



CHEAT SHEET 1

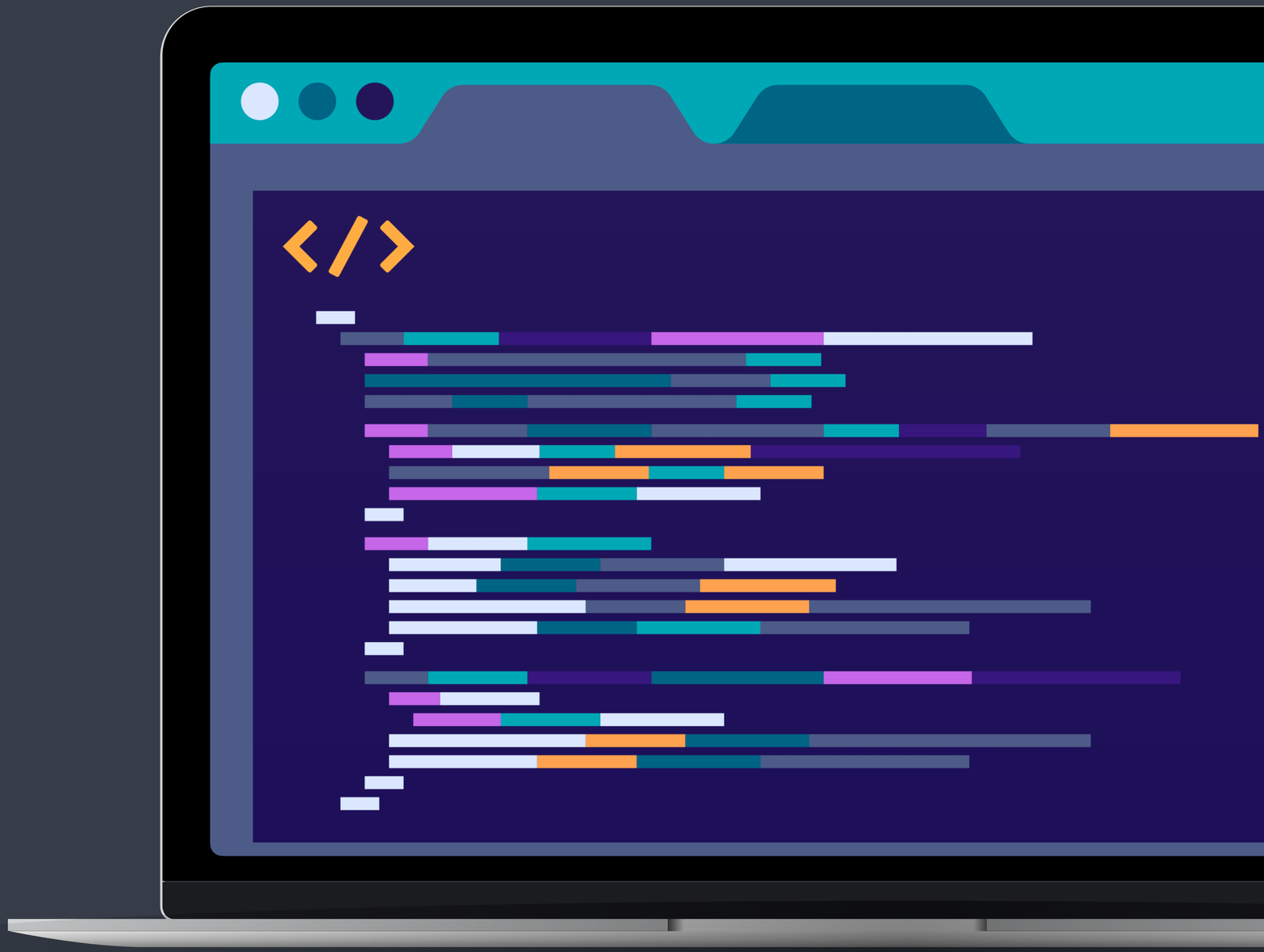
WHERE & WHAT

```
pwd # print working dir
cd # go to home dir
cd [path] # change dir (remember path)
ls # list dir content
man [cmd] # get info about command
[cmd] --help # view the help for command
```

FILE/DIR BASICS

```
rm [name] # remove file or dir
cp [name] # copy a file/dir
mv [name] [path] # move file/dir
```

2. PROJECT ORGANIZATION & BACKUP



IS YOUR COMPUTER A LAUNDRY BASKET?

Why should I care about directory structure and file naming?

- Not nice to not “work” in a mess.
- If your computer crashes it is MUCH easier to recover work if files/directories are structured.
- If you apply a command (bash, R, Python) to a group of files/directories, naming is important!
- Large files should be stored smartly to save space (storage & memory).

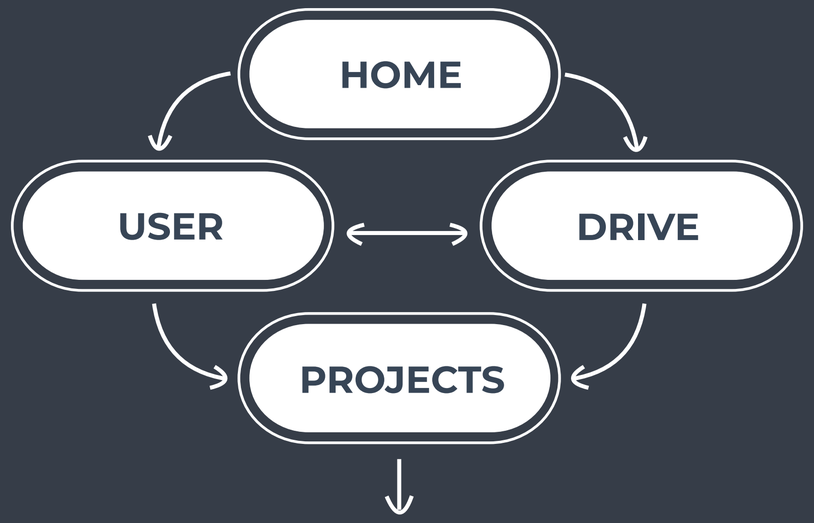
DO'S and DON'TS

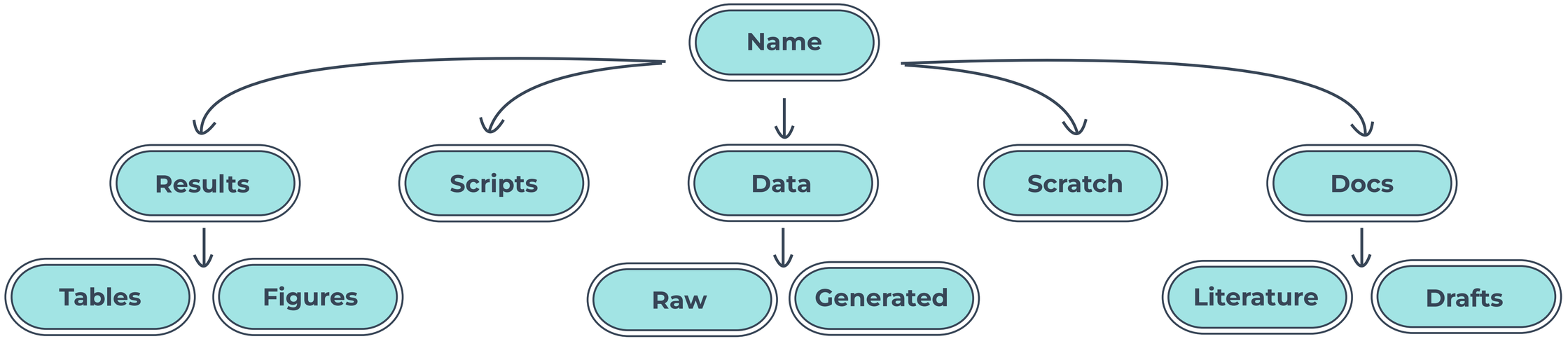
- ✓ Consistent and Logical Directory Structure
- ✓ Consistant & Non-redundant File Naming
- ✓ Version Control System git/GitHub & Back-up
- ✓ Clean and Update Your Computer Regularly

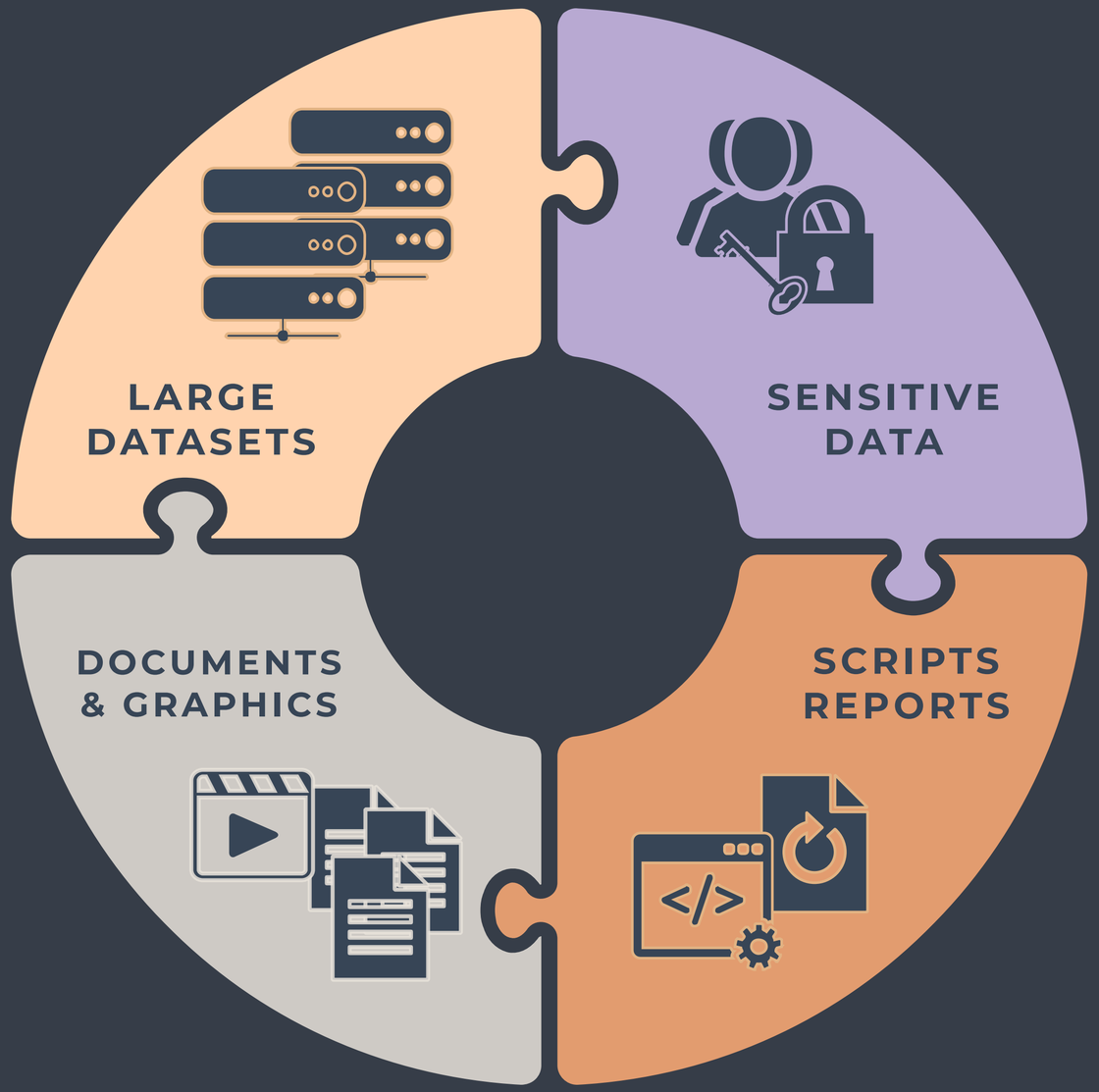
- ✗ Don't Use Symbols or Spaces in File/Directory Names
- ✗ Don't store multiple copies of file, instead point to it (path).
- ✗ Don't Store Large Data/Files on Your Local Computer
- ✗ Don't Store Sensitive Datasets on Your Computer

A SUGGESTION FOR STRUCTURE

LOCAL COMPUTER







STORAGE & BACKUP

- BIG DATA & SENSITIVE DATA:
  - Your own KU drives (S-drive)
  - ERDA & SIF (KU data storage)
  - Lab or department server
  - Lab or department cloud solution

- DOCX, EXCEL, POWERPOINT:
  - Google Drive
  - Dropbox
  - KU drives

- SCRIPTS:
  - **git/GitHub**
  - (KU drives)

GIT & GITHUB

Git is a version control software, created by Linus Torvalds for the management of the Linux kernel.

Other clients exist.

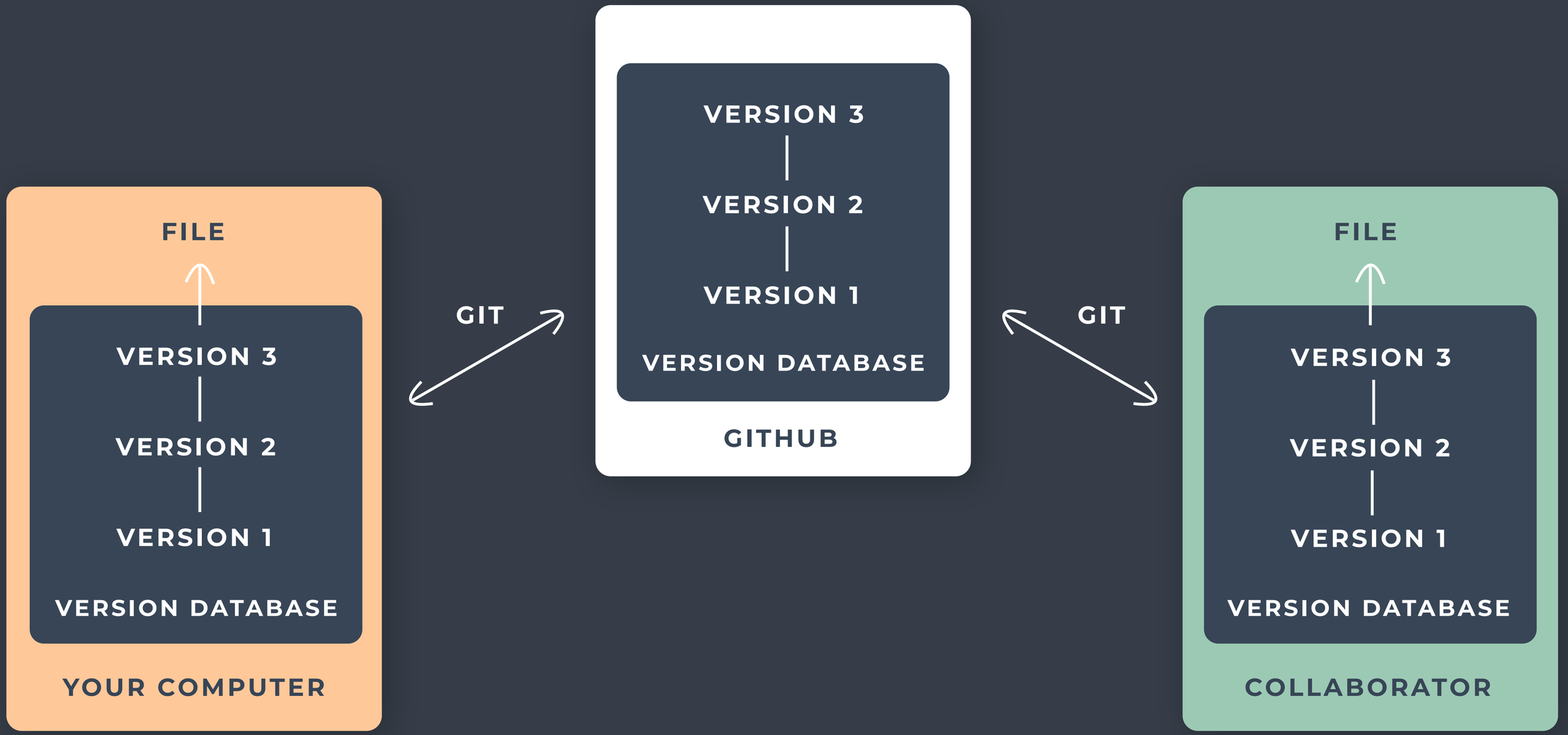
https://git-scm.com

GitHub is a web-based hosting service for version control that uses git

Other services exist.

https://github.com

WHAT IS A VERSION CONTROL SYSTEM?



GIT WORKS FROM THE COMMAND LINE

Git is used from the command line to edit directories & files in a version controlled way

A git command always begins with `git`

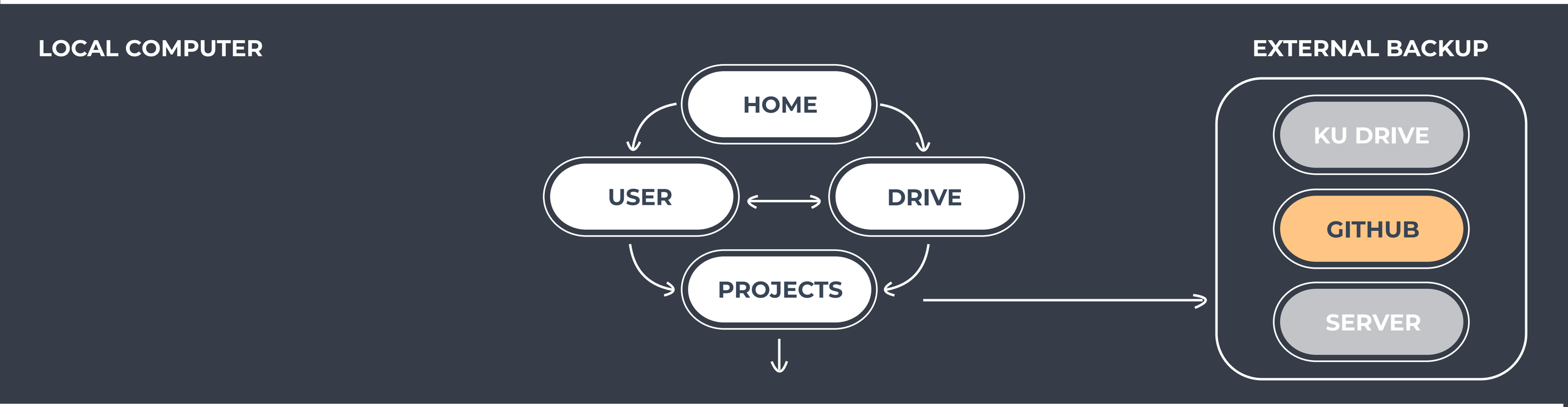
Git enables a local computer to interact with a cloud service (GitHub) to back up your work

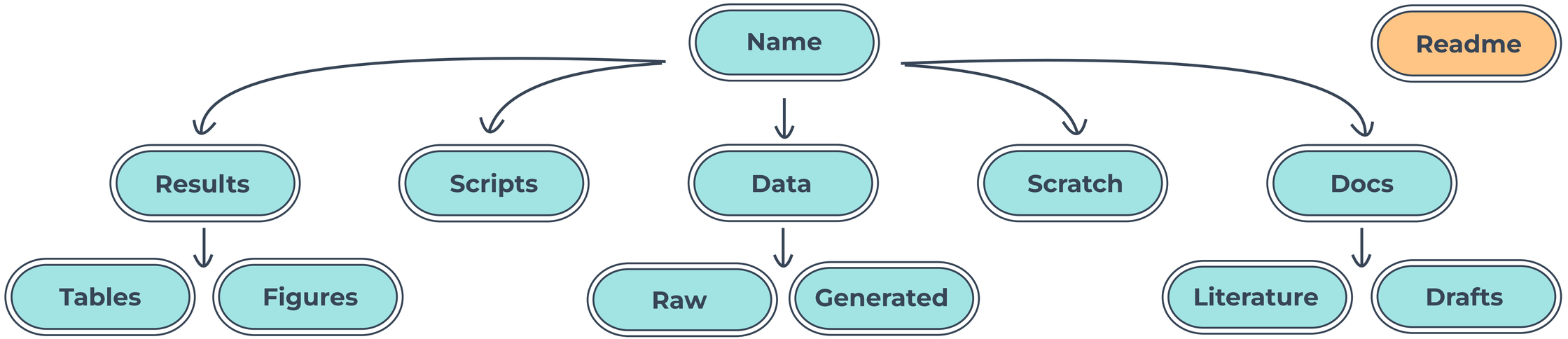
Sounds cool, yes!
Take our 'Introduction to git & GitHub' Workshop and learn how to git

```
git clone [your repository]

Username: [your user]
Password: [your password]
```

```
git status (status of the repo)

git add (add new files or changes)

git commit -m (commit the changes)

git push (push the changes to GitHub)
```

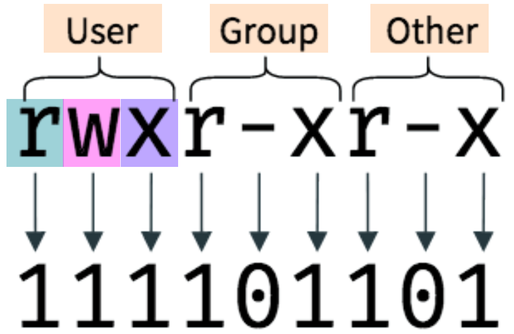
A SUGGESTION FOR STRUCTURE





USERS, GROUPS & PERMISSIONS

- Files & directories have **permission settings**
- Permissions denote who can **read**, **write** *(edit)* and **execute** a file/dir
- Permissions can be changed **IF** your are a system administrator (sys admin).
    - **Private computer :** You are sys admin
    - **KU computer :** You may be the system admin
    - **Shared KU drives :** You are not sys admin
    - **HPCs, Servers & Clouds :** You are not sys admin




```
chmod +rwx [filename]
chmod -wx [directoryname]

chmod u+rw [filename]
chmod g-w [filename]
chmod o-rw [filename]
```

CHEAT SHEET 1

WHERE & WHAT

```
pwd # print working dir
cd # go to home dir
cd [path] # change dir (remember path)
ls # list dir content
man [cmd] # get info about command
[cmd] --help # view the help for command
```

FILE/DIR BASICS

```
rm [name] # remove file or dir
cp [name] # copy a file/dir
mv [name] [path] # move file/dir

touch [name] # make a file
mkdir [name] # make a dir
```

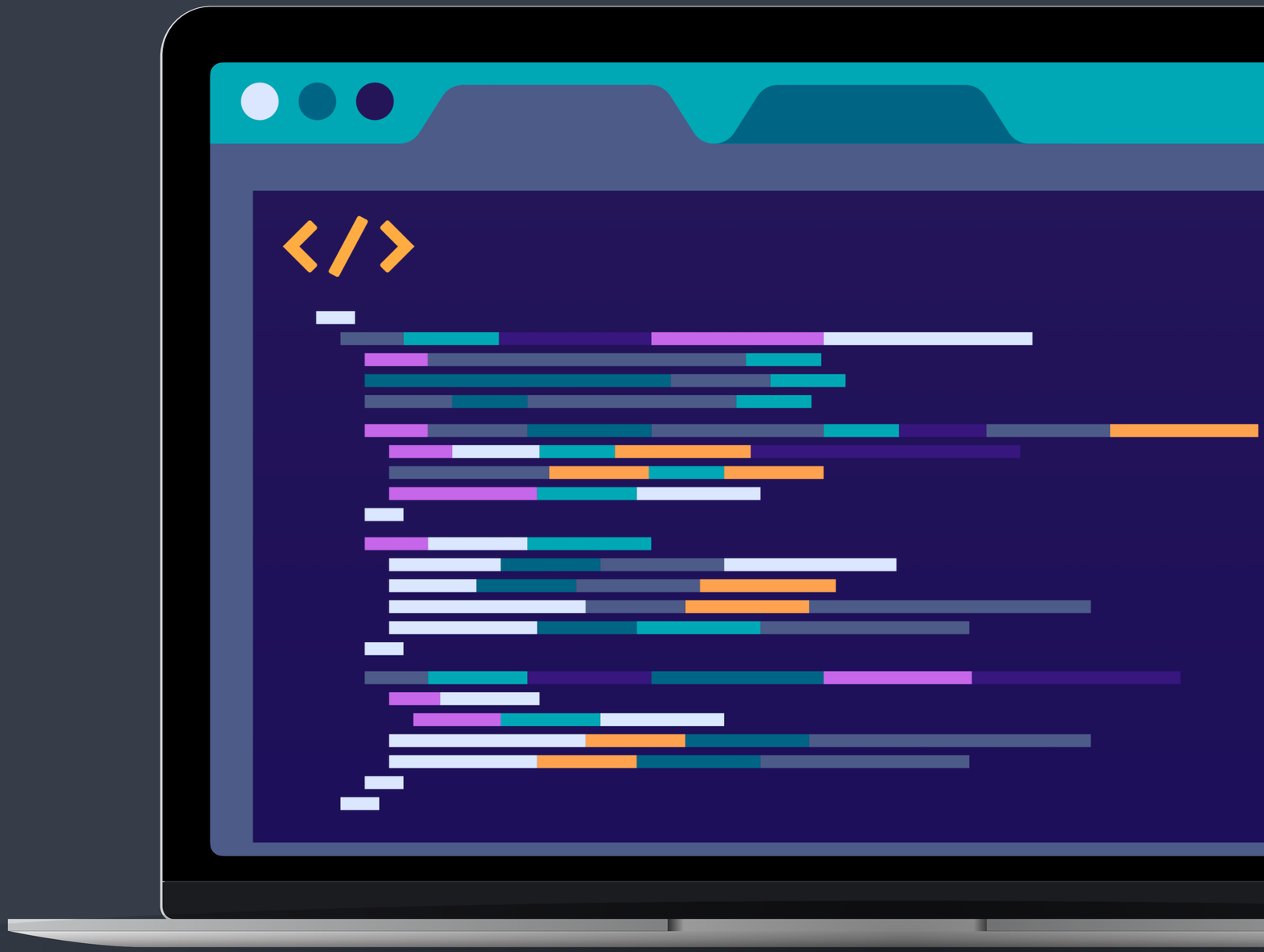
SPECIAL CHARACTERS

```
* # select everything
/ # forward slash paths
\ # escape character (don't use for now)
.. # one dir up/back
. # current dir
- # denotes a flag/argument
```

SIZE & PERMISSION

```
ls -lh * # all sizes
ls -lh [name] # file/dir size
du -sh # disk space
chmod +rwx [name] # add permission
chmod -rwx [name] # remove permission
chmod ugo+rw [name] # specify who, add permission
```

3. WORKING WITH FILES



VIEWING FILES

- Many roads lead to Rome… For viewing files we can use:

- **`less [file]`:**
- Prints N first lines of file. You can change the default N. Interactive viewing.
- Exit with **q**

- **`cat [file]`:**
- (concatenation), prints all lines of file.
- Exit with **ctrl + c**

- **`head/tail [file]`:**
- Prints **n first / last** lines of file. You can change the default **n**. Static viewing.

VIEW - SUBSET & RENAME

echo

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

---

less & cat

```
less myfile.txt
```

Patient	Age	Sex	Smoker	Grade
ID1	61	Female	No	G2
ID2	58	Female	Yes	G3
.	.	.	.	.

```
cat myfile.txt
```

---

head & tail

```
head -n 20 myfile.txt

head -n 20 myfile.txt > myfile_small.txt
```

> = Redirect - Save output to new file.

More on this later.

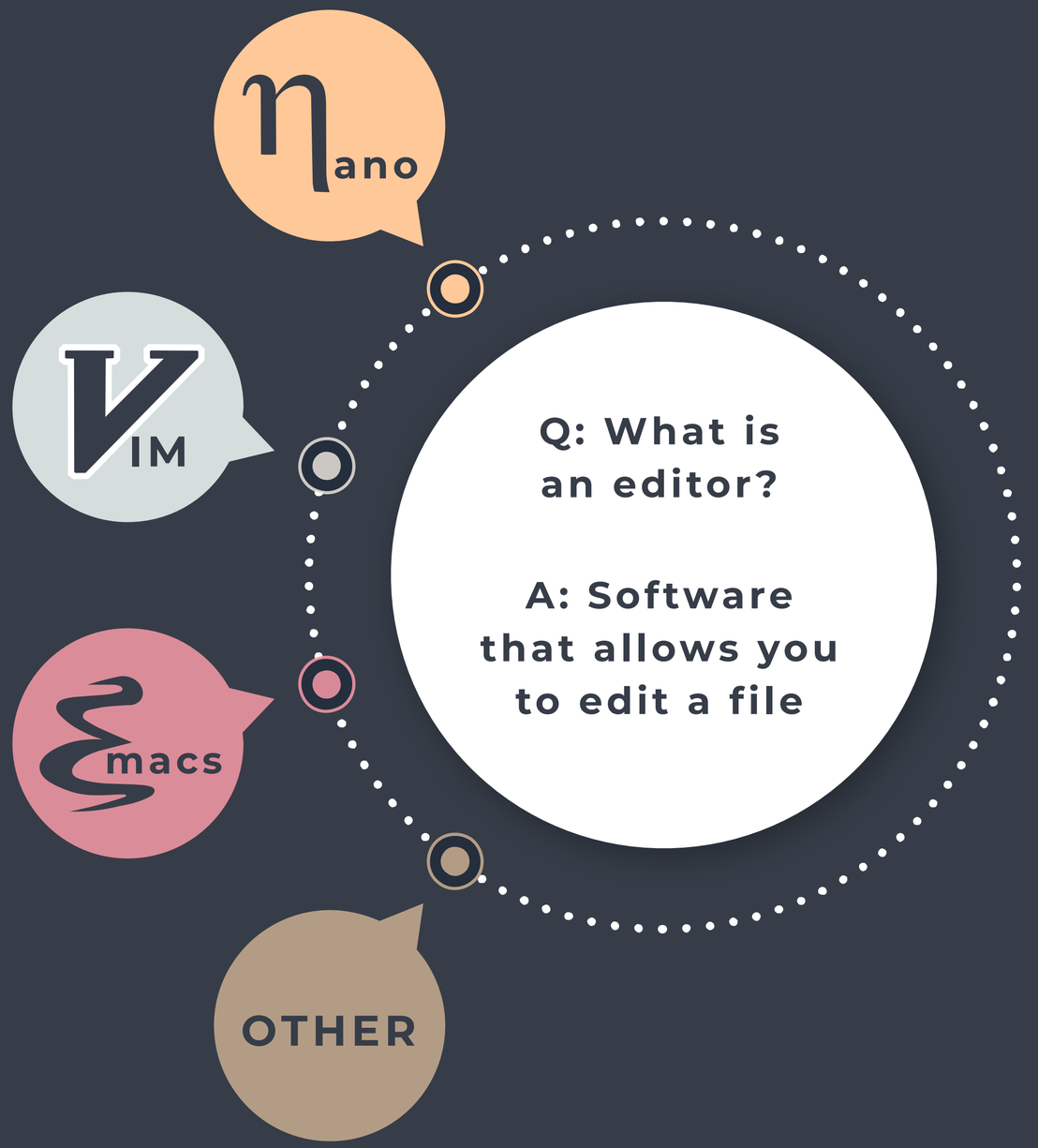
EDITING FILES

- **Nano** `[file]`:
  - Simple and easy to use
  - Limited functionality
  - Exit with **Ctrl + x (Y + enter)**

- **VIM or VI** `[file]`:
  - Many functionalities, i.e. complex
  - Keyboard shortcuts
  - Exit with **:q (:w or :q!)**

- **EMACS** `[file]`:
  - Oldest editor
  - Keyboard shortcuts, **Ctrl, Alt/Esc + [L]**
  - Exit with **Ctrl + x (Ctrl + s)**

- **OTHER** `[file]`:
  - There are many other editors …



COMPRESSED FILES

.GZ

Compresses a file with standard gzip (GNU zip) compression.

.TAR

Combines all files within a directory in one, single archive file.

.TAR.GZ

gzip and tar together. Gives a compressed archive of multiple files put together inside a single file.

```
gzip [file.gz]

gunzip -k [file.gz]

unzip [file.zip]

tar -cvf [file.tar.gz] [dir]

tar -xvf [file.tar.gz]

tar -tf [file.tar]
```

CHEAT SHEET 2

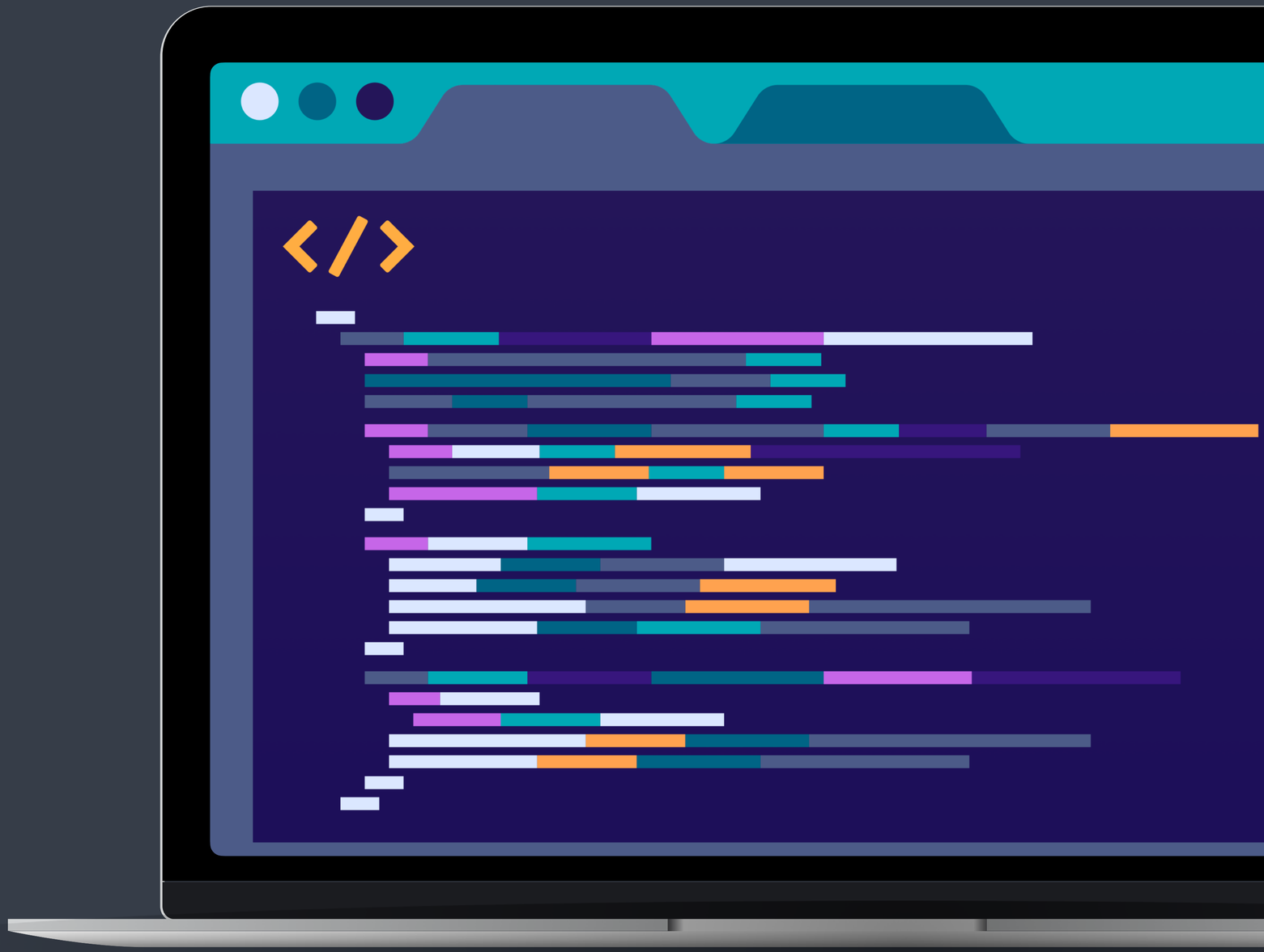
View Files

```
less [file] # view file content
cat [file] # view file content (full)
head / tail -n 10 [file] # view n first/last lines
nano [file] # https://www.nano-editor.org/dist/latest/cheatsheet.html
vim | vi [file] # https://vim.rtorr.com/
```

Compressed Files

```
tar -[f] [file] # .tar files
gzip -[f] [file] # decompress (.tar).gz files
unzip -[f] [file] # decompress .zip files
zless -[f][file] # view .gz file w/o decompression

others: zcat, zmore, gzcat
```

4. MORE BASH COMMANDS



FROM FILE NAVIGATION TO MANIPULATION

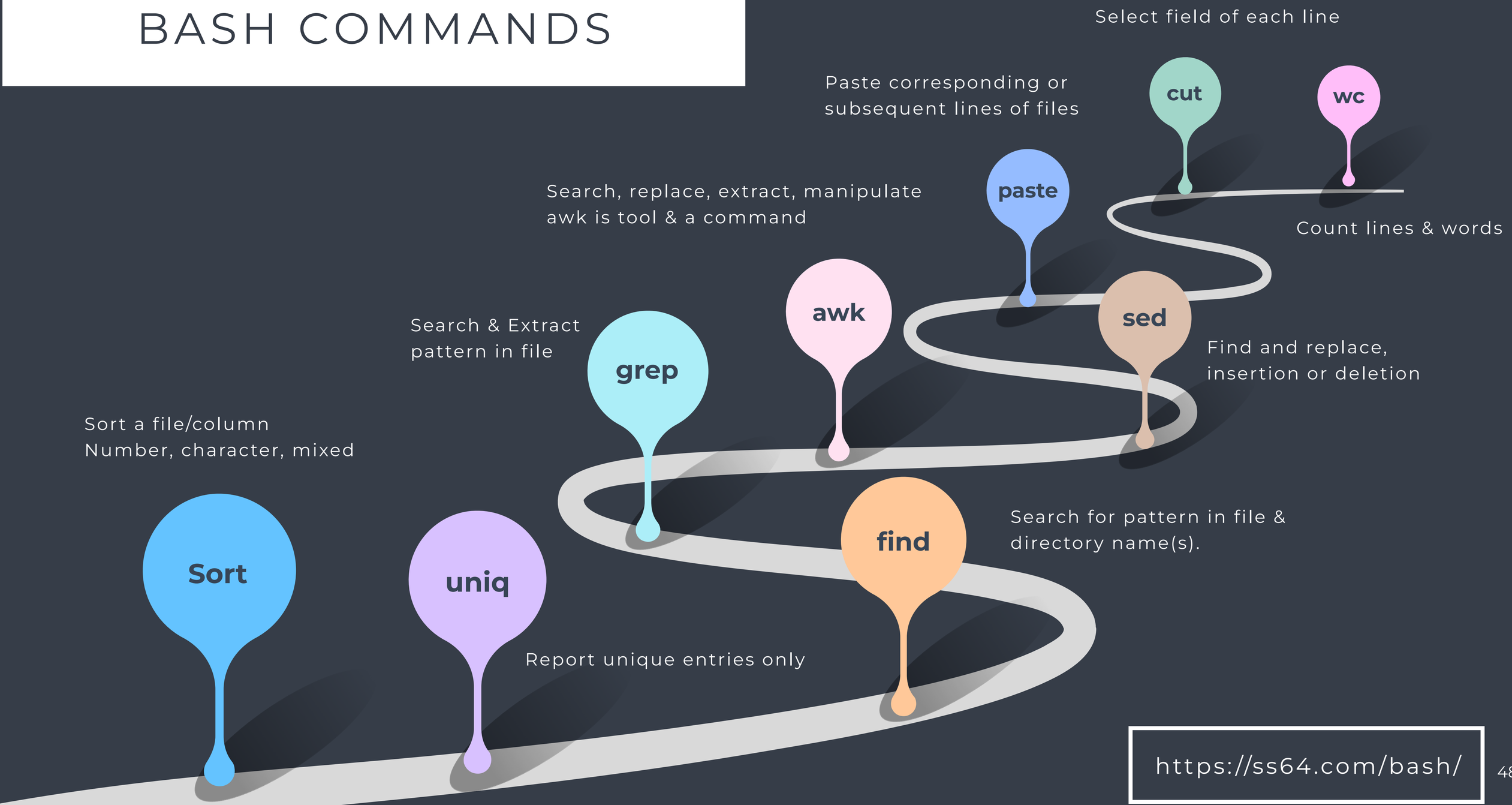
- **FILES & DIRECTORIES:**
  - **Move & Copy**
  - **Make & Remove**
  - **Open & Read**
  - **Edit & Save**
  - **Subset & Rename**
  - **View & Change Permissions**

NAVIGATE THE TERMINAL & FILE MANAGEMENT

- **FILES:**
  - **Sort**
  - **Count Lines & Entries**
  - **Merge & Concatenate**
  - **Find & Replace Patterns**
  - **Cut & Paste**
  - **Insert & Delete**

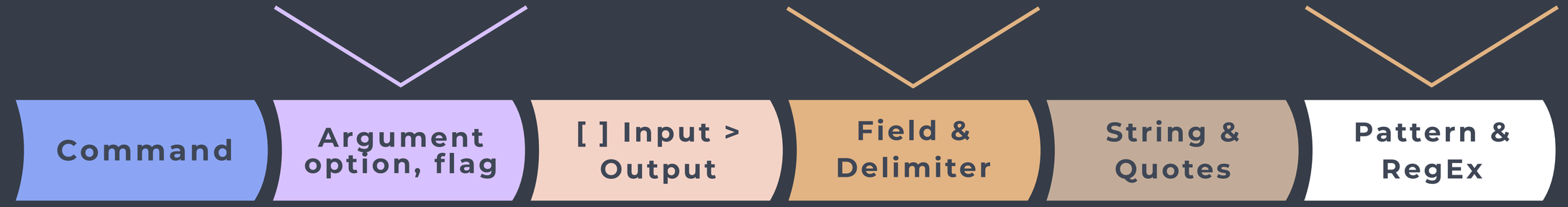
FILE MANIPULATION, WRANGLING, SUMMATION

BASH COMMANDS



https://ss64.com/bash/

A LITTLE TERMINOLOGY



Examples:
cd
touch
cp
less

What arguments does the command take?

Argument: what to act on
Option/Flag: how to act

Input = dir / file
[ ] = command -flag
> = from to (**redirect)**
Output = dir / file

name ; age ; height
Tom ; 31 ; 183
Julie ; 35 ; 168

Fields = 3
Delimiter = **;**

Some arguments are ***strings***, denoted with **“”** or **‘’**.

Regular expression (**RegEx**)= sequence of characters that specify a *flexible* search pattern

REGULAR EXPRESSIONS

Regular expression (RegEx) = sequence of characters that specify a *flexible* search pattern

Anchors

^	Start of string, or start of line in multi-line pattern
\A	Start of string
$	End of string, or end of line in multi-line pattern
\Z	End of string
\b	Word boundary
\B	Not word boundary
\<	Start of word
\>	End of word

Character Classes

\c	Control character

Quantifiers

*	0 or more	{3}	Exactly 3
+	1 or more	{3,}	3 or more
?	0 or 1	{3,5}	3, 4 or 5

Add a ? to a quantifier to make it ungreedy.

Escape Sequences

\	Escape following character
\Q	Begin literal sequence
\E	End literal sequence

"Escaping" is a way of treating characters which have a special meaning in regular expressions literally, rather than as special characters.

Groups and Ranges

.	Any character except new line (\n)
(a|b)	a or b
(...)	Group
(?:...)	Passive (non-capturing) group
[abc]	Range (a or b or c)
[^abc]	Not (a or b or c)
[a-q]	Lower case letter from a to q
[A-Q]	Upper case letter from A to Q
[0-7]	Digit from 0 to 7
\x	Group/subpattern number "x"

Ranges are inclusive.

https://cheatography.com/davechild/cheat-sheets/regular-expressions/

Regex Testers : https://regexr.com/ https://regex101.com/

CHEAT SHEET 2

View Files

less [file] # view file content
cat [file] # view file content (full)
head / tail -n 10 [file] # view n first/last lines
nano [file] # **https://www.nano-editor.org/dist/latest/cheatsheet.html**
vim [file] # **https://vim.rtorr.com/**

Compressed Files

tar -[f] [file] # .tar files
gzip -[f] [file] # decompress (.tar).gz files
unzip -[f] [file] # decompress .zip files
zless -[f][file] # view .gz file w/o decompression

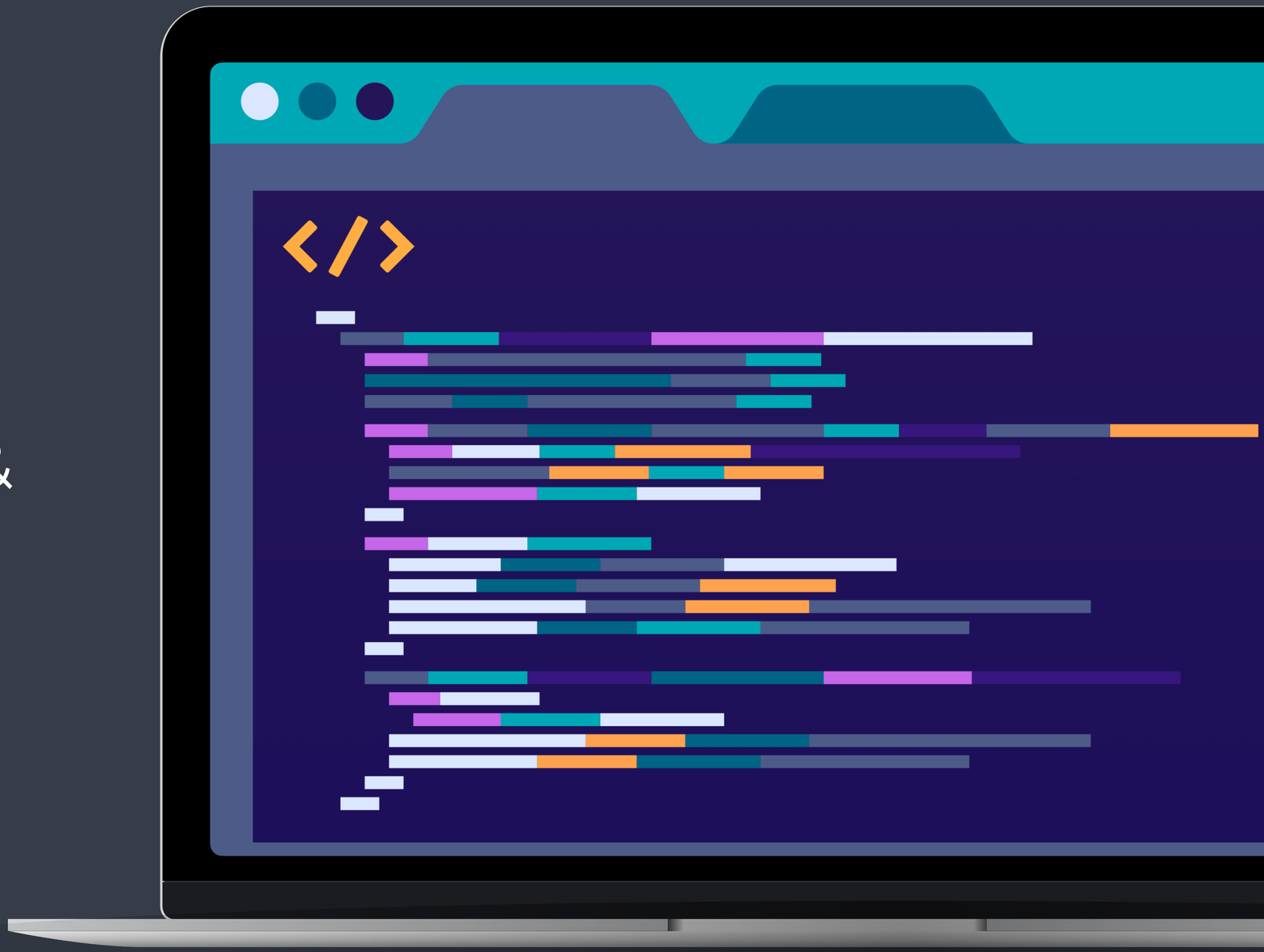
others: zcat, zmore, gzcat

Manipulating Files

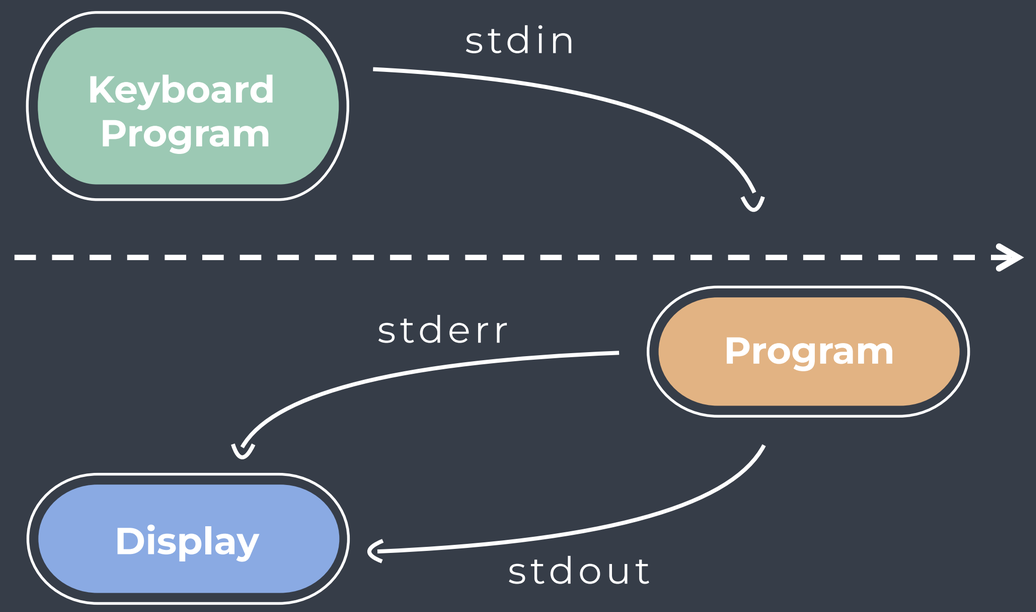
wc -[f][file] # Count lines, characters, bits
sort -[f][file] # Sort file (by field/column)
uniq -[f][file] # Return unique values
cut -[f][file]: # Extract field/column
paste -[f][files]: # Merge file lines

sed -[f]'command'[file] # Insertion, deletion, …
grep -[f]['pattern'][file] # Search for pattern
awk '{pattern}'[file] # Search, replace, extract, …
find -[f][path]['pattern'] # Search pattern in file name

5. REDIRECTION & PIPES

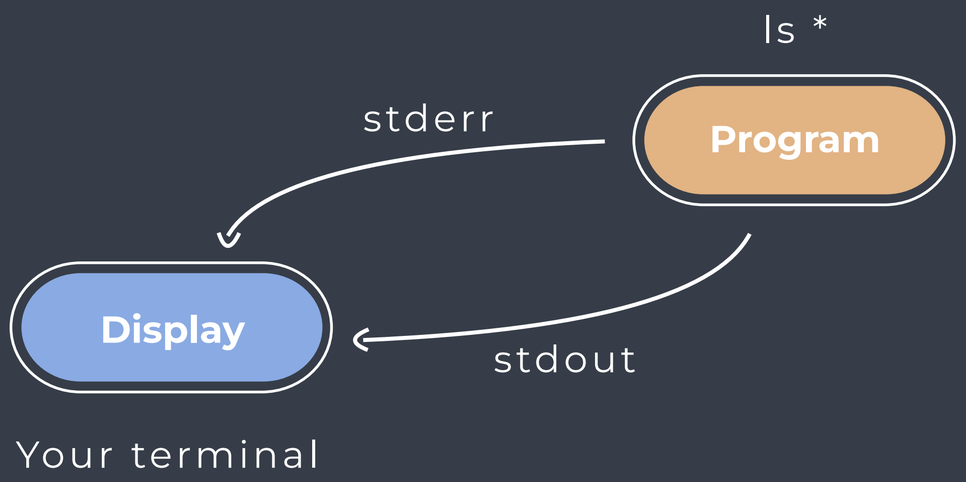


A TALE OF THREE STREAMS



- Data in unix always moves along one of **three (data) streams**:
  - **stdin** (Standard In) – the stream along which input to commands moves
  - **stdout** (Standard out) - the output of commands moves along this stream
  - **stderr** (Standard error) – error messages move via this stream
- Each of the streams can be redirected separately.

A TALE OF THREE STREAMS

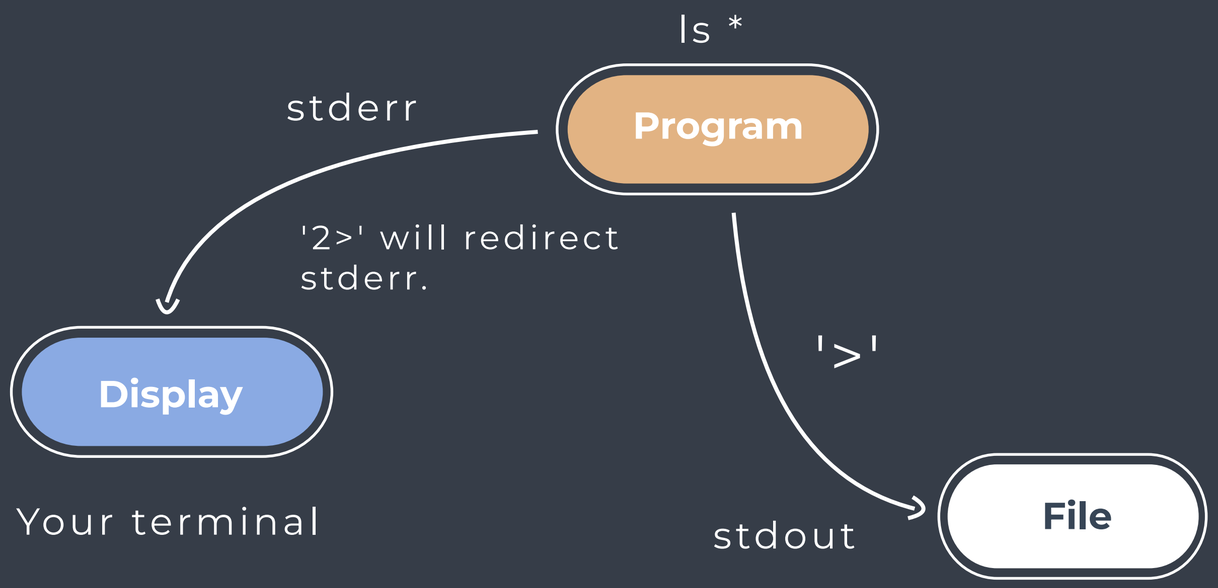



```
BashFigure1.jpeg
Commandline_Workshop.pptx
Data
Examples
Just_Bash_It_Exercises.md
Just_Bash_It_Exercises.Rmd
Just_Bash_It_Presentations.md
Just_Bash_It_Presentations.Rmd
LICENSE
README.md
```

- **stdout** (standard out) - the output of commands moves along this stream
- **stderr** (standard error) – error messages move via this stream
- By default **stdout** points to the display, your terminal window

A TALE OF THREE STREAMS

But **stdout** and **stderr** can be **redirected** to i.e. a file:



Redirect stdout:

```
$ ls > list_results.txt
```

Redirect stderr:

```
$ ls 2> err.txt
```

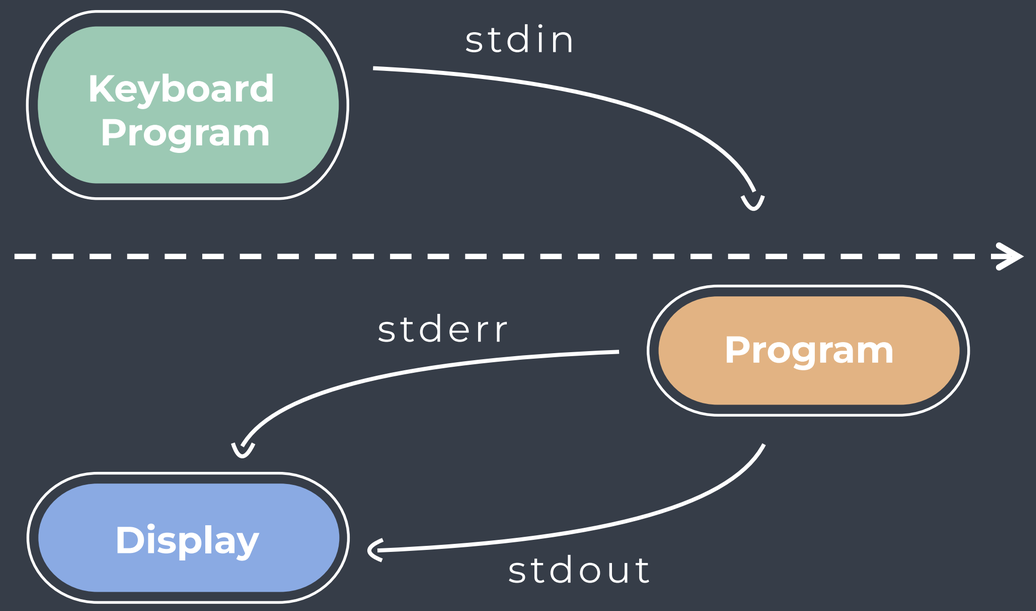
Redirect both into different files:

```
$ ls 2> err.txt > list_results.txt
```

Redirect both into the same file:

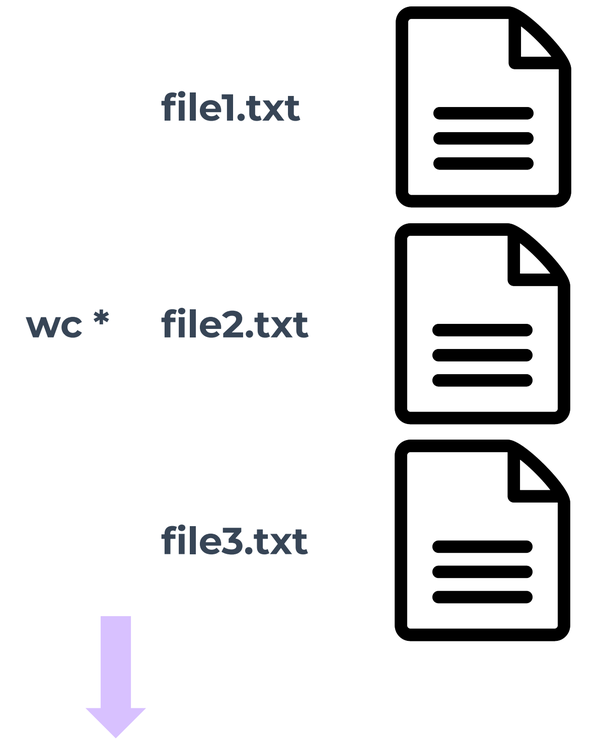
```
$ ls &> both
```

A TALE OF THREE STREAMS



- Many commands require input data to work on, i.e. **wc** works on a file.
- Usually input data comes from the keyboard, i.e. the name of the file to word count.
- But it can also come directly from another program!

CHAINING (PIPING) COMMANDS

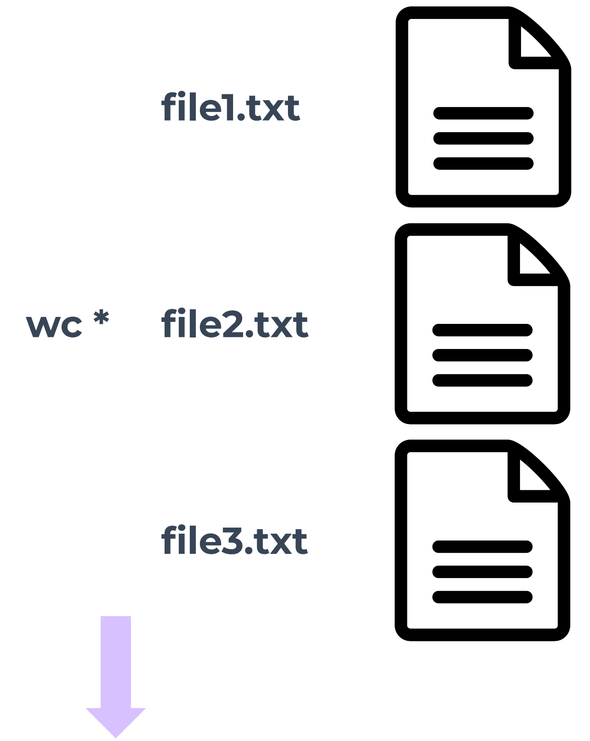



```
$ wc *.txt > file_lengths
```

```
$ sort -n file_lengths
```

But now we have an intermediate file we don't need!

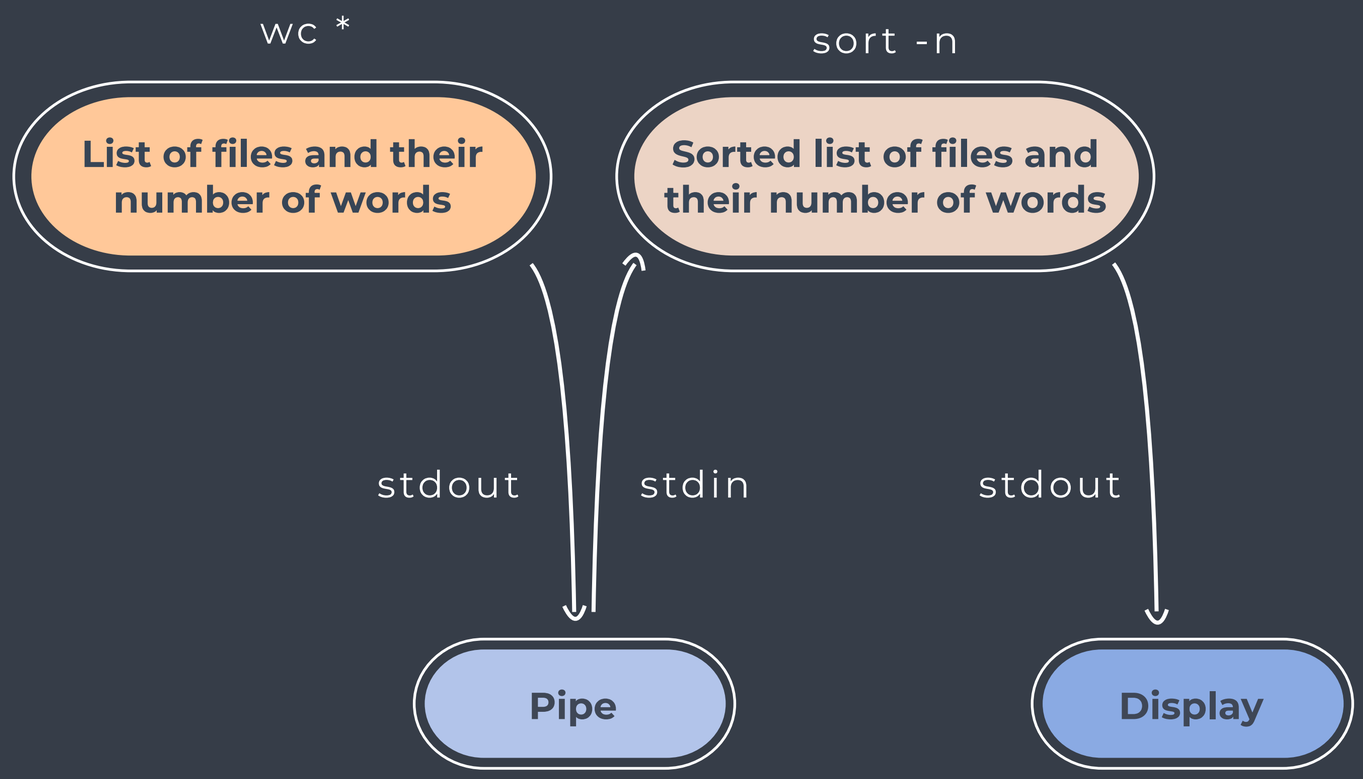
CHAINING (PIPING) COMMANDS



Instead we can use a pipe:

```
$ wc *.txt | sort -n
```

CHAINING (PIPING) COMMANDS



pipe symbol = |

```
$ wc *.txt | sort -n
```

CHAINING (PIPING) COMMANDS

Some examples:

- Take the second column and count how often each element occurs:

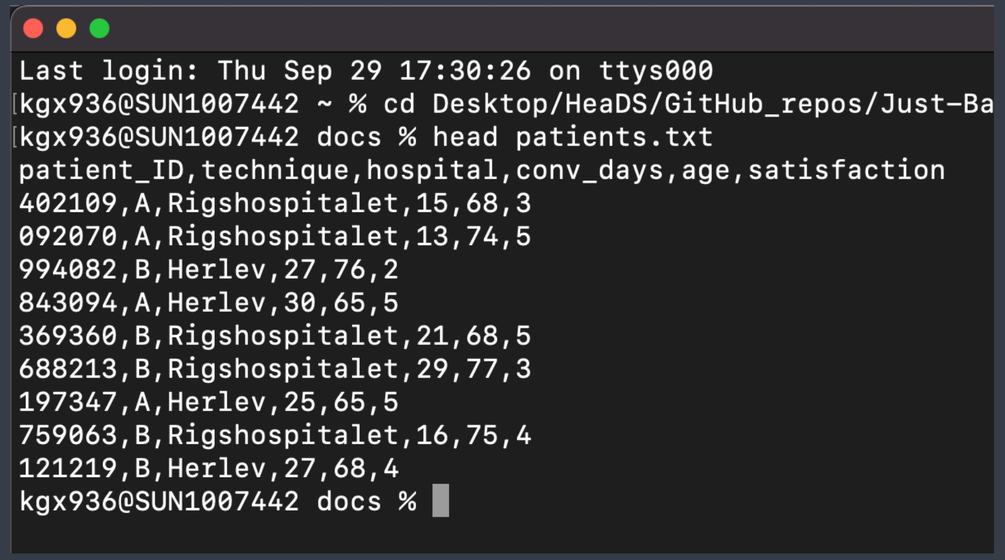
```
$ cut -d ',' -f 2 patients.txt | sort | uniq -c
```

- Get only lines containing 'Herlev' and sort by age:

```
$ grep 'Herlev' patients.txt | sort -t ',' -n -k5
```

- If we don’t want the entire line we could also cut out the age column before sorting:

```
$ grep 'Herlev' patients.txt | cut -d ',' -f 5 | sort -n
```



CHEAT SHEET 2

View Files

less [file] # view file content
cat [file] # view file content (full)
head / tail -n 10 [file] # view n first/last lines
nano [file] # **https://www.nano-editor.org/dist/latest/cheatsheet.html**
vim [file] # **https://vim.rtorr.com/**

Compressed Files

tar -[f] [file] # .tar files
gzip -[f] [file] # decompress (.tar).gz files
unzip -[f] [file] # decompress .zip files
zless -[f][file] # view .gz file w/o decompression

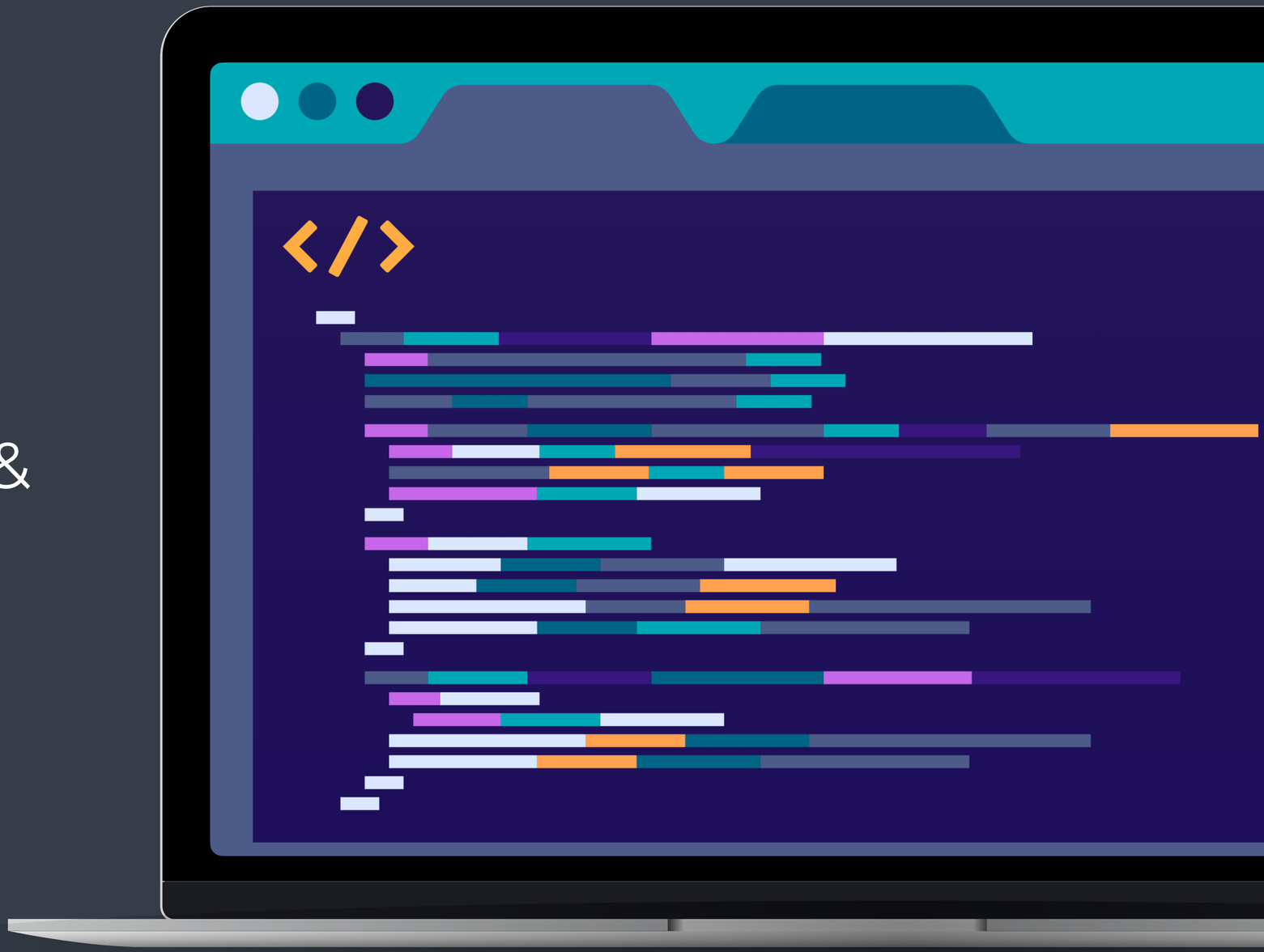
others: zcat, zmore, gzcat

Manipulating Files

wc -[f][file] # Count lines, characters, bits
sort -[f][file] # Sort file (by field/column)
uniq -[f][file] # Return unique values
cut -[f][file]: # Extract field/column
paste -[f][files]: # Merge file lines

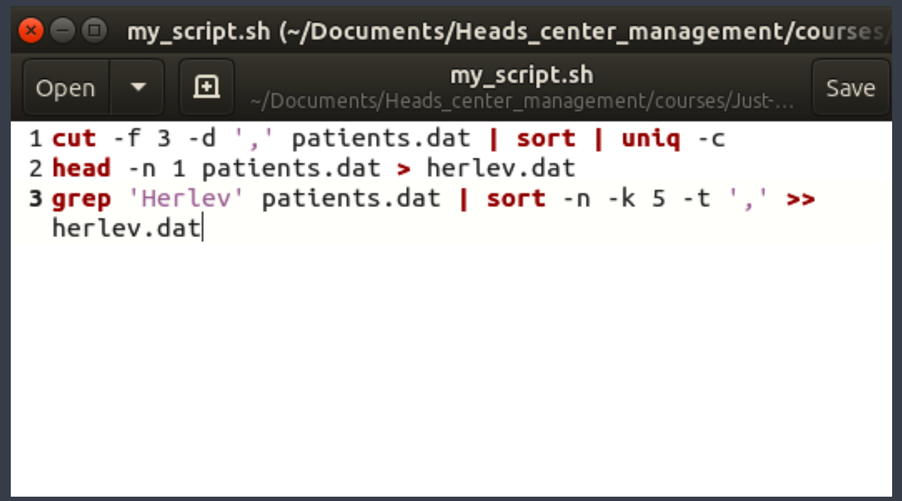
sed -[f]'command'[file] # Insertion, deletion, …
grep -[f]['pattern'][file] # Search for pattern
awk '{pattern}'[file] # Search, replace, extract, …
find -[f][path]['pattern'] # Search pattern in file name

6. SHELL SCRIPTS & LOOPS



WRITING SCRIPTS

```
(base) henrike@henrike-ThinkPad-X390:~/Documents/Heads_center_managem
ent/courses/Just-Bash-It/Examples/docs$ cut -f 3 -d ',' patients.dat
| sort | uniq -c
      1 herlev
      9 Herlev
      1 hospital
     10 Rigshospitalet
(base) henrike@henrike-ThinkPad-X390:~/Documents/Heads_center_managem
ent/courses/Just-Bash-It/Examples/docs$ head -n 1 patients.dat > herl
ev.dat
(base) henrike@henrike-ThinkPad-X390:~/Documents/Heads_center_managem
ent/courses/Just-Bash-It/Examples/docs$ grep 'Herlev' patients.dat |
sort -n -k 5 -t ',' >> herlev.dat
```



- Save your commands for later use!
- Re-run anytime, always same result
- Check your script if you don't remember the steps of an analysis
- Back-up your scripts on i.e. github, KU drive, ect.

```
(base) henrike@henrike-ThinkPad-X390:~/Documents/Heads_center_managem
ent/courses/Just-Bash-It/Examples/docs$ bash my_script.sh
      1 herlev
      9 Herlev
      1 hospital
     10 Rigshospitalet
(base) henrike@henrike-ThinkPad-X390:~/Documents/Heads_center_managem
ent/courses/Just-Bash-It/Examples/docs$
```

COMMAND LINE INPUT

- Add **arguments** when you execute your script, these will be passed to it.
- Useful if you want to i.e. specify the file you want to run the script on.
- A script is called with ***bash*** (or simply ***sh***).
- The **first argument** after the script name will be **$1** inside the script.
- The **second argument** will be **$2** and so on.

```
# Comment line script 1
# Usage: sort_this.sh file_name

sort $1 -n
```

```
$ bash sort_this.sh patients.txt
```

```
$ bash sort_this.sh my_textfile.txt
```

```
# Comment line script 2
# Usage: cut_col file_name column_number

cut -d ',' -f $2 $1
```

```
$ bash cut_col patients.txt 3
```

LOOP IN BASH

```
#example of a for loop in bash

for file in *.txt
do
      echo $file
      wc $file
done


#the general syntax of a for loop is:

for [iterator] in [generator]
do
      commands
done
```

- Loops allow you to repeat the same action several times, once for each file for example.
- Many bash commands can be run on several files, i.e.

```
$ wc *.txt
```

- However, a loop is useful if you want to execute several commands for each file.

OTHER SCRIPTING

```
$ bash my_script.sh
```

- This is a bash script. It's basically just a text file but we tell the computer to use bash to interpret it by calling it with 'bash my_script.sh'
- We can also run scripts in other languages from the command line, i.e. python:

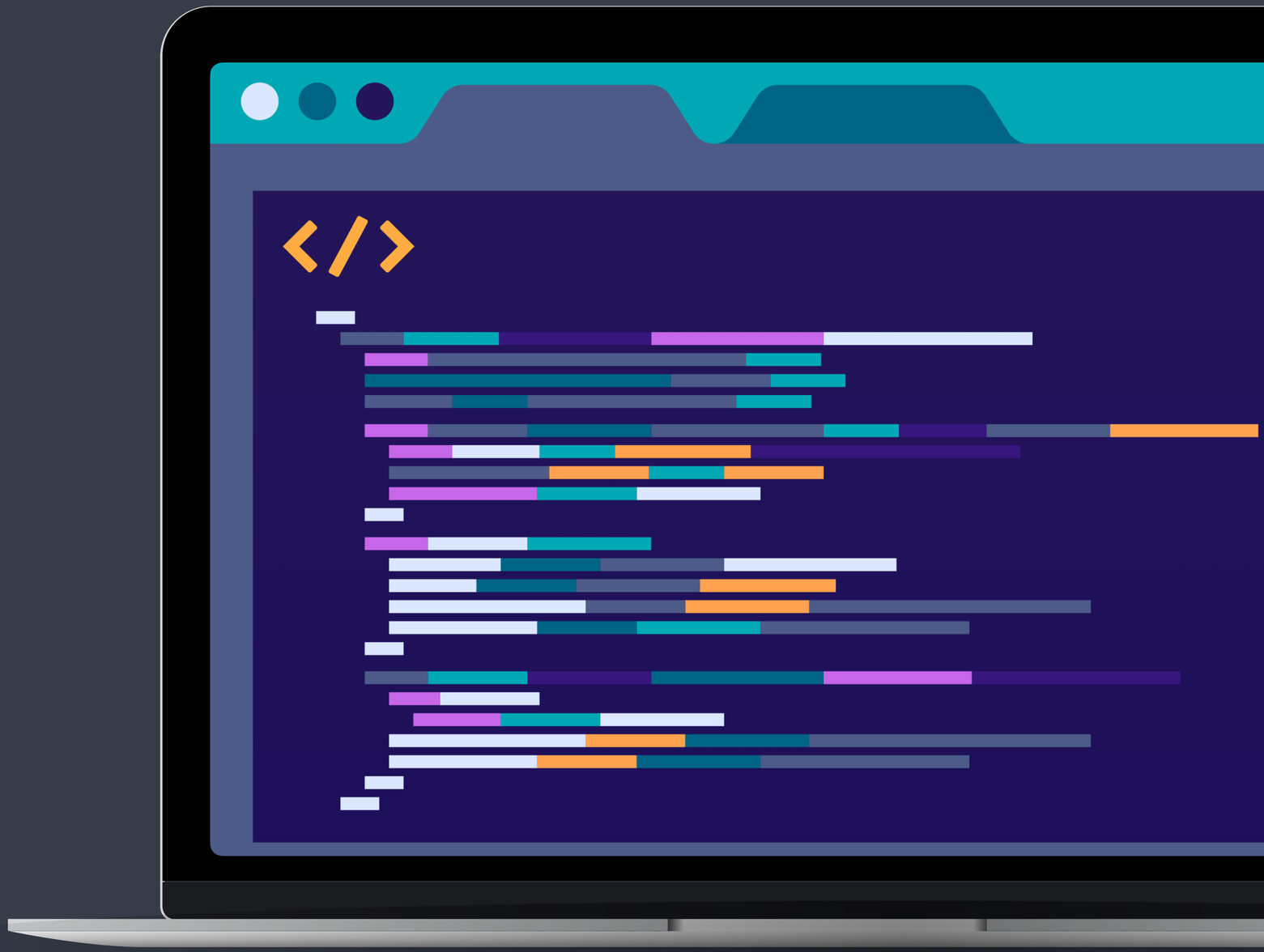
```
$ python align_reads.py
```

- Scripts in the R language are called with the command ‘Rscript':

```
$ Rscript deSEQ_cancer_Data.R
```

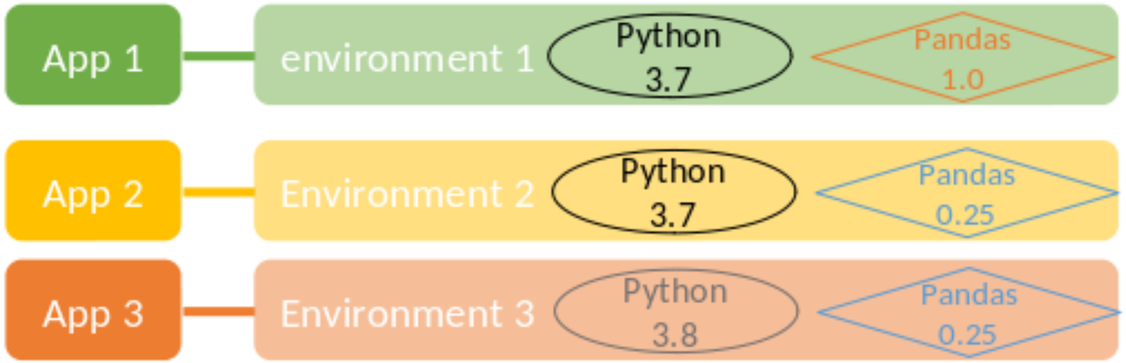
- You need to have R and python installed on your computer in order to be able to call them on the command line.

7. SOFTWARE INSTALLATION UPKEEP & MORE



CONDA®

- **Conda** is an open source package & environment management system.
- Many softwares exist as conda packages, you can use conda to install them and their dependencies.
- **An environment** is a specific combination of packages in a specific version.
- IF you do not update the packages, running an analysis in the same environment will give the same result.



PROS

Comprehensive, many options

Environment management

Well documented

CONS

Can be overwhelming

Homebrew

- **Homebrew** - source software package management system, for OSX, Ubuntu (Linux).
- NOT an environment manager, package manager only.
- Creates separate directories for libs and configurations, adds symlinks to *user/local*.
- Manage all installed softwares via the terminal, install, check, update, remove…

PROS

Easy to use

Good documentation

Commands a few and logical

CONS

Packages, NOT environments

APT-GET

- **apt-get** is a command-line tool for handling packages in Linux (Ubuntu, *'Windows'*).
- APT = *Advanced Packaging Tool*
- Upgrade and remove of packages along with their dependencies.
- NOT an environment manager, package manager only.

PROS

Works like any linux command

No installation (is default)

Robust to OS updates

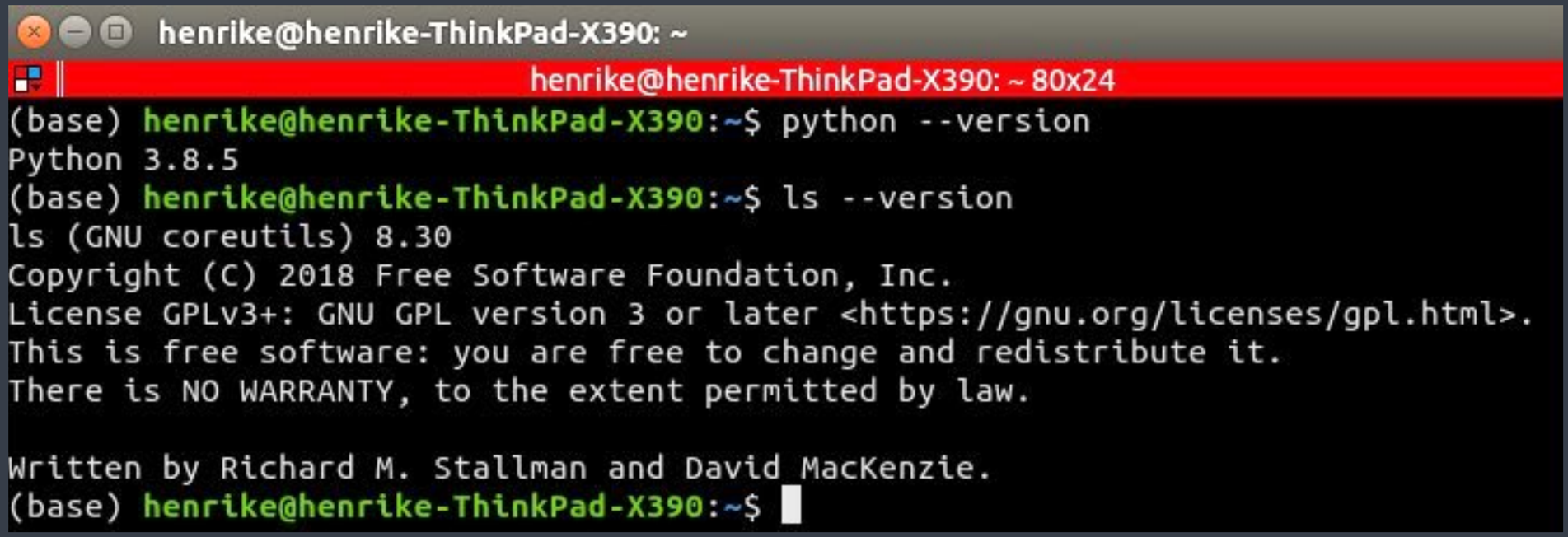
CONS

Packages, NOT environments

SOFTWARE VERSION

You can generally check the version of installed software with:

```
[name of software] --version
```



KEEPING THINGS UP TO DATE

Update your software with the same tool you used to install it:

- Installed with **conda**: (update only the current environment)

```
$ conda update [software]
```

- Installed with **homebrew** (updates a software):

```
$ brew upgrade [software]
```

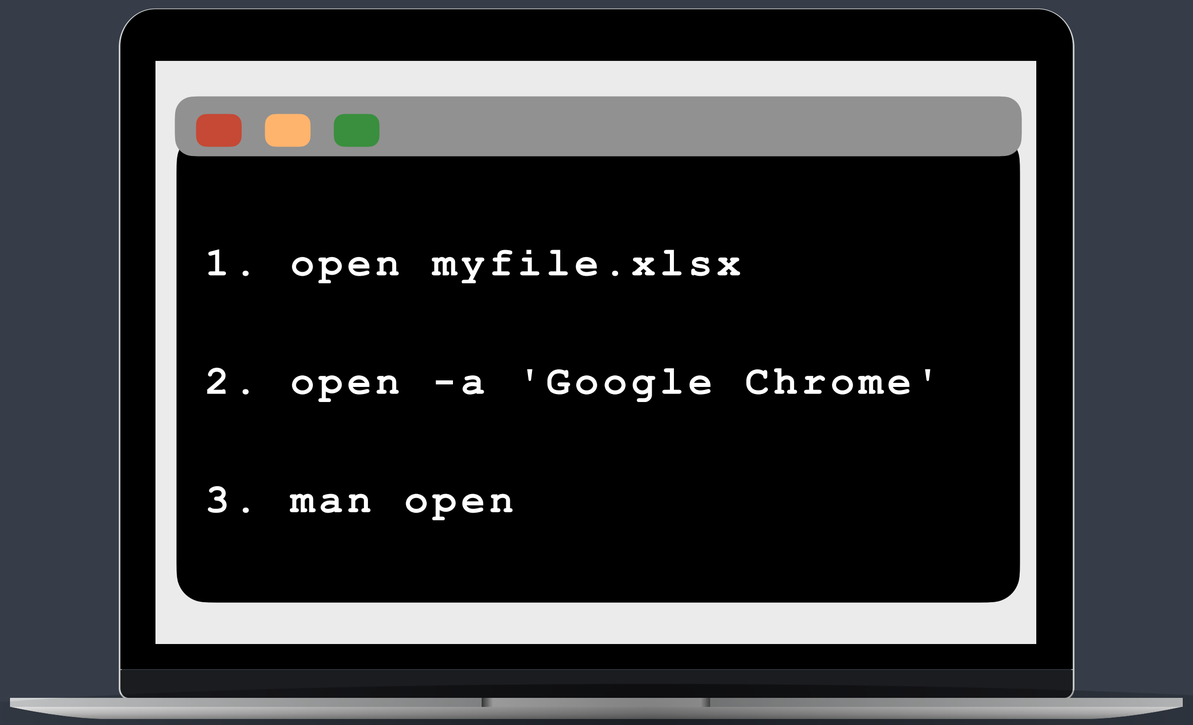
- Installed with **apt-get**:

```
$ apt-get update [software]
```

- With **Windows App store**:

Update via Windows App store

OPENING APPS & RUNNING SOFTWARE



1. Some, but not all **files** can be **opened** without specifying an app.

2. Many **apps** can be **launched** with open -a.

3. Some **softwares** are designed to run on both the command line and in a GUI, while others are specific to one.

Monitor processes, find them, check memory & 'kill' them when necessary:

```
$ top
$ ps aux (processes for all users)
$ kill [pid]
```

CONFIGURATION FILES

- Config files are used to specify parameters, options, settings and preferences applied to your OS or a software.
- Types of config files; system-wide, program-specific, user-specific.
- Extension will pertain to what is configured.
- Rarely permission to change system-wide files. Software-specific files you can usually edit.
- **N.B** config files are ‘hidden’, i.e. ls -la (or similar) to see them.

BASH SHELL

/etc	/User
.profile	***.profile***
.bashrc	***.bashrc***
.bash_profile	*.bash_profile*

Z SHELL (BASH +)

/etc	/User
.zprofile	***.zprofile***
.zshrc	***.zshrc***
.zsh_profile	*.zsh_profile*

https://landoflinux.com/linux_bash_configuration_files.html

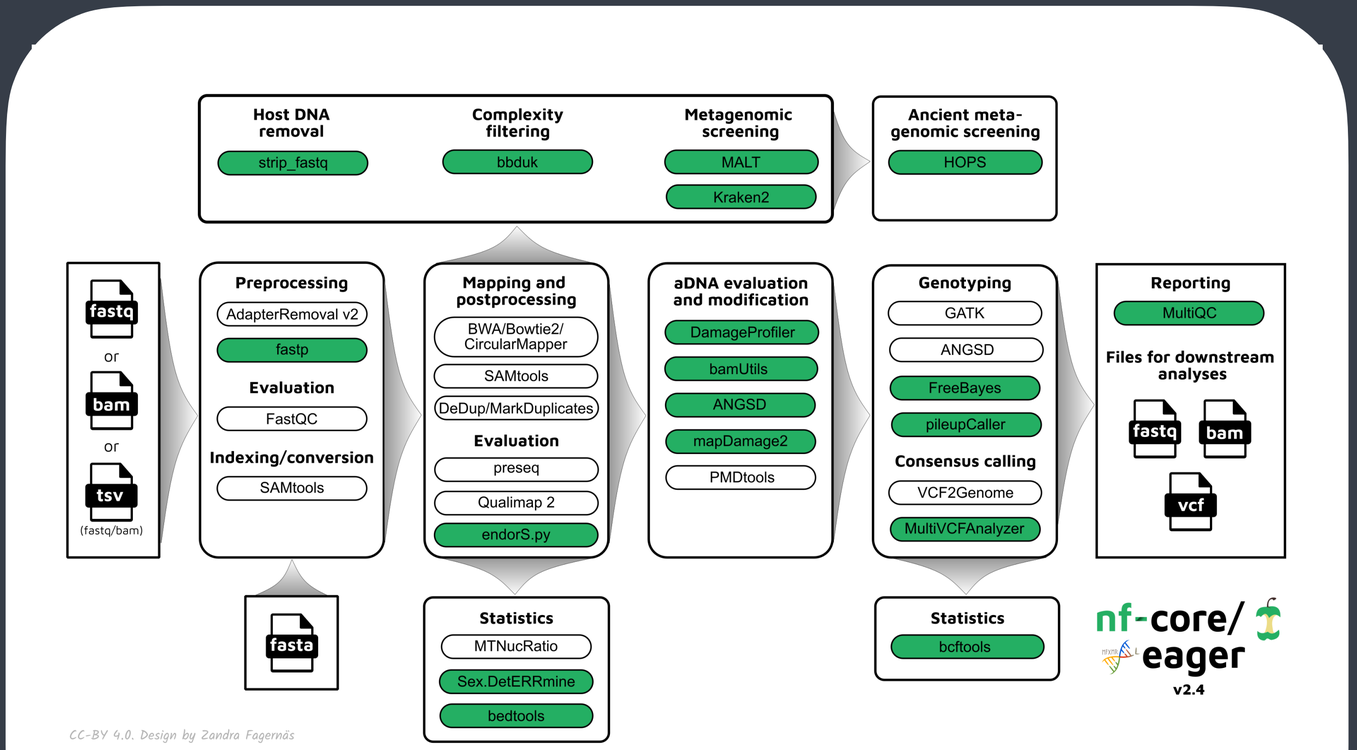
WORKFLOW LANGUAGES

- **Workflow languages:**
  - Tools to create reproducible pipelines - more robust than bash scripts.

- Workflow languages allow you to:
  - Control the flow of your pipeline
  - Restart at various defined points.
  - Ensure previous commands have. executed without error.
  - Logging & Reporting of your job.

snakemake

https://snakemake.readthedocs.io/en/stable/



nextflow

https://www.nextflow.io/

THANK YOU FOR TODAY