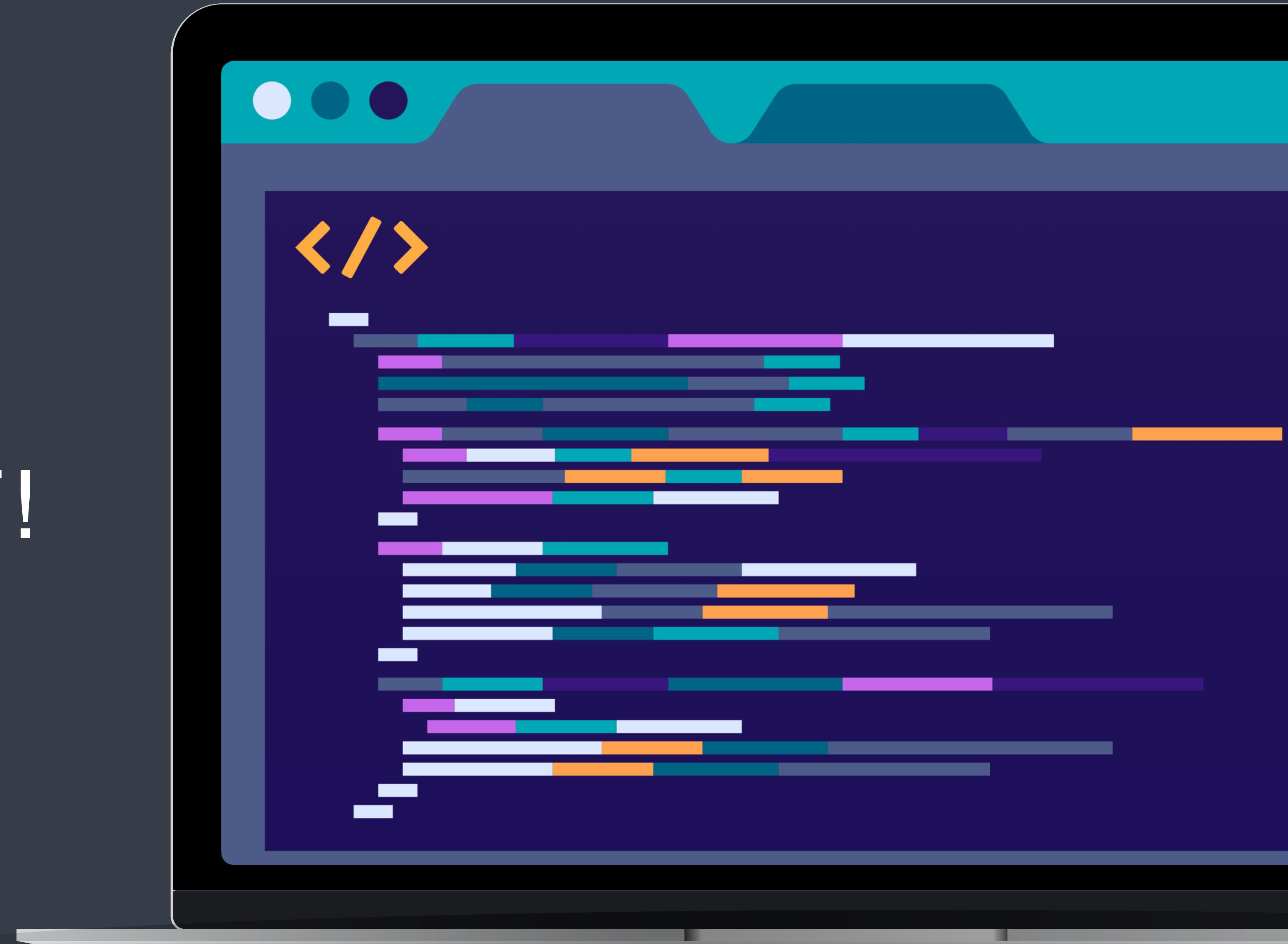
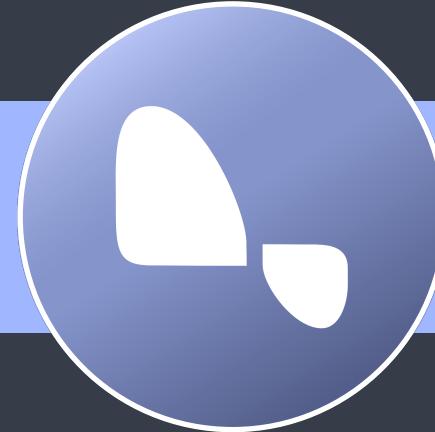


JUST BASH IT!

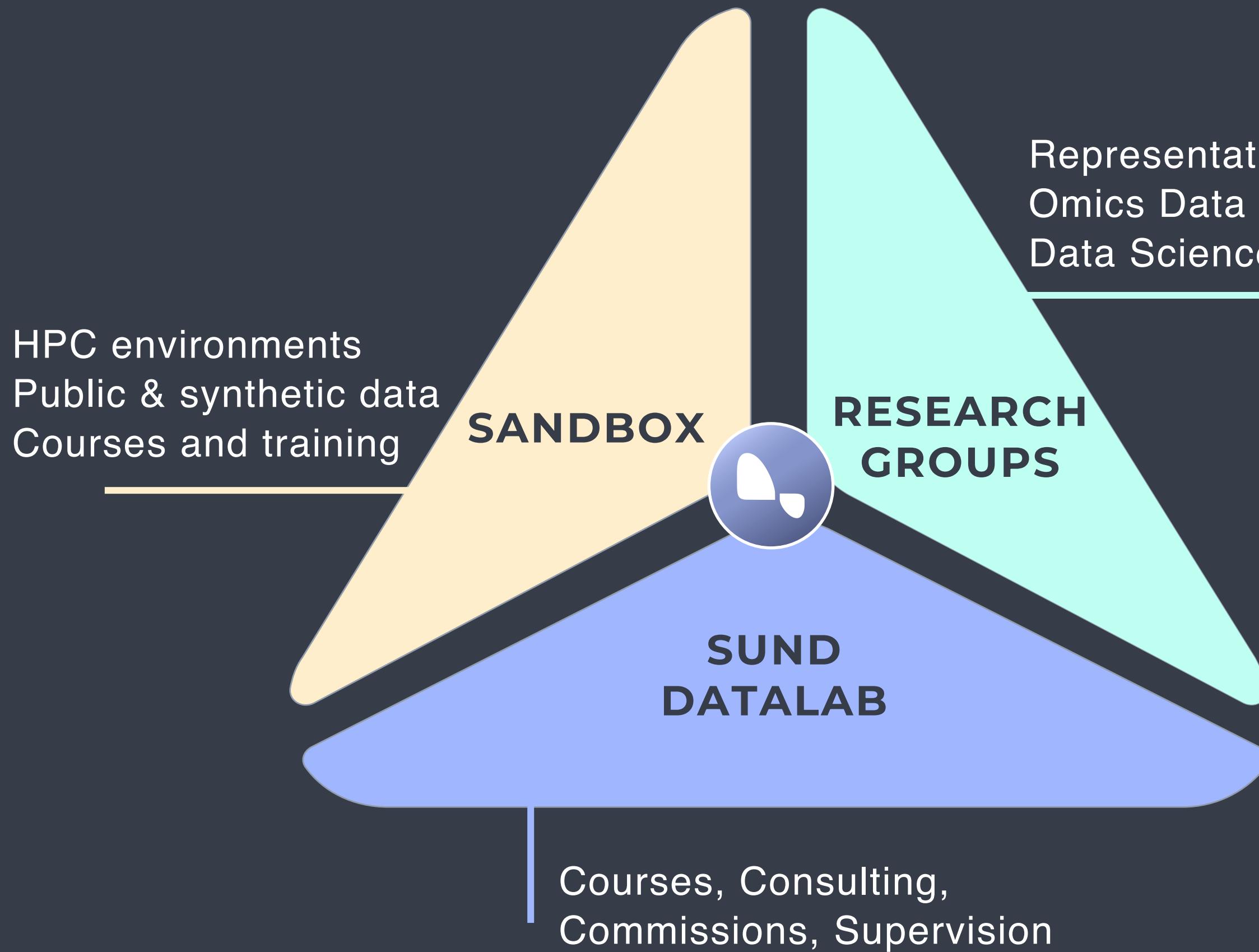


WHO ARE WE?

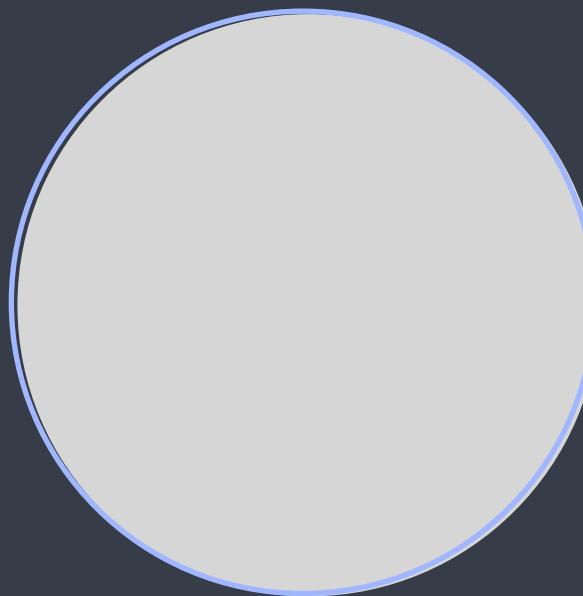


HEADS (SUND) DATA LAB

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



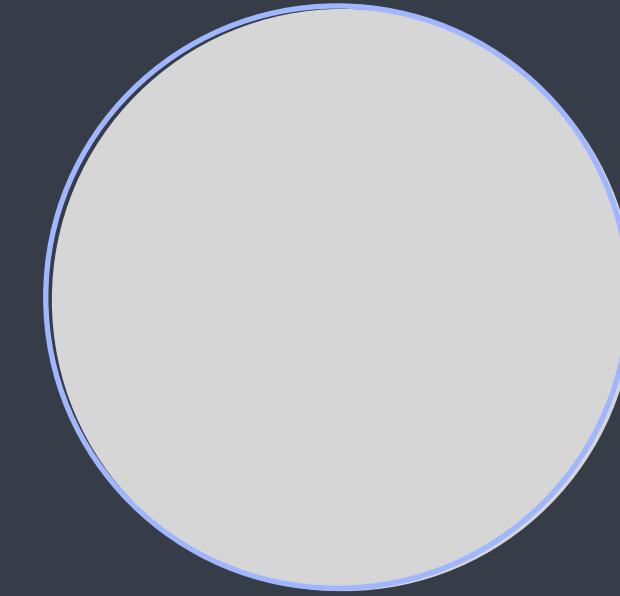
Henrike Zschach



XXX XXX



Thilde Terkelsen



David Galligani*

PROGRAM

Day 1

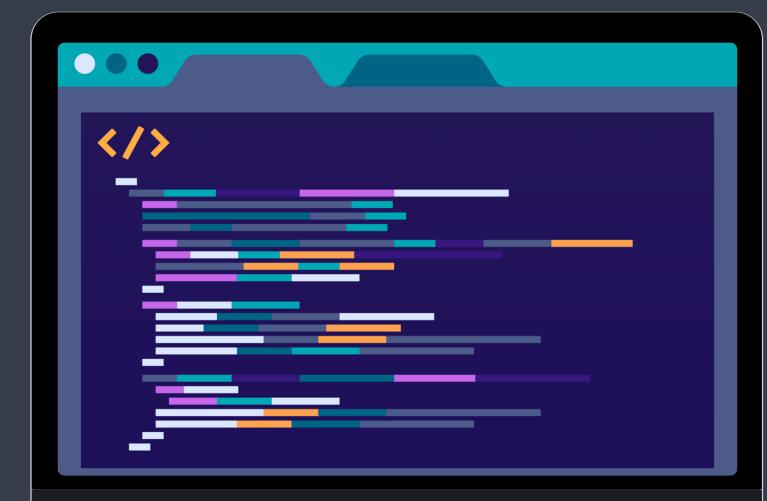
08:45	MORNING COFFEE
09:00	INTRODUCTION TO COMMANDLINE
09:15	PART 1: NAVIGATING FILES & DIRECTORIES
09:35	EXERCISE 1
10:00	PART 2: FILE OPERATIONS
10:15	EXERCISE 2
10:30	COFFEE BREAK
10:45	PART 3: PROJECT ORGANIZATION & BACKUP
11:05	EXERCISE 3
11:30	PART 4: VIEWING AND EDITING FILES
12:00	LUNCH
13:00	EXERCISE 4
13:20	PART 5: DATA WRANGLING
13:45	COFFEE BREAK
14:00	EXERCISE 5
15:00	Q + A

Day 2

08:45	MORNING COFFEE
09:00	PART 6: DATA WRANGLING 2
09:25	EXERCISE 6
10:25	COFFEE BREAK
10:40	PART 7: REDIRECTION & PIPES
11:00	EXERCISE 7
12:00	LUNCH
13:00	PART 8: SHELL SCRIPTS & LOOPS
13:30	EXERCISE 8
14:15	COFFEE BREAK
14:30	PART 9: INSTALLATIONS, CONFIG FILES, WORKFLOW LANGUAGE
15:00	EXERCISE 9
15:30	BYE BYE

COURSE MATERIALS:

<https://github.com/Center-for-Health-Data-Science/Just-Bash-It>





```
Last login: Fri Jun 24 15:05:34 on ttys000  
[kgx936@SUN1007442 ~ % echo "Just Bash It"  
Just Bash It  
kgx936@SUN1007442 ~ % ]
```

“

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

“How do these concepts connected to my computer?”

SET UP CHECK

```
Last login: Fri Jun 24 15:05:34 on ttys000  
[kgx936@SUN1007442 ~ % echo "Just Bash It"  
Just Bash It  
kgx936@SUN1007442 ~ %  
  
MINGW64:/c/Users/pnv719  
pnv719@SUN1022949 MINGW64 ~  
$ -
```

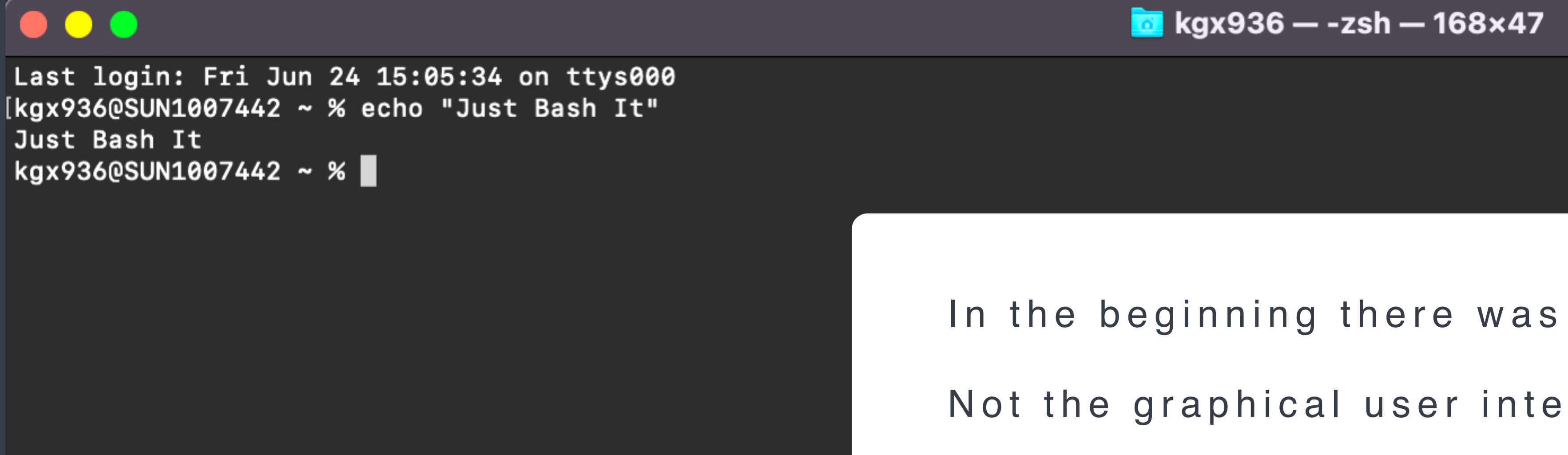
You should all have a terminal.
It looks like this...

During this course we will do
exercises inside the terminal.

We will also show commands
during the lecture.

We encourage you to open your
terminal and try them out as we
go along.

WHAT IS BASH?



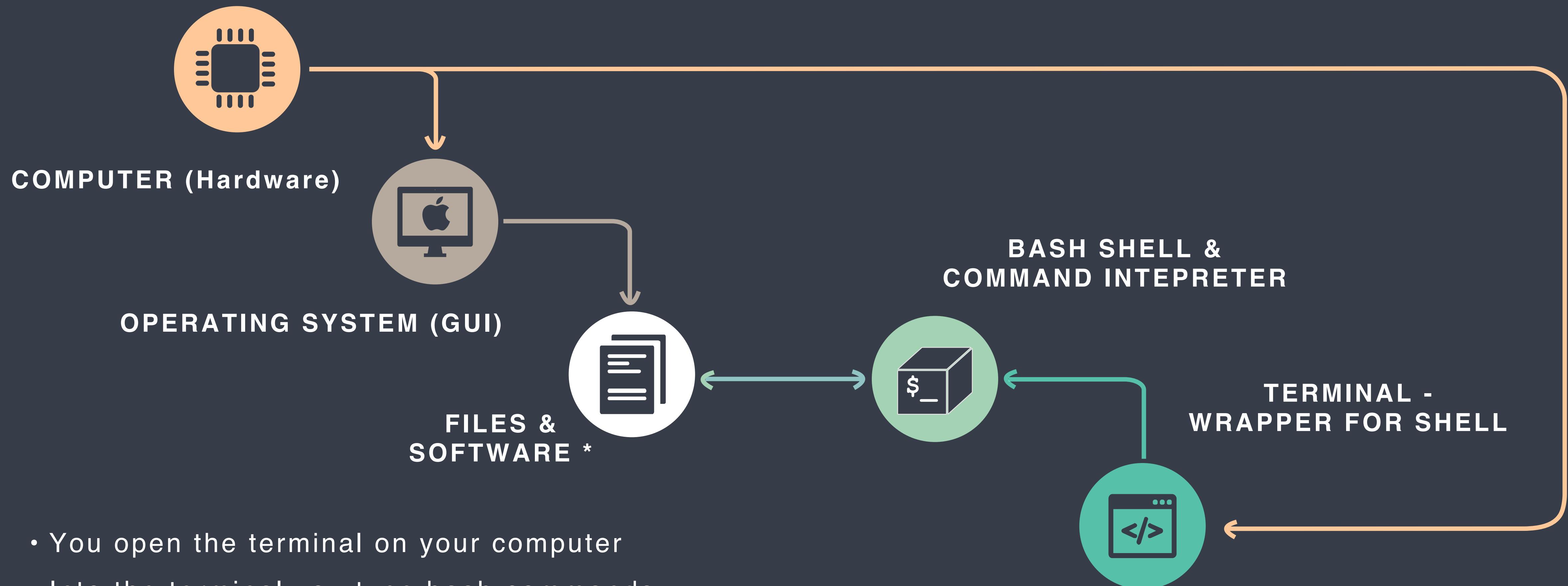
```
Last login: Fri Jun 24 15:05:34 on ttys000
[kgx936@SUN1007442 ~ % echo "Just Bash It"
Just Bash It
kgx936@SUN1007442 ~ %
```

In the beginning there was the word –
Not the graphical user interface!

The first computers had no graphical user interfaces (GUI) where you could click. Instead, you interacted with the computer via the terminal.

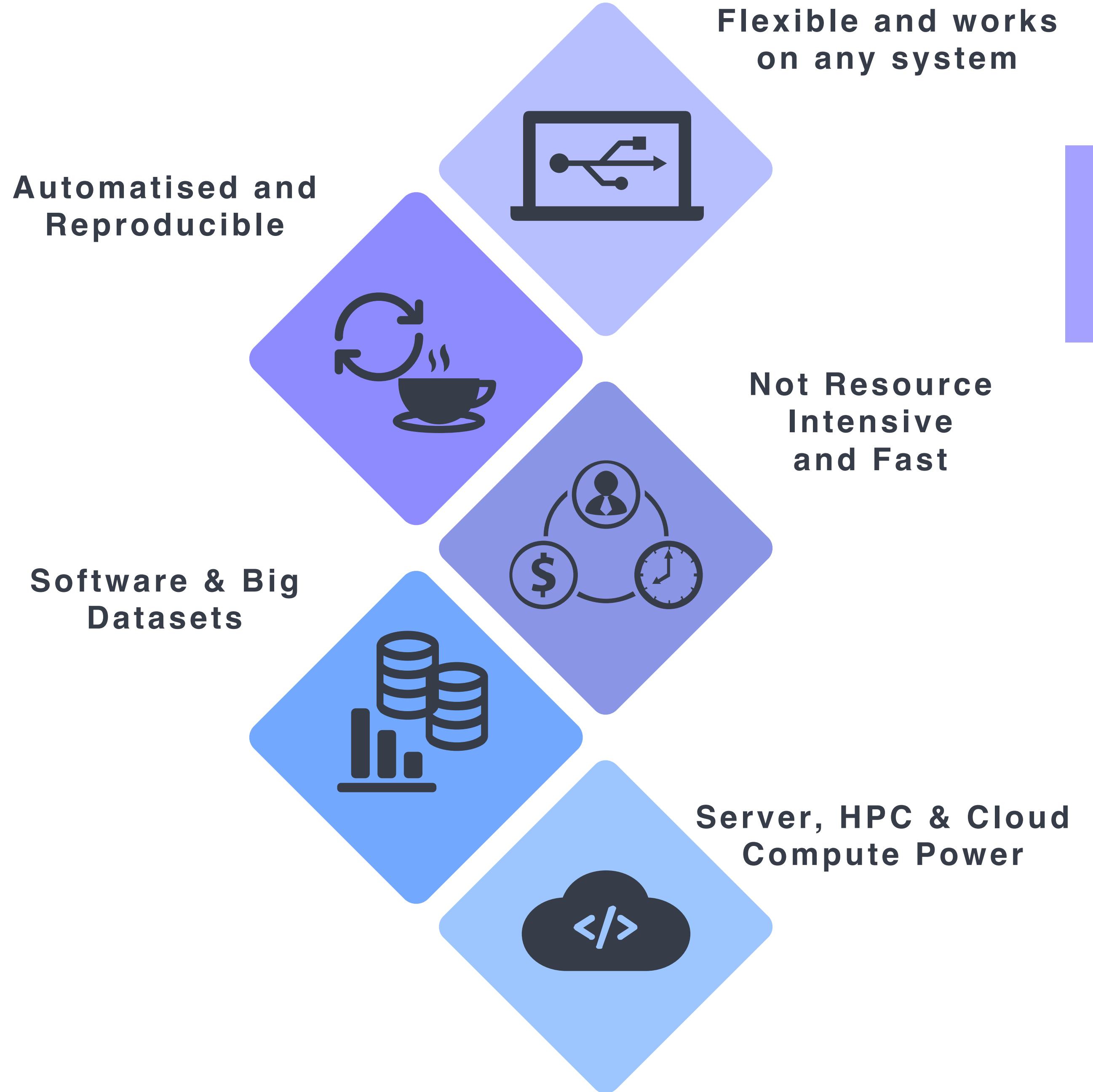
There are still computing environments that do not have a GUI.
Many scientific programs do not have a GUI either. You need to execute them via the terminal.

TERMINOLOGY



- You open the terminal on your computer
- Into the terminal you type bash commands
- Commands are interpreted by the shell

* software == executable file



SELL IT TO ME!

Benefits of bash & command line:

- Your analysis will be reproducible, automated and parallelised.
- Fast processes, less computationally intensive.
- Command line software.
- 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

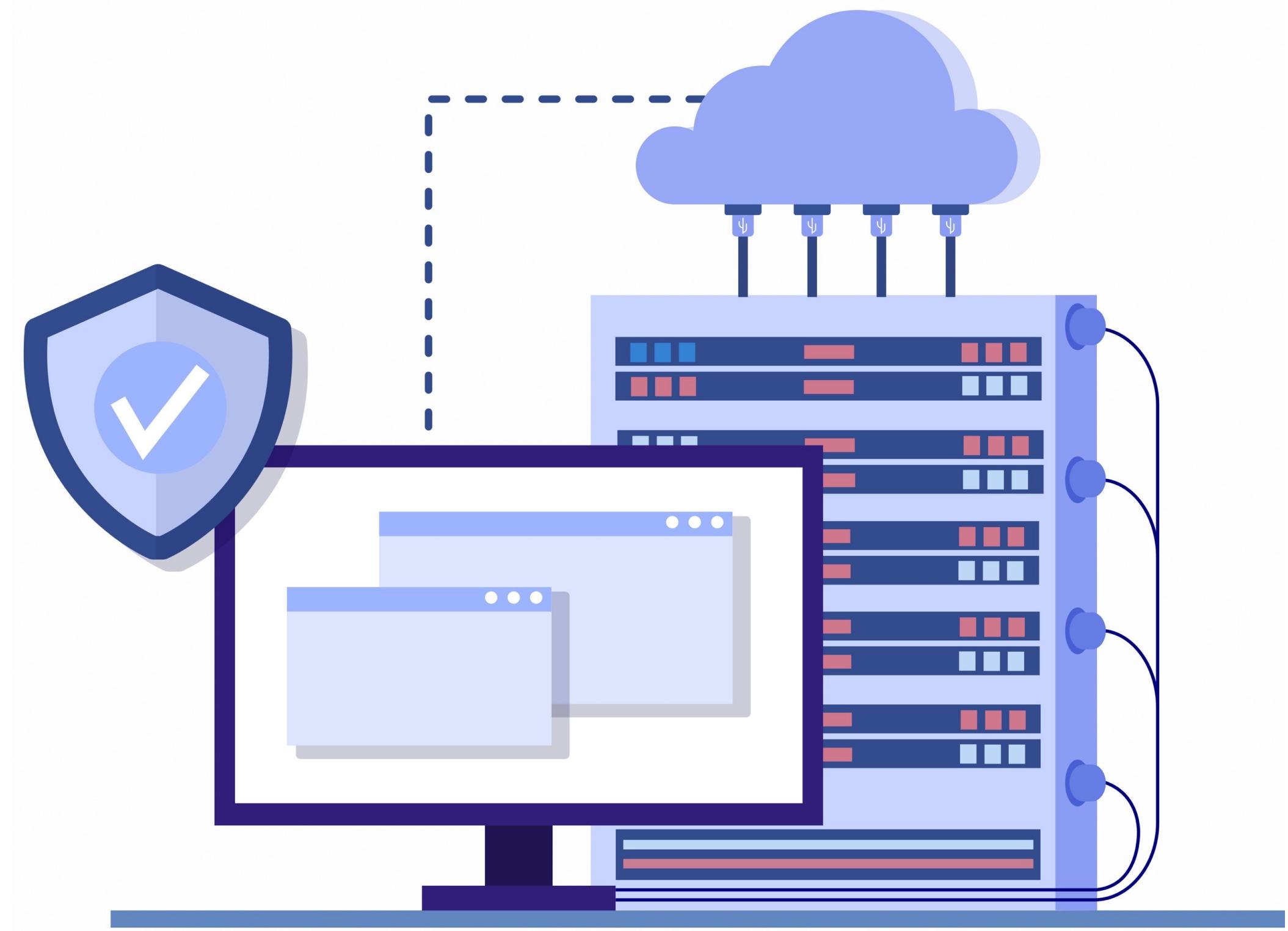
WORKING ON A REMOTE SERVER

Server or High Performance Computing (HPC)
System.

When:

- 01 Not enough computational power (CPU, GPU)
- 02 Not enough memory (RAM) or storage (HDD)
- 03 Not secure, i.e. person sensitive data
- 04 Software is complicated to install and set up

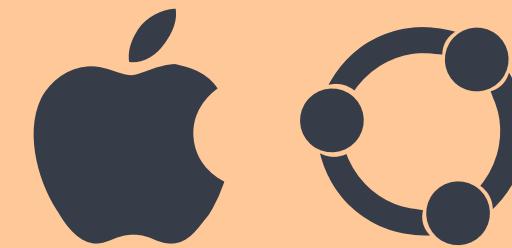
Most servers and a many HPCs do not have GUIs ,
you interact with them through the **command line**



HOW DO I GET A BASH SHELL?

YEAH!
YOU HAVE A BASH SHELL &
TERMINAL ALREADY.

Search for terminal on your
laptop and open it.

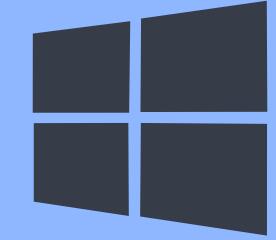


OS X or UBUNTU

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

OR INSTALL A BASH SHELL/TERMINAL:

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

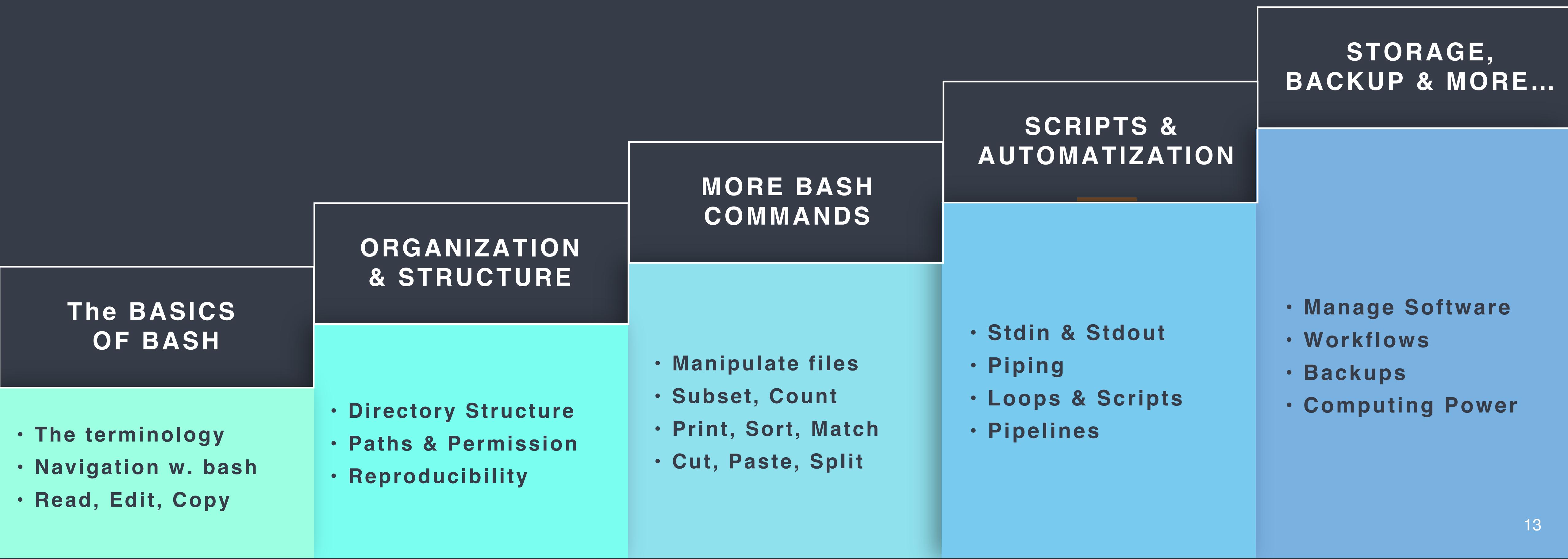


WINDOWS

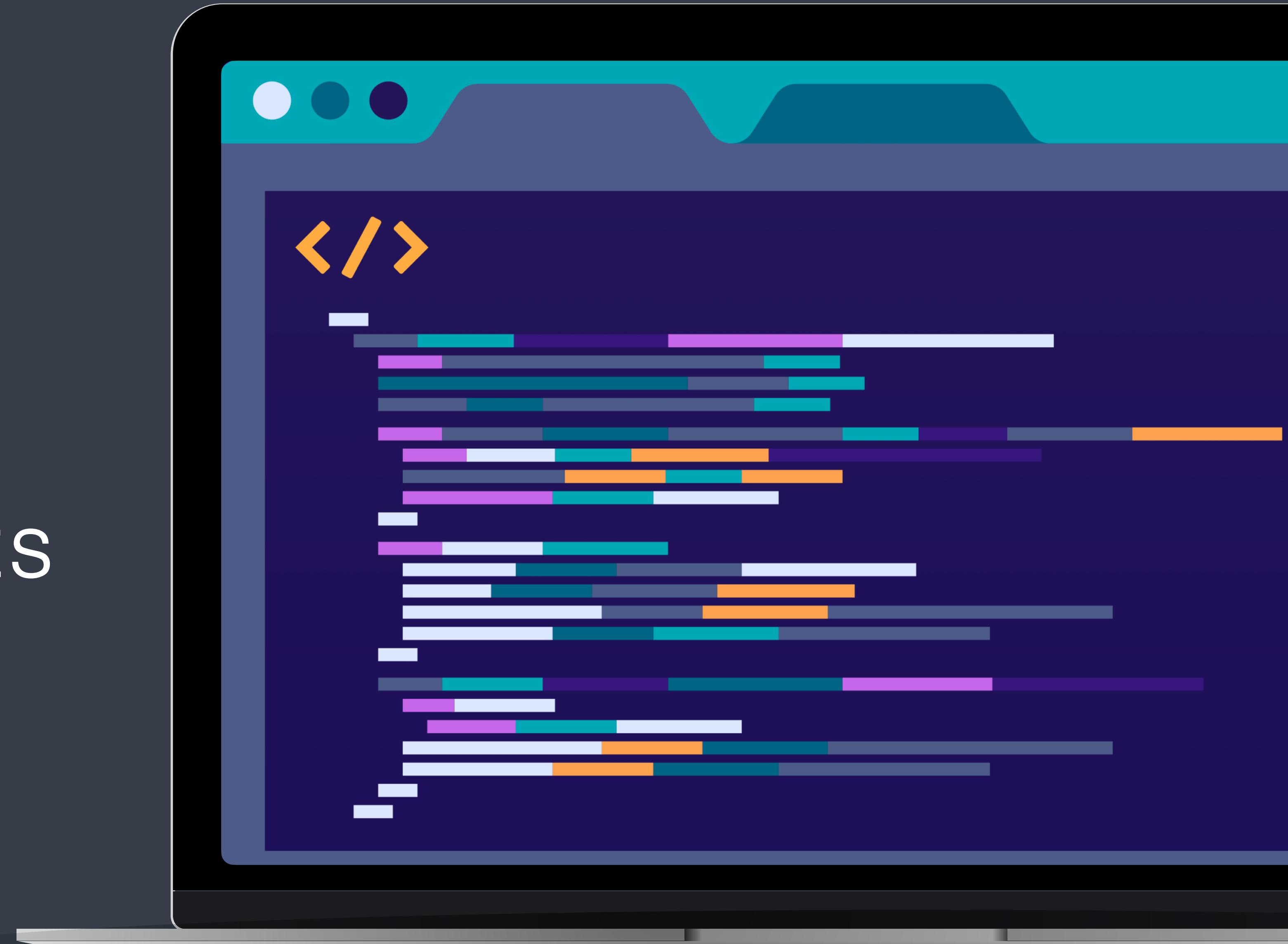
In this course Windows users will be working on **gitbash**



WHAT WILL I LEARN IN THIS COURSE?



1. NAVIGATING FILES & DIRECTORIES

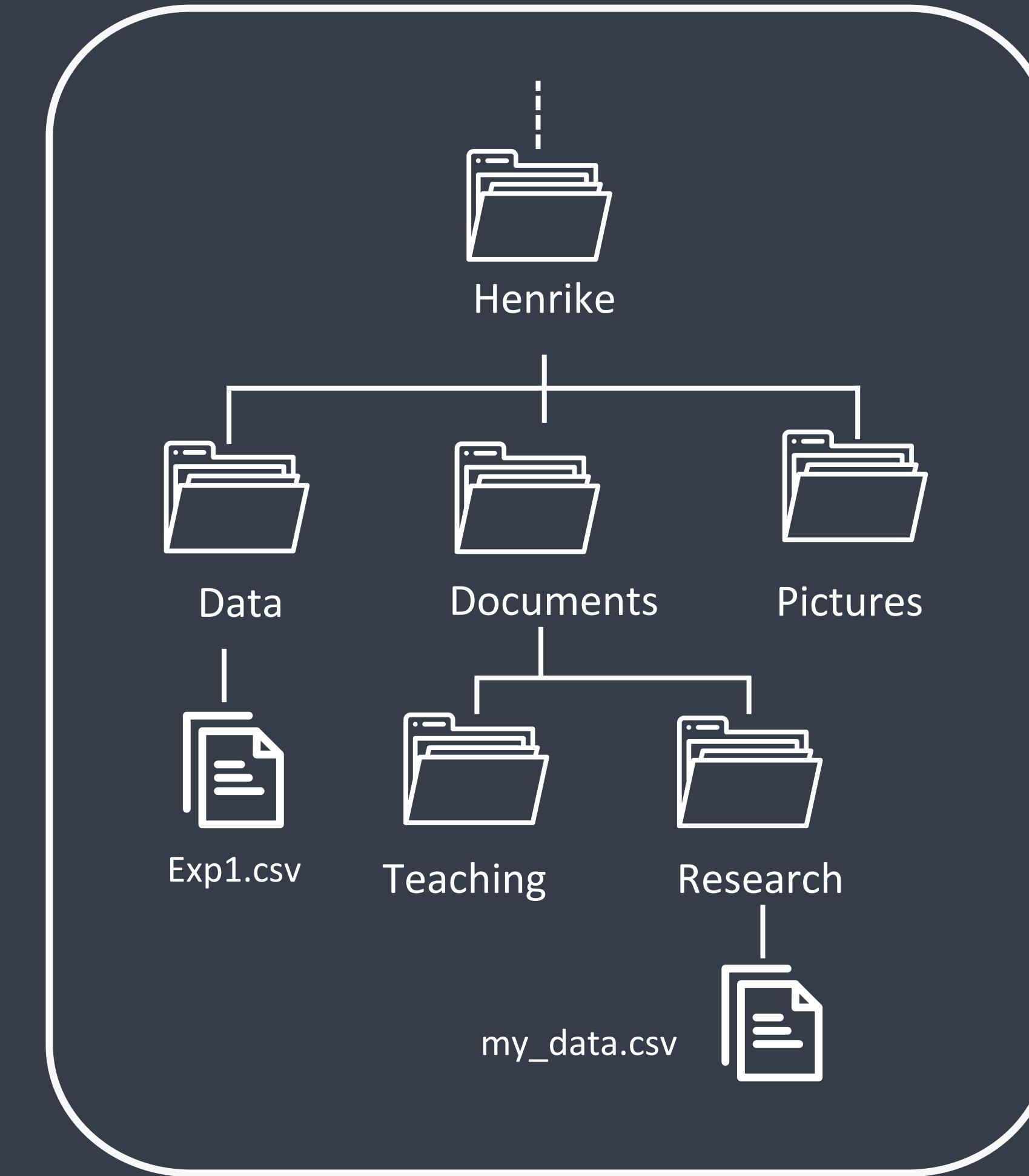
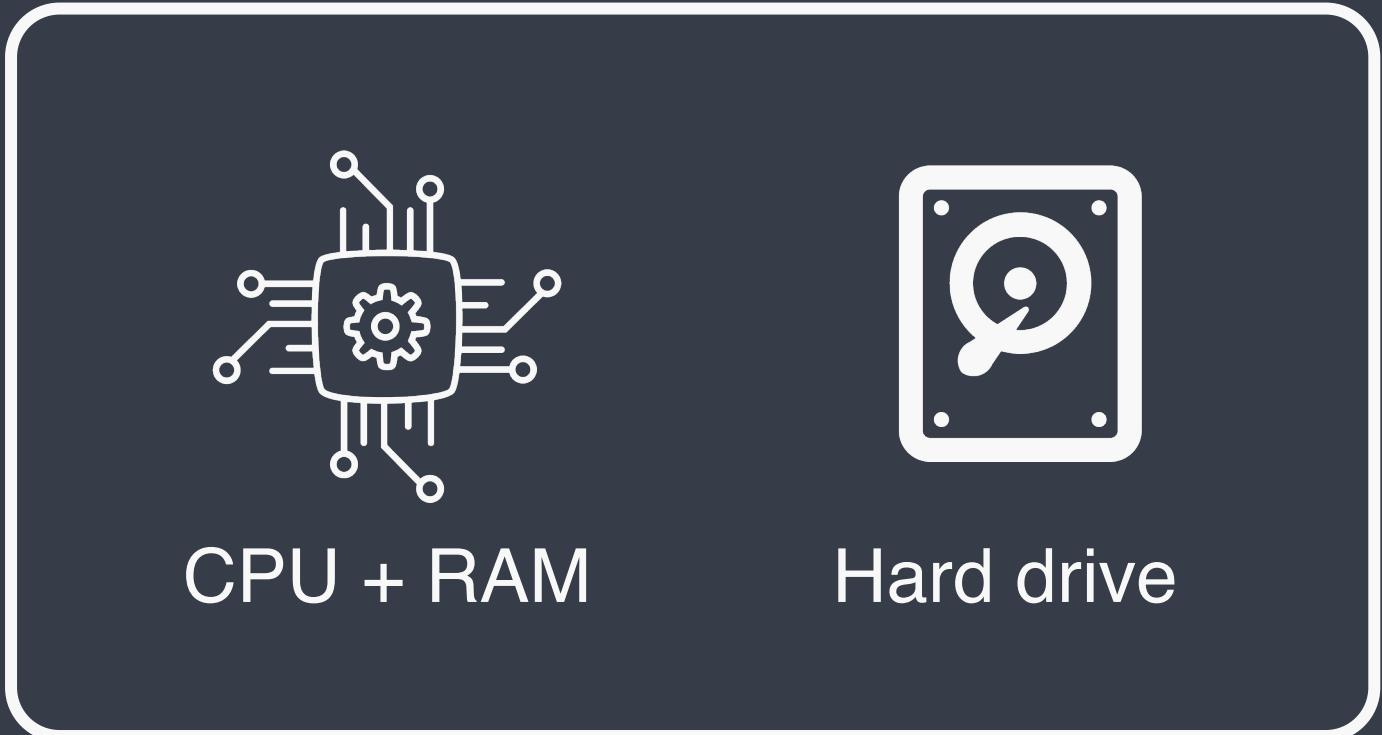


THE ANATOMY OF A COMPUTER

Software



Hardware

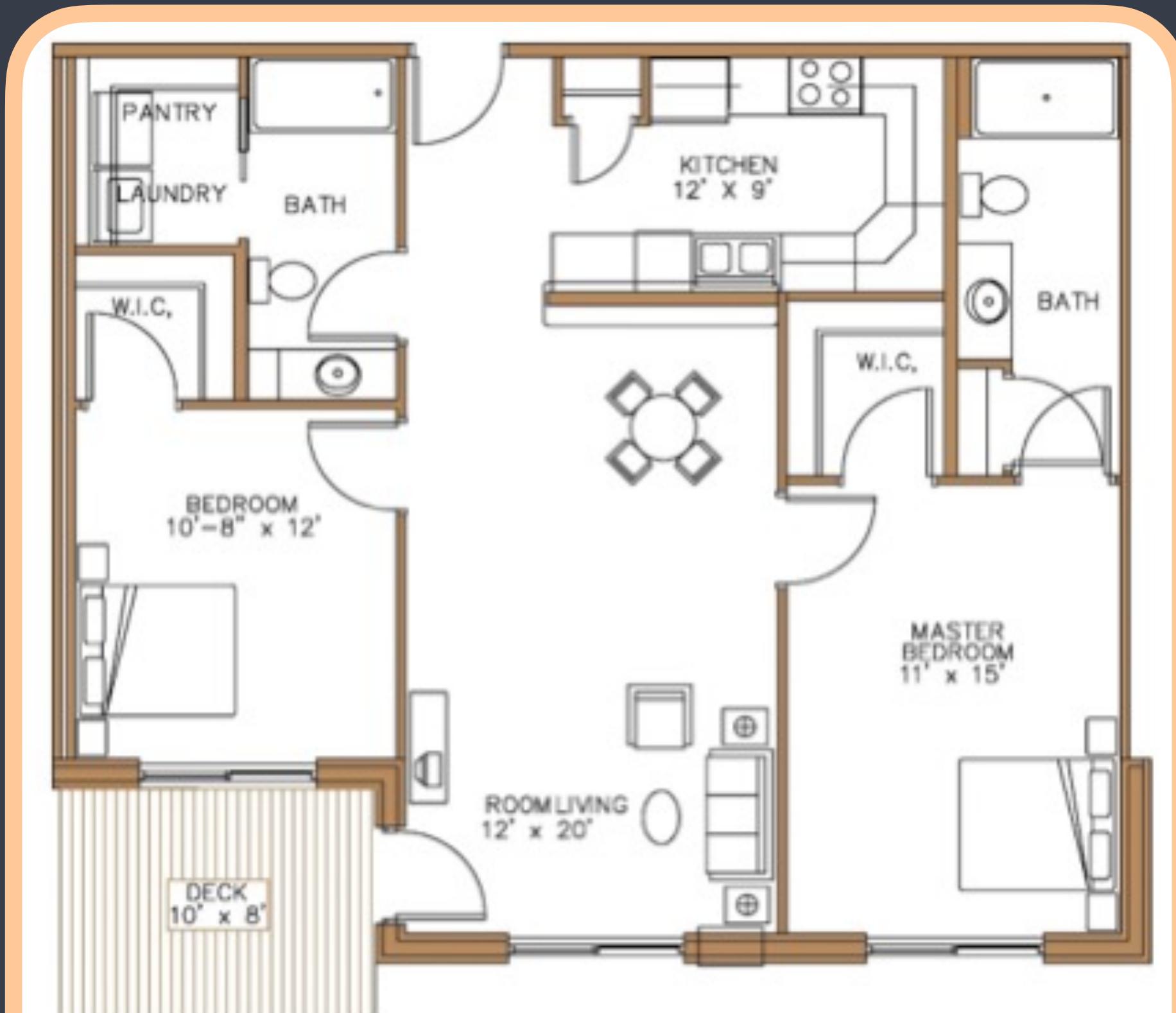


File tree

THE FILE TREE

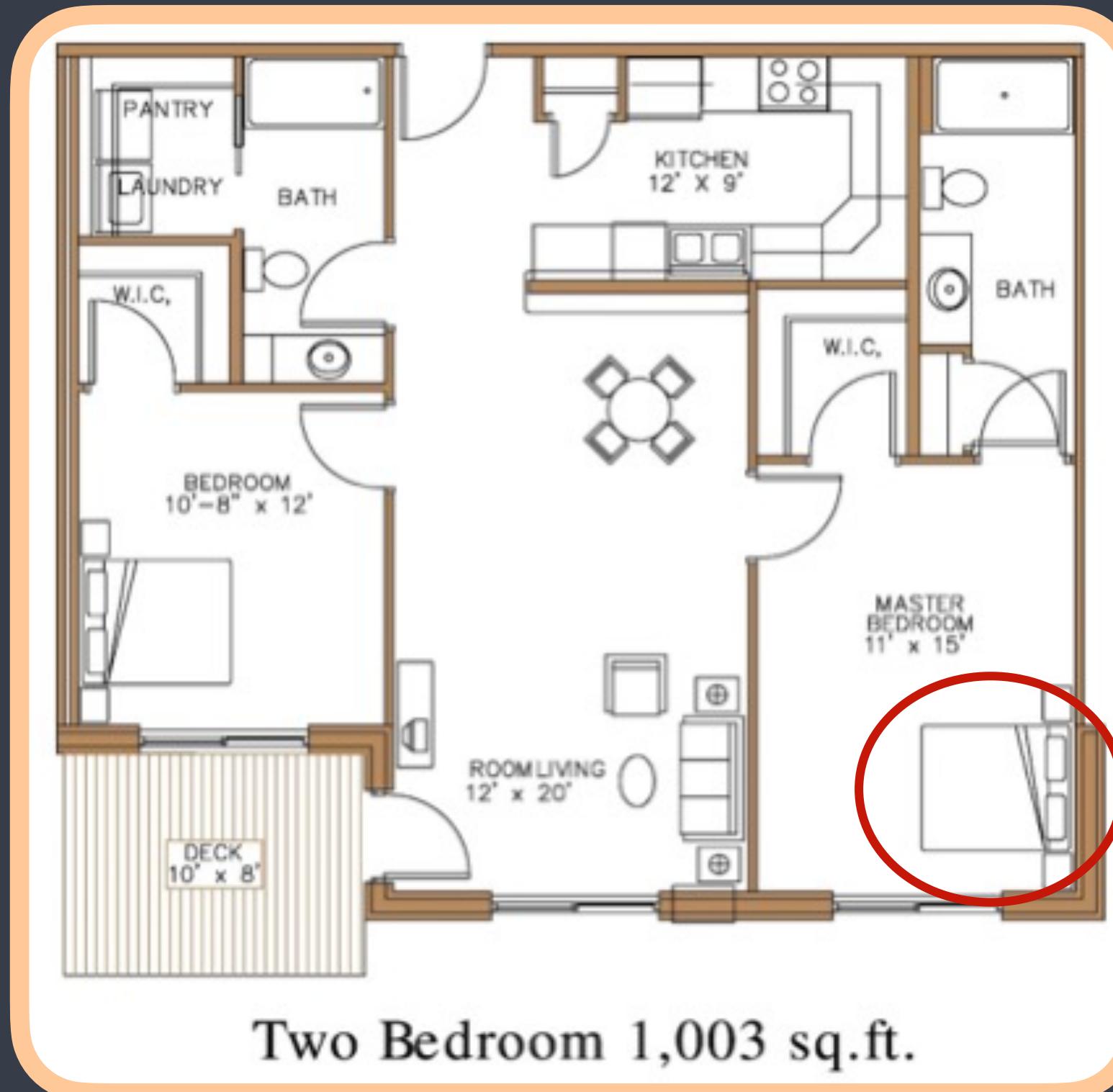
You can think of the **file tree** like the floor plan of a house:

- Rooms and items are in specific places. The bed i.e. is in the bedroom.
- You are always in a specific room. You cannot be in two different rooms at the same time.
- In order to interact with items, you need to know where in your house it is.
- Items can be moved from one room to another.



THE FILE TREE

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

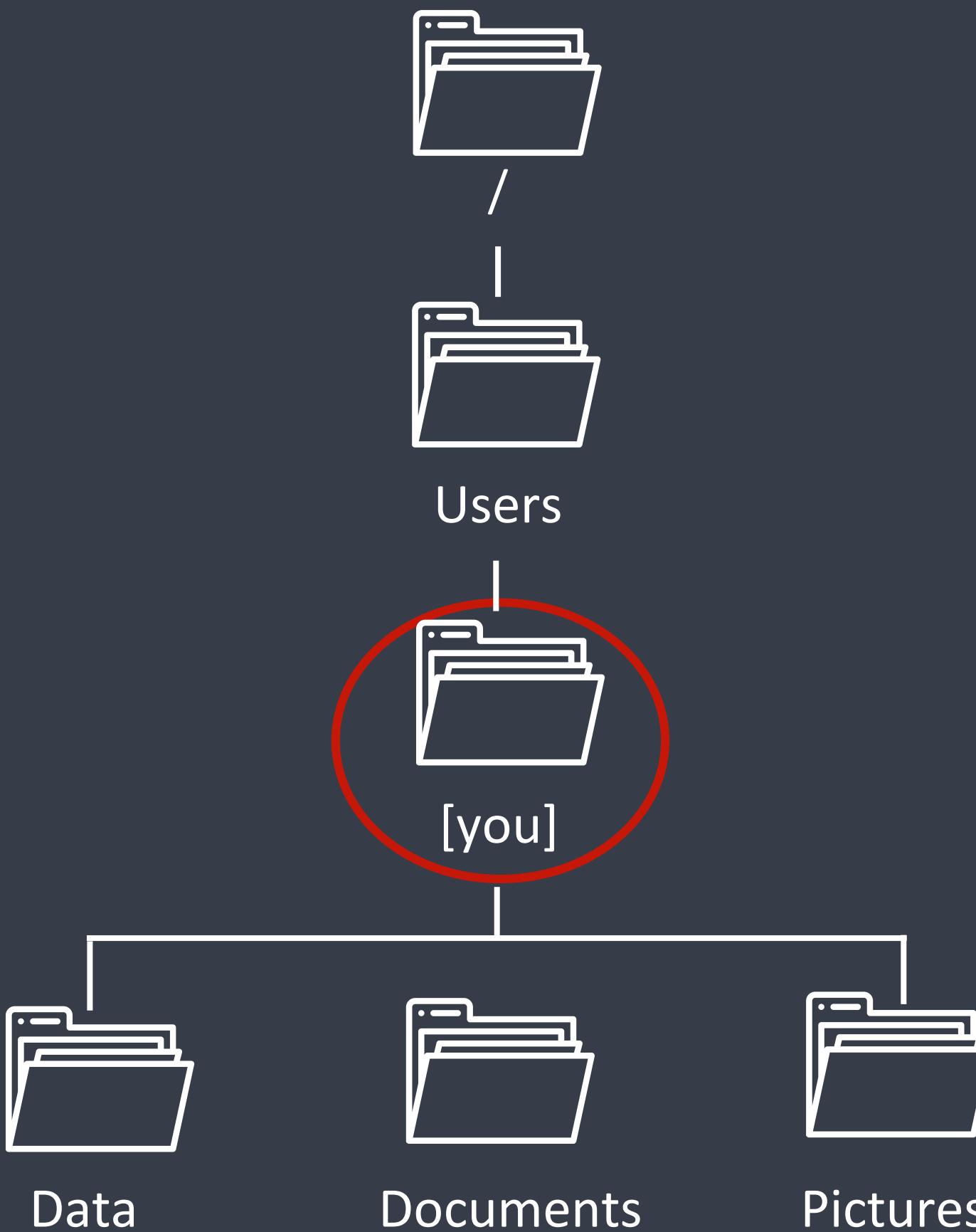


The **directory tree** is hierarchical and starts at the 'root' = '/'



THE HOME DIRECTORY

- The home directory is a special place in the file tree. It is named with your **user name** that is typically a subfolder of the ‘users’ folder.
- This is where you will be when you open your terminal
- The path to the home directory can be abbreviated with the symbol: ~
- You may notice this symbol later when looking at your working directory or changing to another directory.



THE HOME DIRECTORY

What the home directory is called can differ between operating systems and computers.
All home directories are beautiful <3!

```
kgx936 ~ %  
Last login: Tue Oct 31 13:56:40 on ttys000  
kgx936@SUN1007442 ~ % pwd  
/Users/kgx936  
kgx936@SUN1007442 ~ %
```

```
11/11/2022 14:38.08 /home/mobaxterm pwd  
/home/mobaxterm
```

```
henrike@henrike-ThinkPad-X390: ~  
henrike@henrike-ThinkPad-X390: ~ 80x24  
(base) henrike@henrike-ThinkPad-X390:~$ pwd  
/home/henrike  
(base) henrike@henrike-ThinkPad-X390:~$
```

```
MINGW64:/c/Users/pnv719  
  
pnv719@SUN1022949 MINGW64 ~  
$ pwd  
/c/Users/pnv719  
  
pnv719@SUN1022949 MINGW64 ~  
$ -
```

As a rule of thumb, the home directory is generally two steps from the root '/' and it is personal.

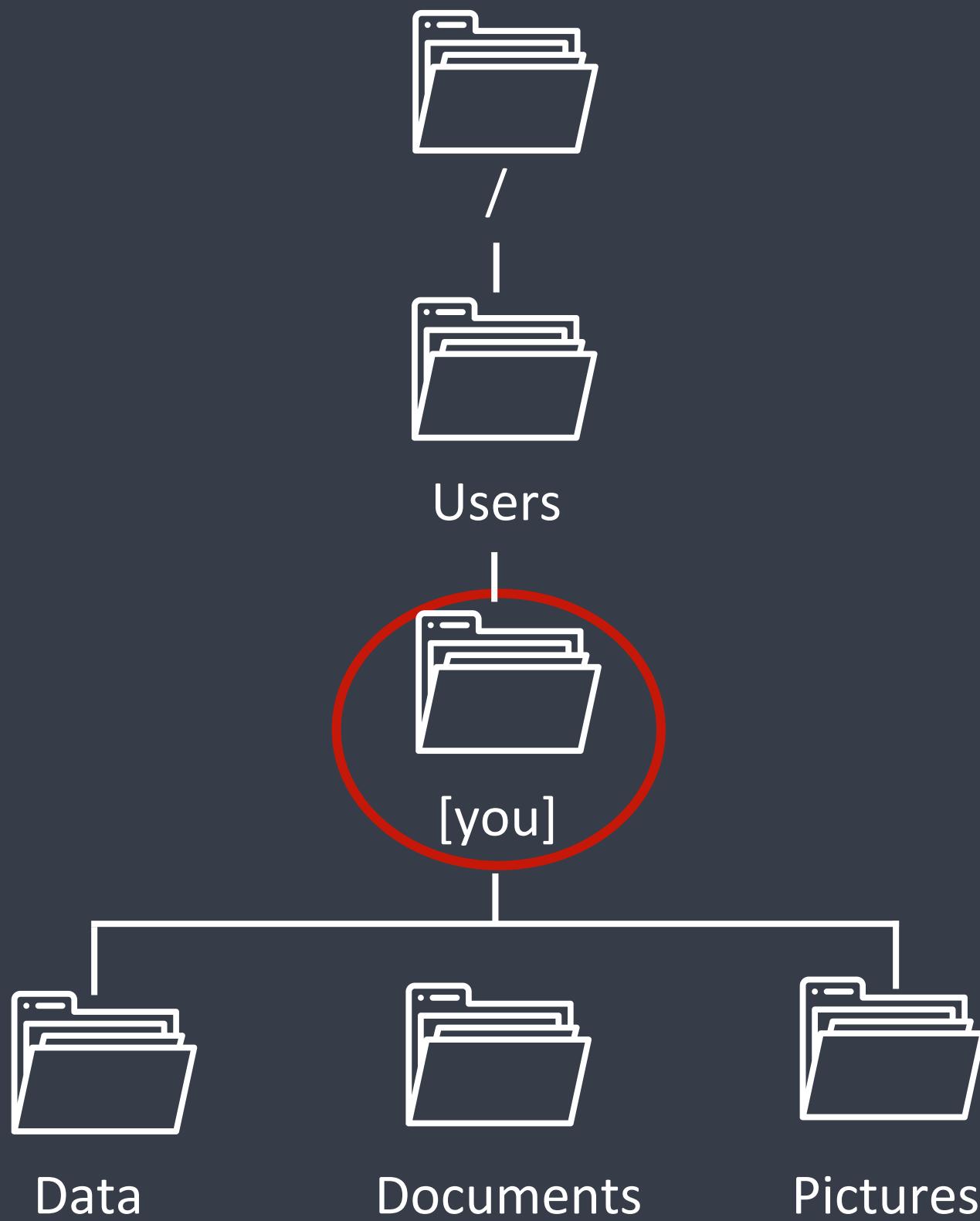
When there are several users on the same machine each has their own home directory.

CURRENT DIRECTORY

- When you operate on the terminal you are always located in a certain directory.
- The directory you are currently in is called the working directory and you can print it with:

```
$ pwd
```

The scheme shown here and on the following slides corresponds to what you have in the ‘Files’ folder of the git hub repo.



Sometimes not all folders are shown on the scheme.

LISTING CONTENT

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

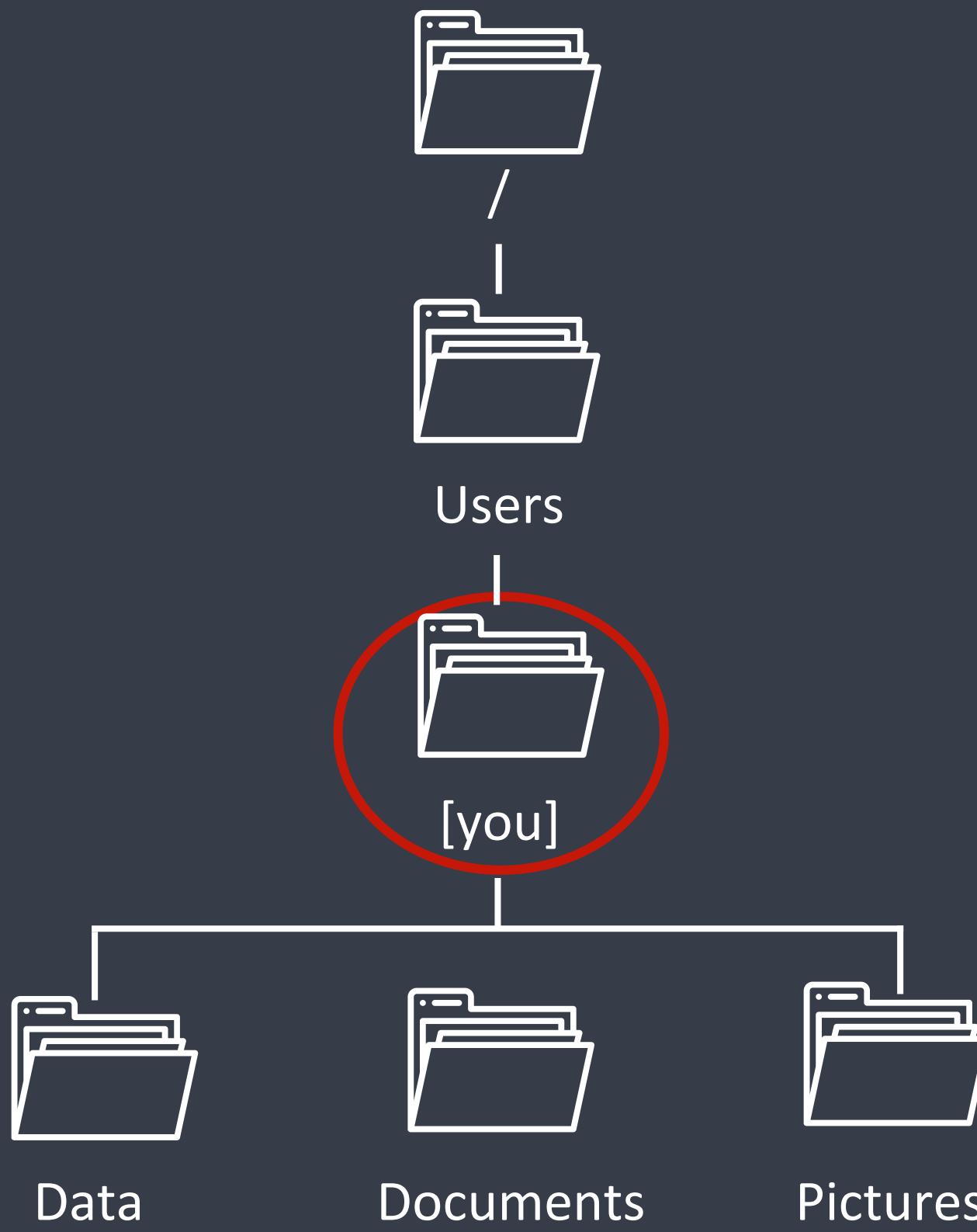
```
$ ls
```

- In the example on the right this would give you the Downloads, Documents and Pictures folders.
- You can list directories you are NOT currently inside of by specifying the **path** to them:

```
$ ls /Users
```

```
$ ls ~/Downloads
```

- ls has many useful options (**-l**, **-t** and **-h**, **-F**). We'll get to that later.



CHANGING DIRECTORIES

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

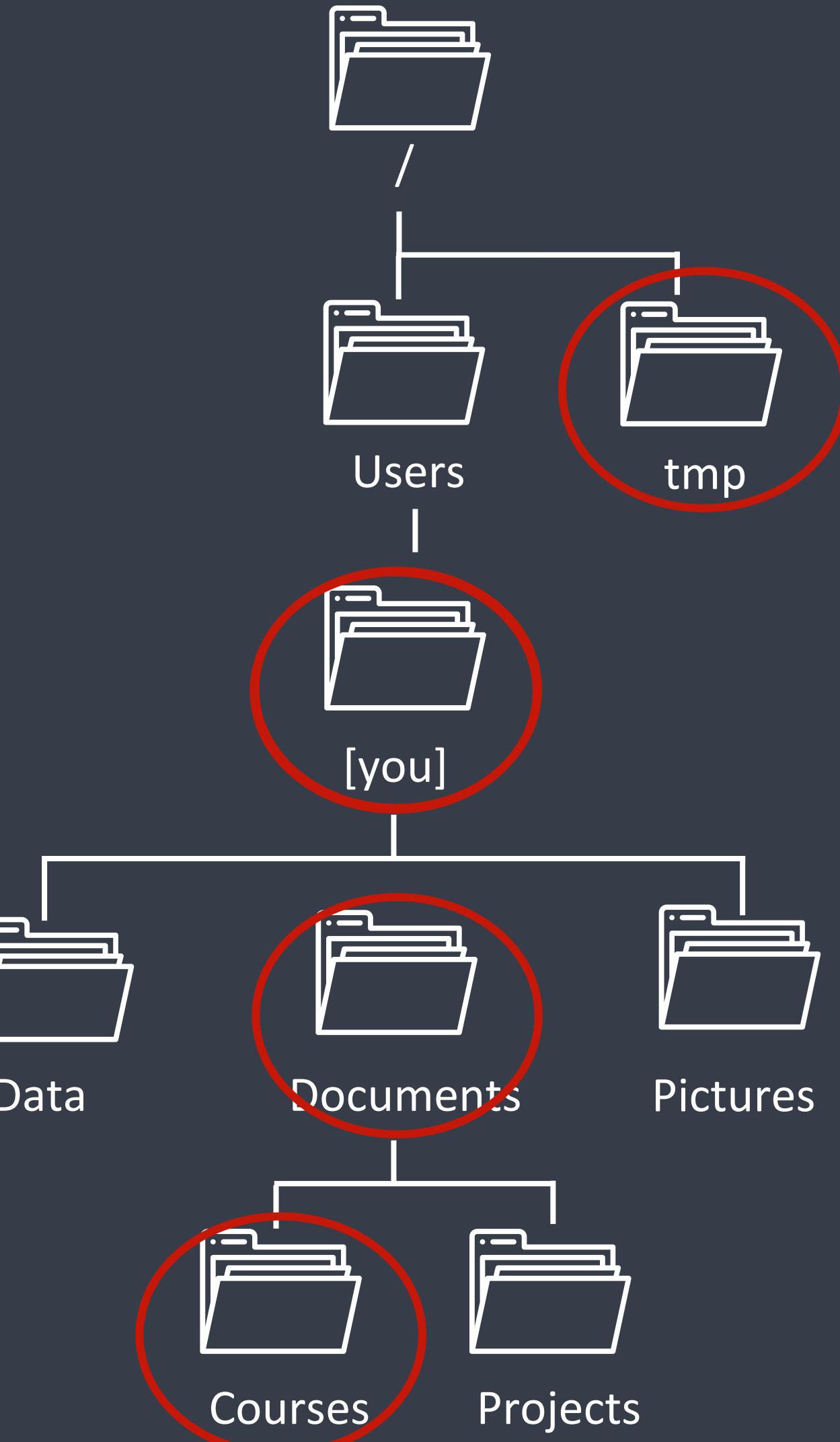
```
$ cd Documents
```

- You can also move several directories down:

```
$ cd Documents/courses
```

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

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



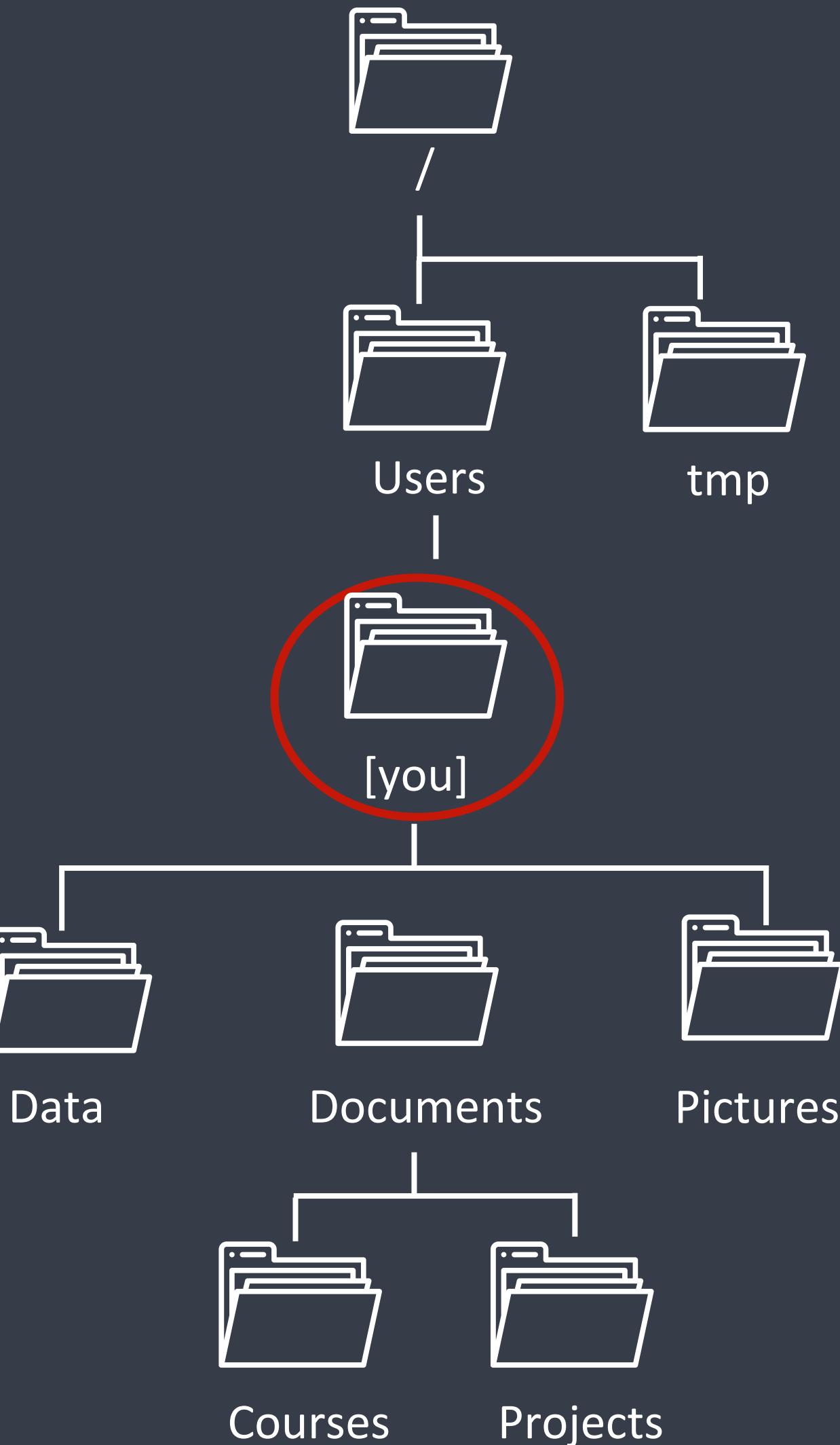
CHANGING DIRECTORIES

- cd without any arguments will always bring you to your home directory

```
$ cd
```

- cd with a dash will bring you to the last directory you were in:

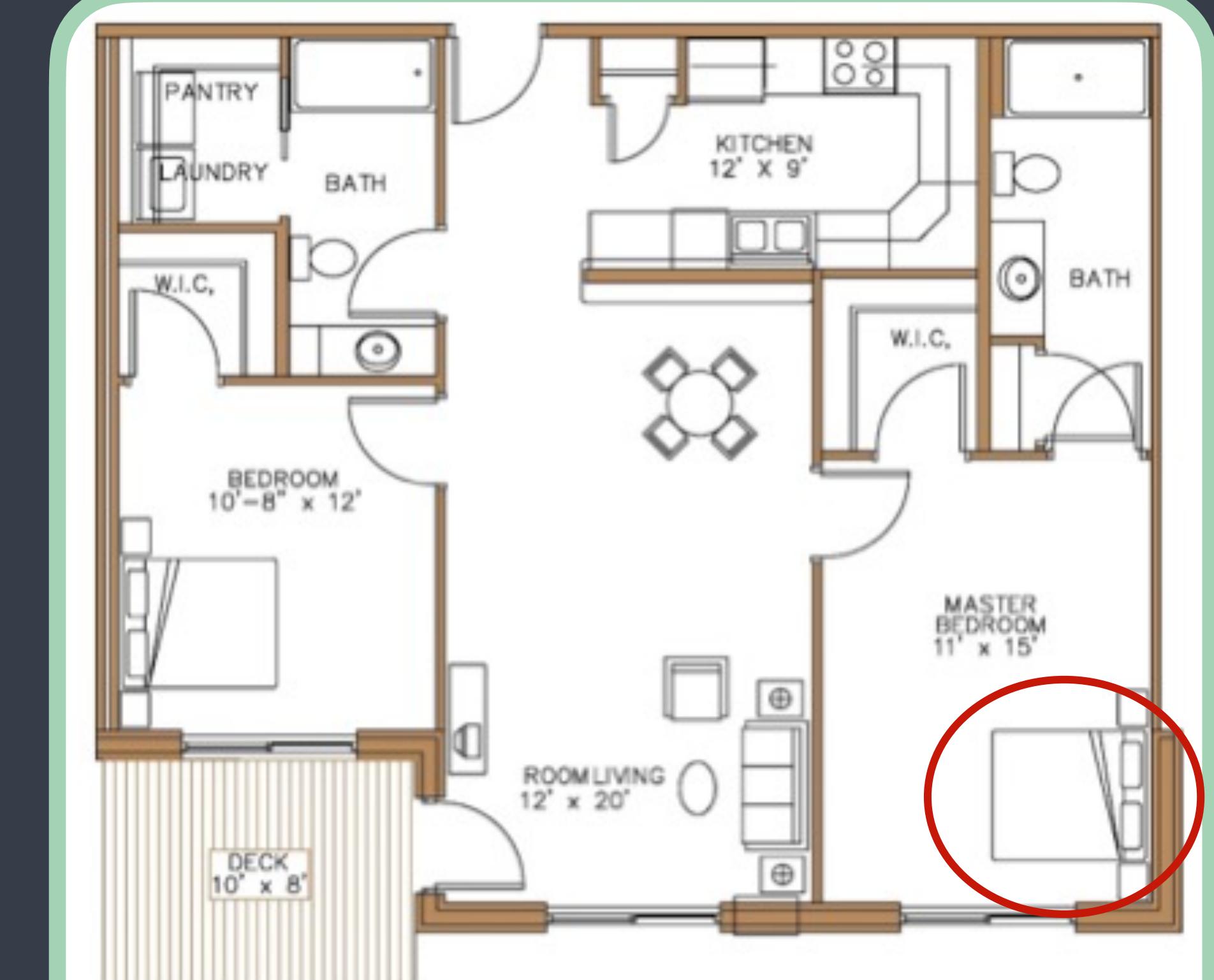
```
$ cd -
```



PATHS

- Each file and directory exist in one specific place on your computer. One could say they have an address.
- This 'address' is called a path.
- Following the example from earlier, the path to your bed would be:

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



Two Bedroom 1,003 sq.ft.

PATHS

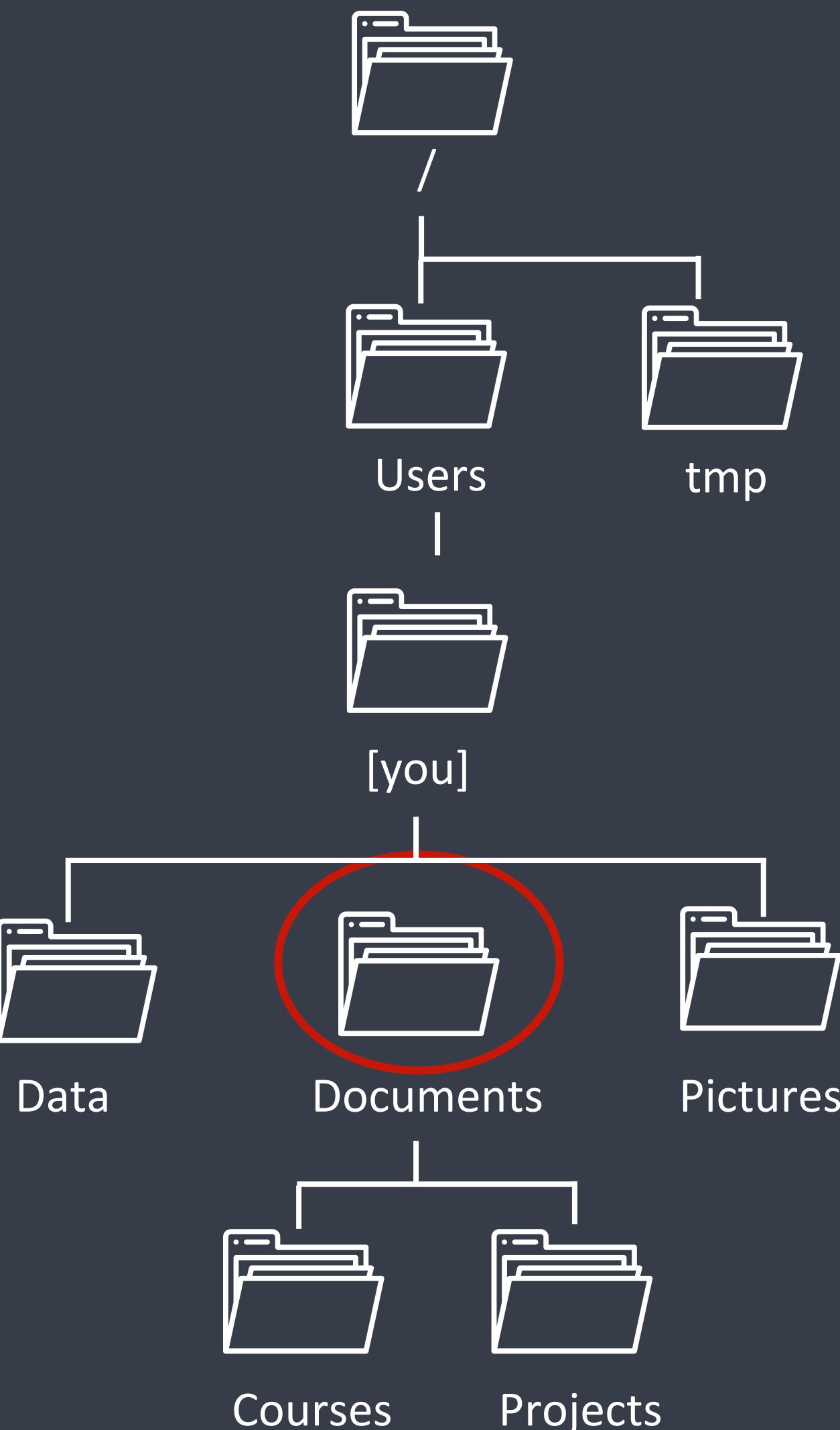
- Paths can be full/absolute or relative.
- Full paths start at the root. Since ~ represents the path to your home directory it includes the root, so:

/Users/[you]/Documents

and

~/Documents

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



PATHS

- A relative path is an address *relative* to the current working directory.
- If am working in ‘Documents’, I can address the file ‘Commandline.pptx’ as:

```
courses/Just-Bash-It/Commandline.pptx
```

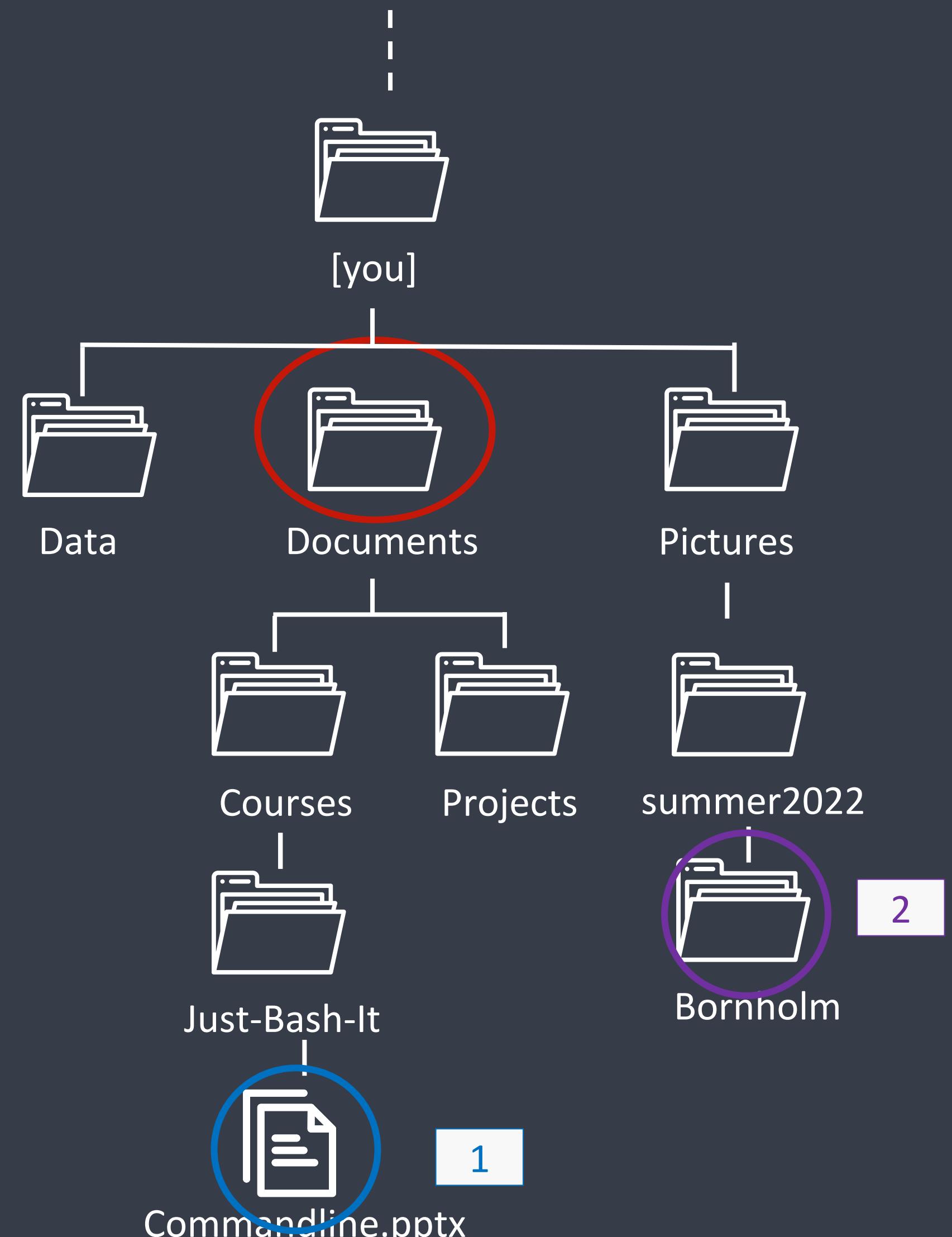
1

- Relative paths can include going up the directory tree:

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

2

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



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:

```
$ ls -lh ~ /Documents/courses
```

The command:
List contents

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

The argument:
Path to the directory
to work on

Whenever we put a space it means we are proceeding to new part of the command.
This is why file and directory names cannot contain spaces, they will be misinterpreted!

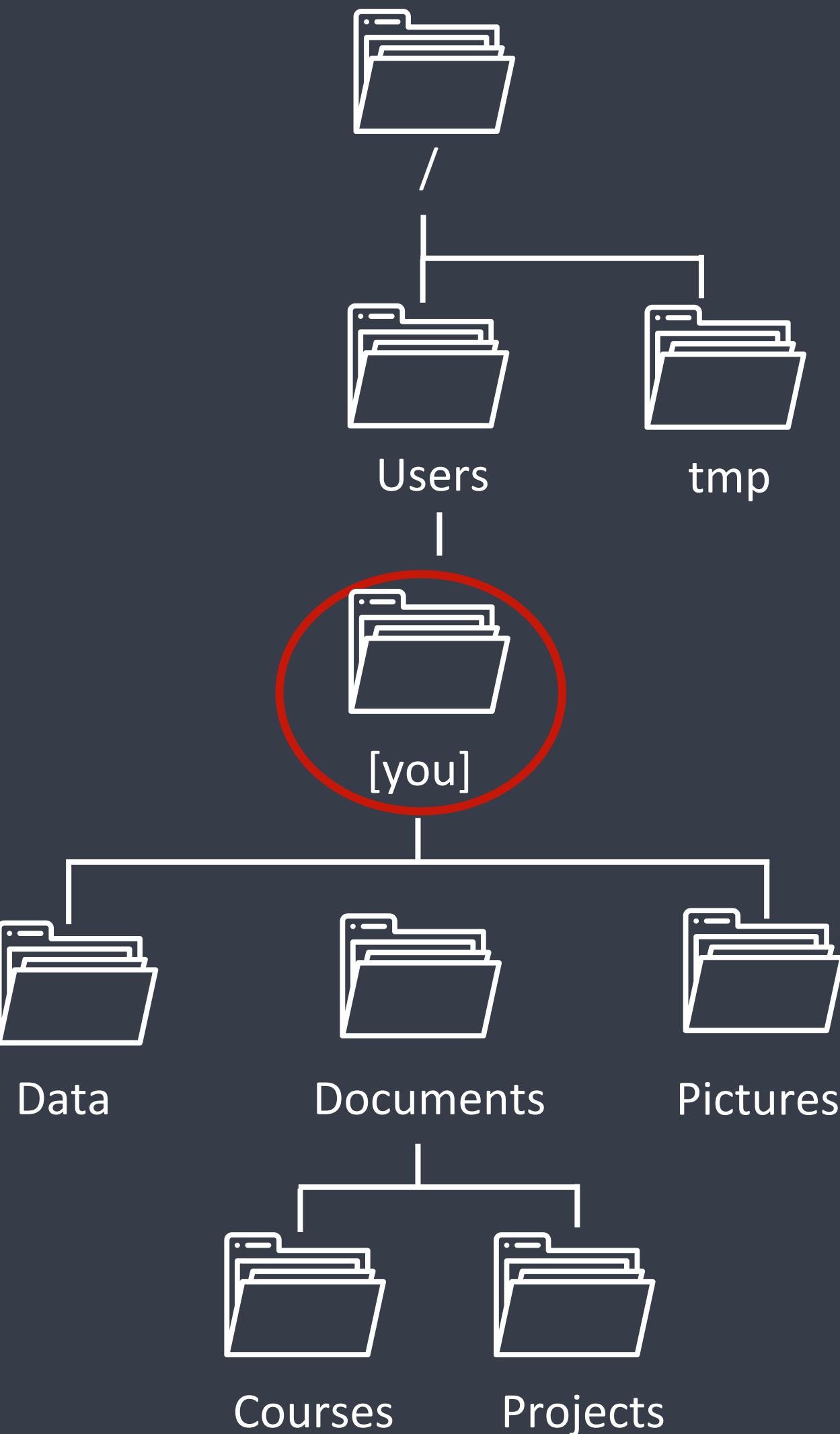
WHEN THINGS GO WRONG

Assuming I am in my home directory and I write:

```
$ cd courses
```

What will happen?

```
MINGW64:/c/Users/pnv719/Documents/Just-Bash-It/Files
pnv719@SUN1022949 MINGW64 ~
$ cd Documents/Just-Bash-It/Files/
pnv719@SUN1022949 MINGW64 ~/Documents/Just-Bash-It/Files (june_2023)
$ ls
Data/  Documents/  Pictures/  Scripts/  readme.txt
pnv719@SUN1022949 MINGW64 ~/Documents/Just-Bash-It/Files (june_2023)
$ cd Courses
bash: cd: Courses: No such file or directory
pnv719@SUN1022949 MINGW64 ~/Documents/Just-Bash-It/Files (june_2023)
$ _
```



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

```
MINGW64:/c/Users/pnv719/Documents/Just-Bash-It/Files
pnv719@SUN1022949 MINGW64 ~
$ cd Documents/Just-Bash-It/Files/
pnv719@SUN1022949 MINGW64 ~/Documents/Just-Bash-It/Files (june_2023)
$ ls
Data/  Documents/  Pictures/  Scripts/  readme.txt

pnv719@SUN1022949 MINGW64 ~/Documents/Just-Bash-It/Files (june_2023)
$ cd Courses
bash: cd: Courses: No such file or directory

pnv719@SUN1022949 MINGW64 ~/Documents/Just-Bash-It/Files (june_2023)
$ -
```

GETTING HELP

- Many commands have a help page, i.e. a **manual**:

```
$ 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)

- **man** is not included in gitbash but you can try **--help** or **-h** instead.

- You can always search the web if all else fails!

```
henrike@henrike-Swift-SF314-42: ~
henrike@henrike-Swift-SF314-42: ~ 56x24
LS(1) User Commands LS(1)

NAME
    ls - list directory contents

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

DESCRIPTION
    List information about the FILEs (the current
    directory by default). Sort entries alphabeti-
    cally if none of -cftuvSUX nor --sort is speci-
    fied.

    Mandatory arguments to long options are manda-
    tory for short options too.

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

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

ge ls(1) line 1/297 7% (press h for help or q to quit)

CHEAT SHEET 1

```
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
```

WHERE & WHAT



```
Last login: Fri Jun 24 15:05:34 on ttys000  
[kgx936@SUN1007442 ~ % echo "Just Bash It"  
Just Bash It  
kgx936@SUN1007442 ~ % ]
```

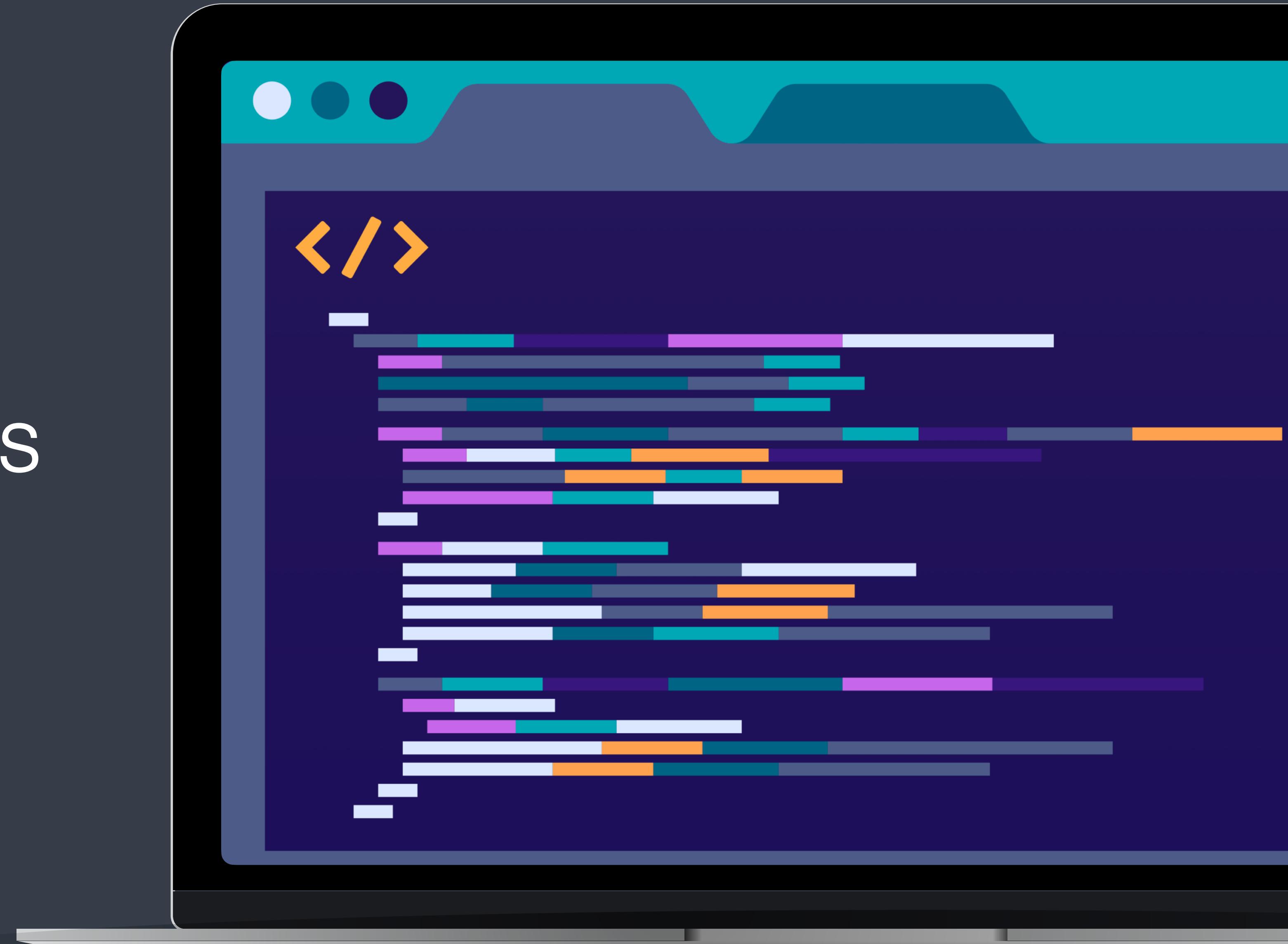
“

Let's Bash It!

Open your terminal on your computer and do **Exercise 1**.

We're here to help if you get stuck!

2. FILE OPERATIONS



MOVING FILES AND DIRS

Let's go into the '*Documents*' folder under the 'Files' of the github repo.

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

```
$ mv pancakes.txt ../recipes
```

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

```
$ mv pancakes.txt egg_cakes.txt
```



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



Documents



bills



recipes



pancakes.txt



egg_cakes.txt

COPY

- Files can be duplicated with **cp**:
- The first argument is the file you want to copy, the second argument is the destination

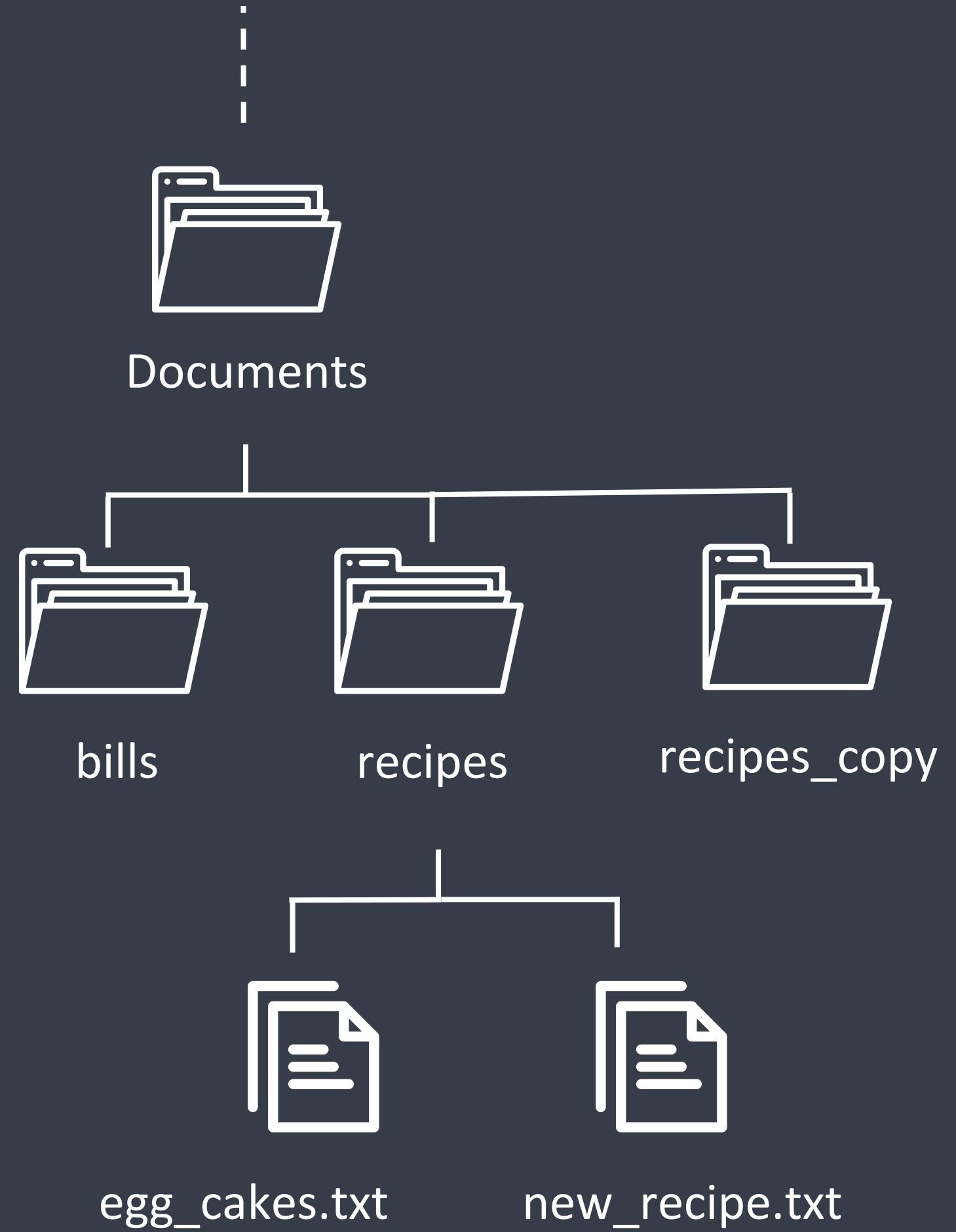
```
$ cp egg_cakes.txt new_recipe.txt
```

- To copy a directory (and its contents!) use **-r**:

```
$ cp -r recipes recipes_copy
```

- You must specify a destination when you copy.
- The destination can be the name of the new file, or another directory to copy the file to:

```
$ cp egg_cakes.txt ../recipes_copy/
```



REMOVE

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

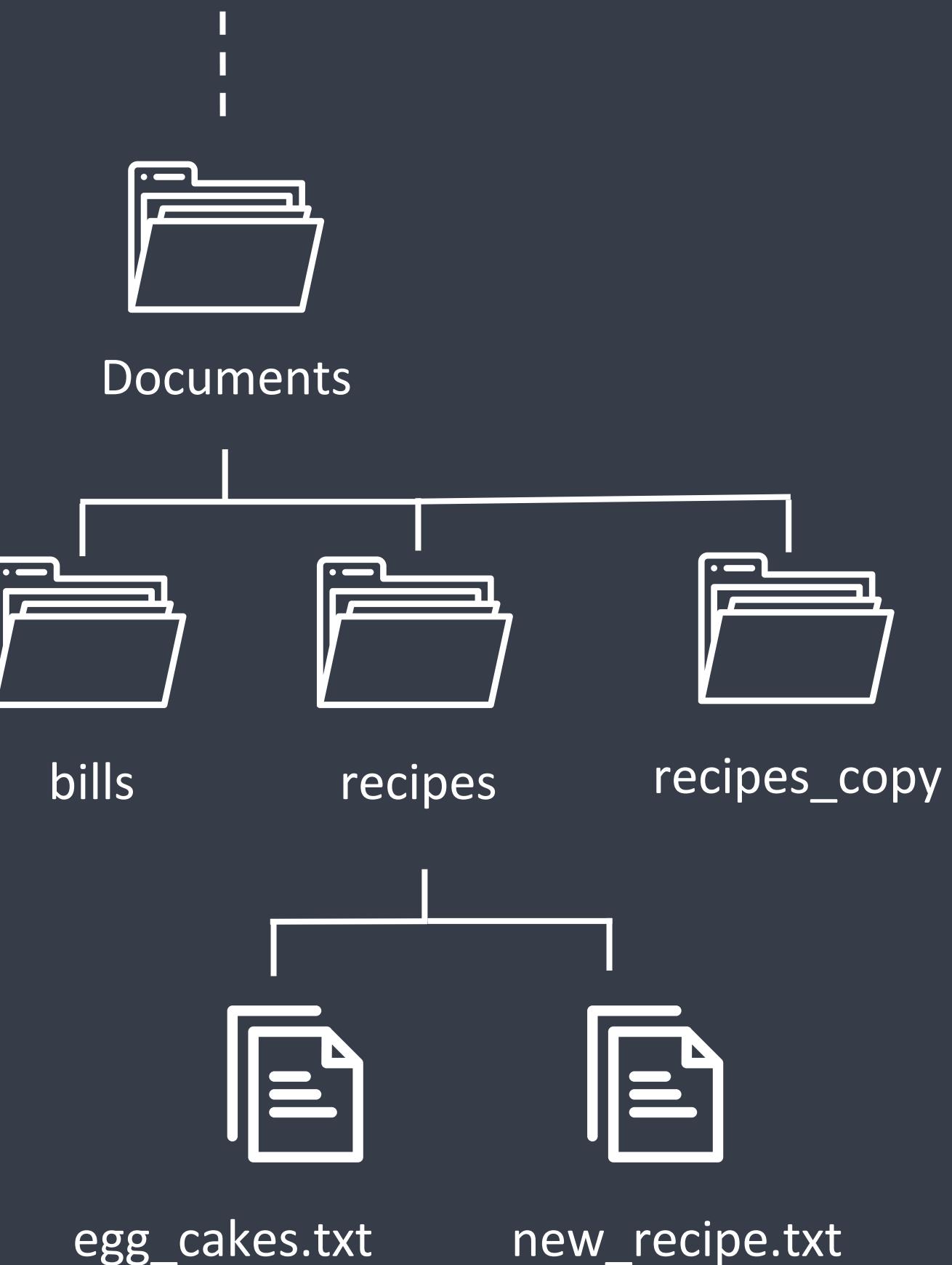
```
$ rm new_recipe.txt
```

- Just like when copying, you need the **-r** flag when removing a directory:

```
$ rm -r recipes
```



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



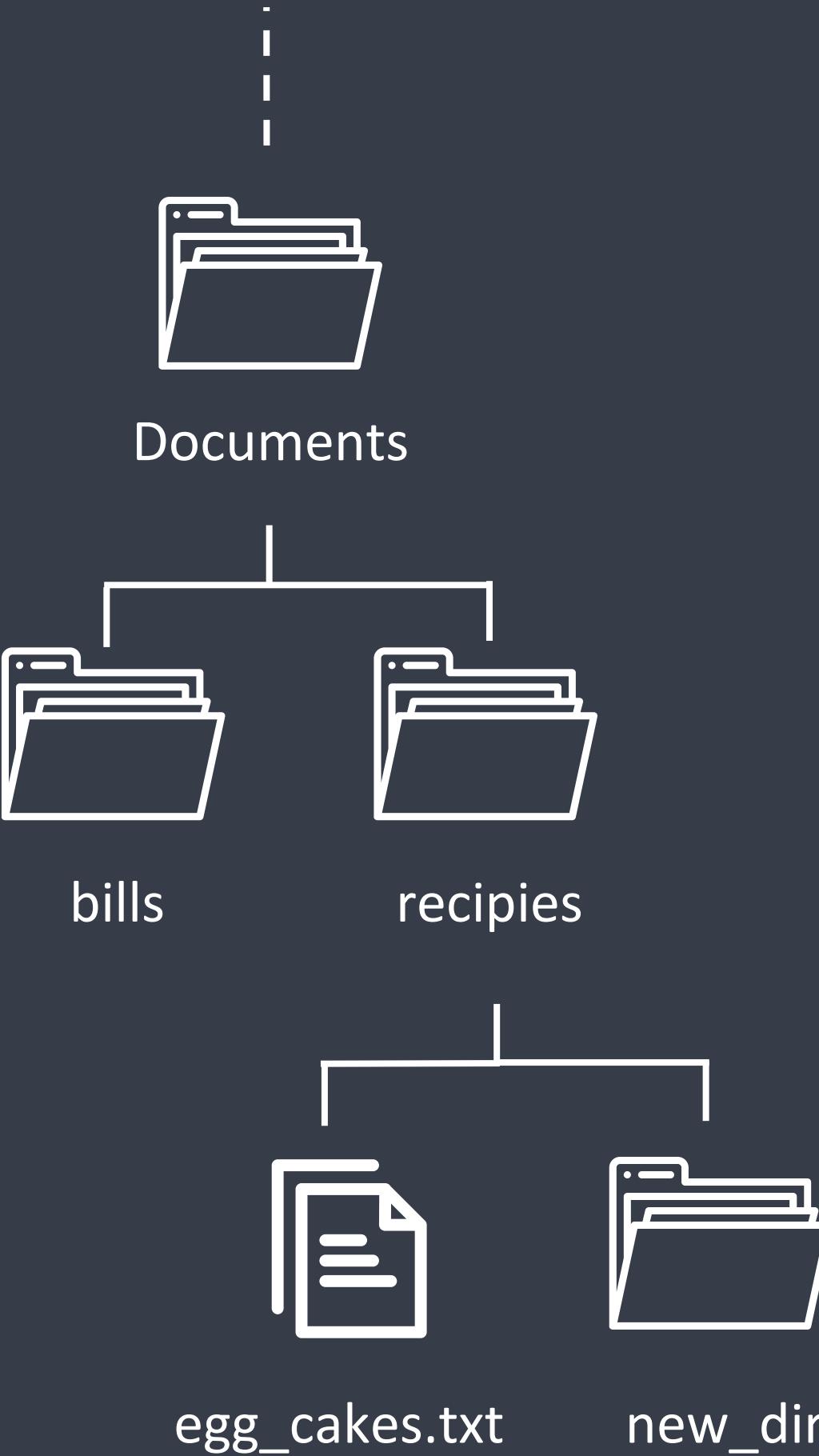
CREATING FILES AND DIRECTORIES

- To create a new directory we use **mkdir**:

```
$ mkdir new_dir
```

- The new directory will not have any content.
- There are many ways to create files, mostly as output from programs, but you can always create an empty file by using **touch**:

```
$ touch a_new_file.txt
```



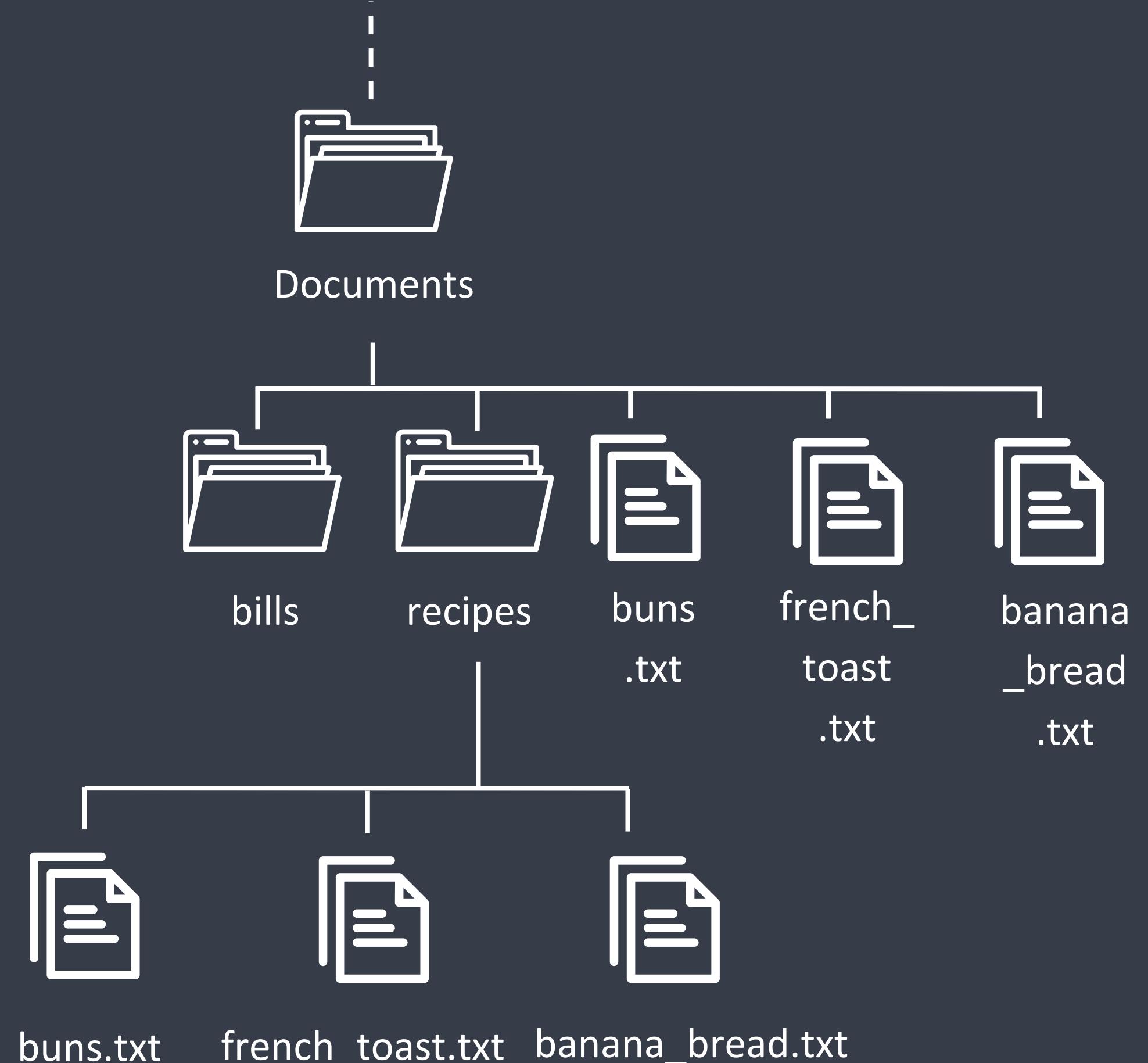
OPERATING ON SEVERAL FILES

- Above we have seen how to interact with single files and directories. But we can also interact with several at once!

- This would take all files named something .txt and move them one directory up:

```
$ mv *.txt .. /
```

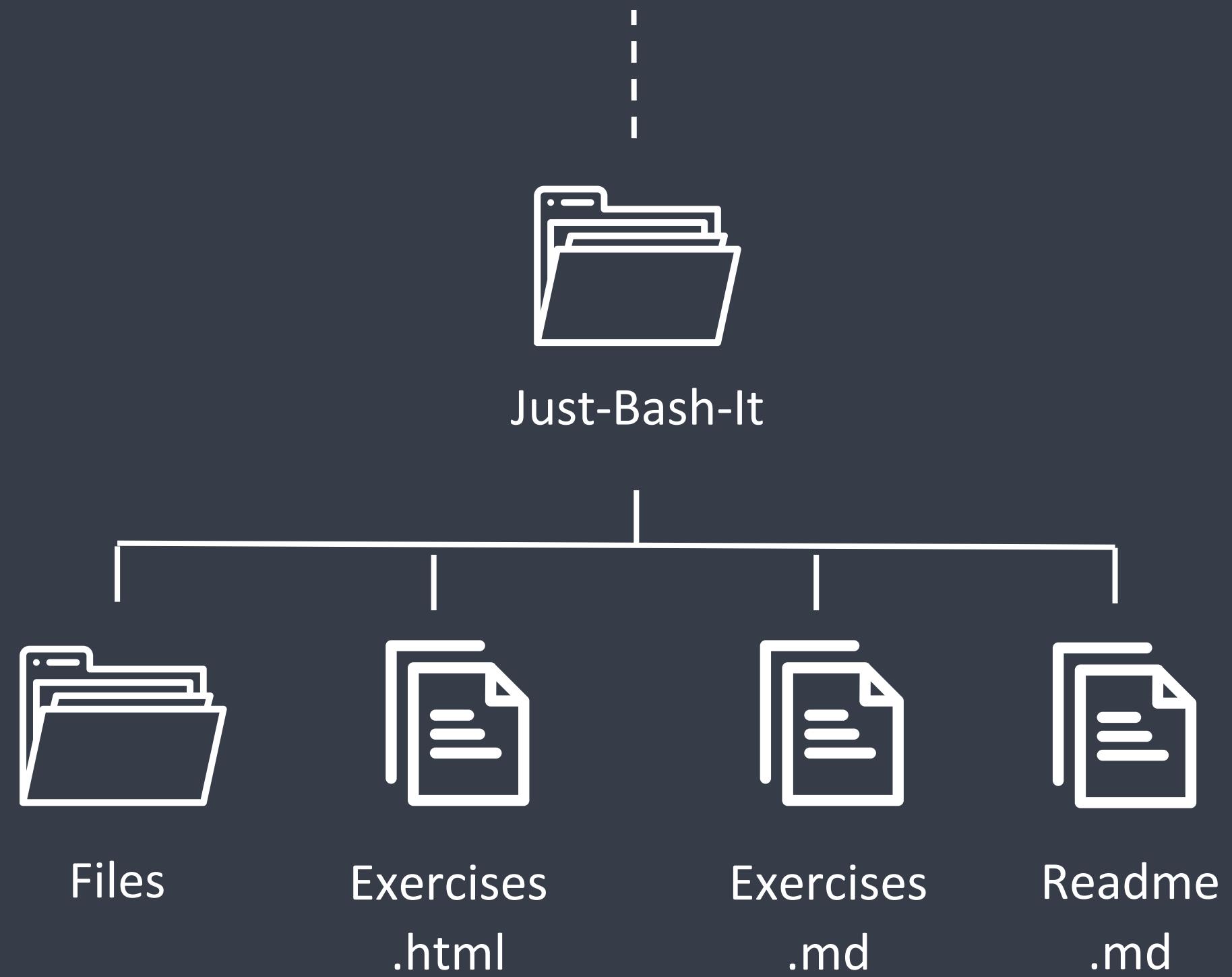
- The star '*' is called a wildcard. It matches every character (letter, number, ect).



OPERATING ON SEVERAL FILES

- You can also use the wildcard '*' to restrict actions to certain files and directories.
- By default 'ls' lists the entire directory. If we only want to see files with names ending in md we can specify that like this:

```
$ ls *.md
```



CHEAT SHEET 2

```
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
```

WHERE & WHAT

```
rm [name] # remove file  
rm -r [name] # remove dir  
cp [name] # copy a file/dir  
mv [name] [path] # move file/dir  
mkdir [name] # make dir  
touch [name] # create file
```

FILE/DIR BASICS



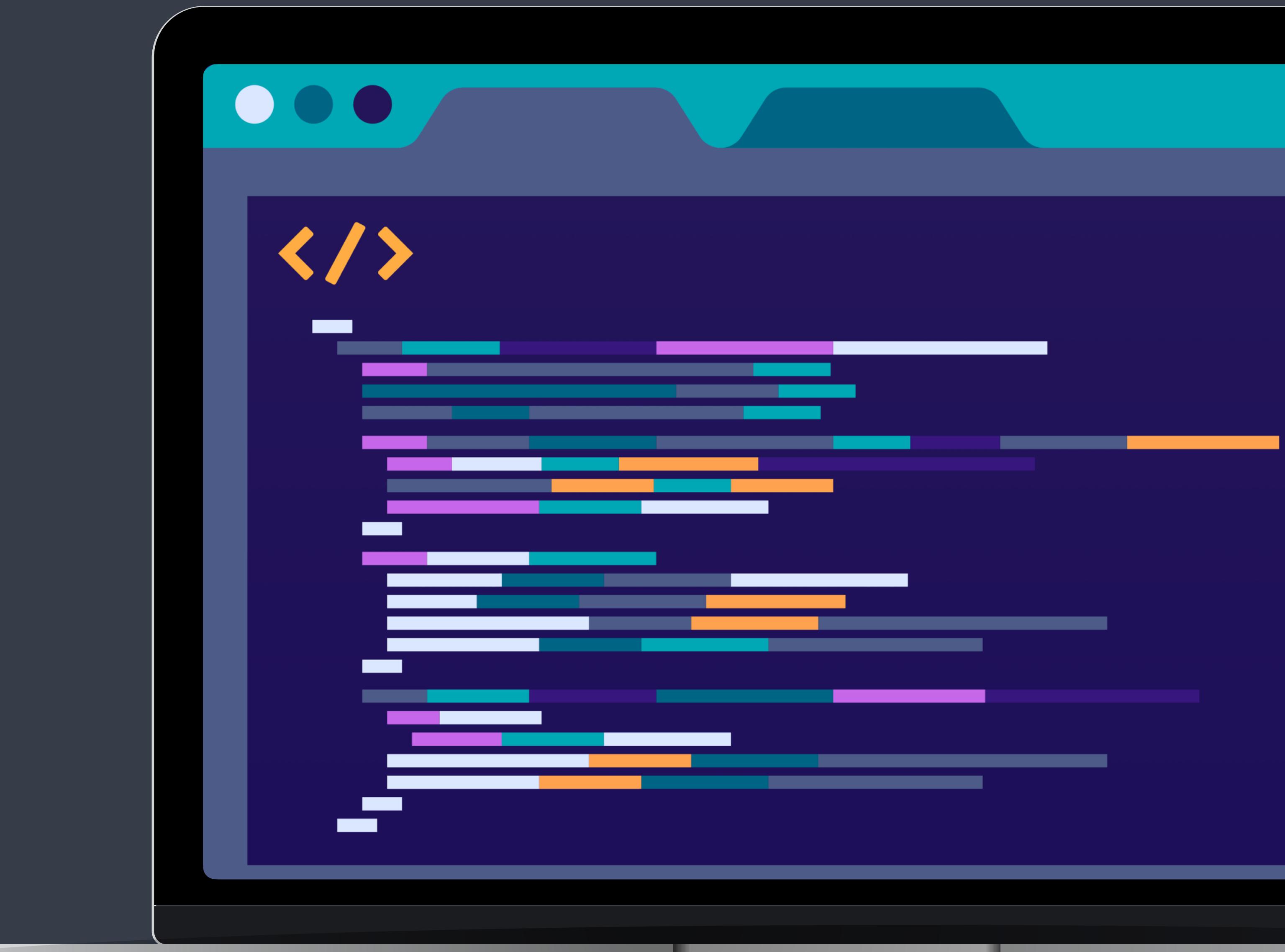
```
Last login: Fri Jun 24 15:05:34 on ttys000  
[kgx936@SUN1007442 ~ % echo "Just Bash It"  
Just Bash It  
kgx936@SUN1007442 ~ % ]
```

“

Let's Bash It!

Time for **Exercise 2!**

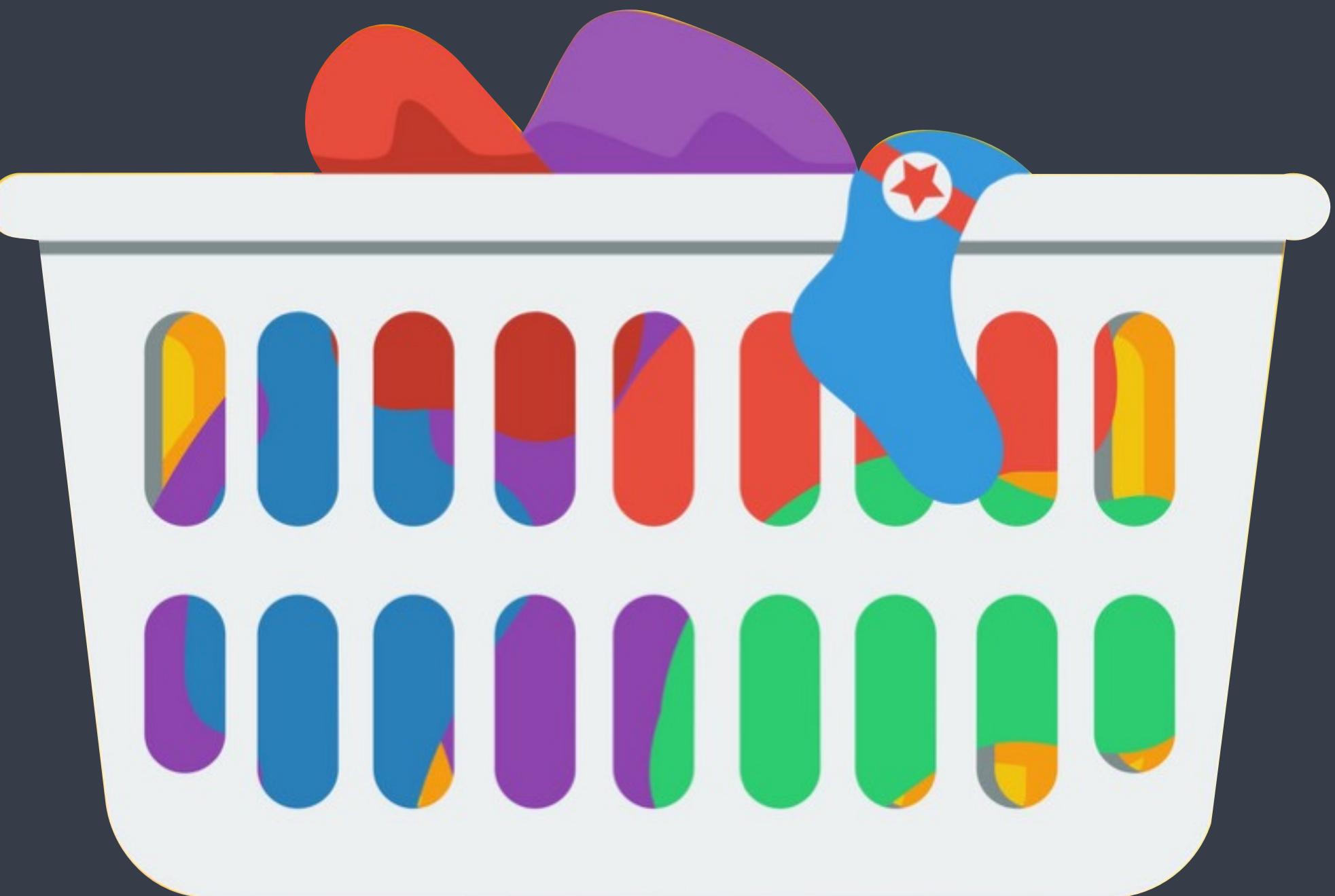
3. PROJECT ORGANIZATION & BACKUP



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).



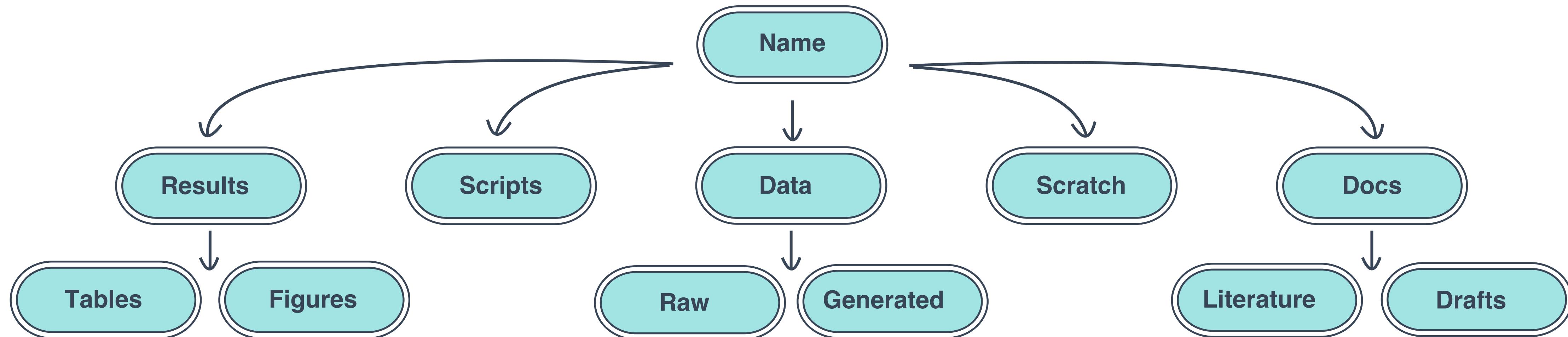
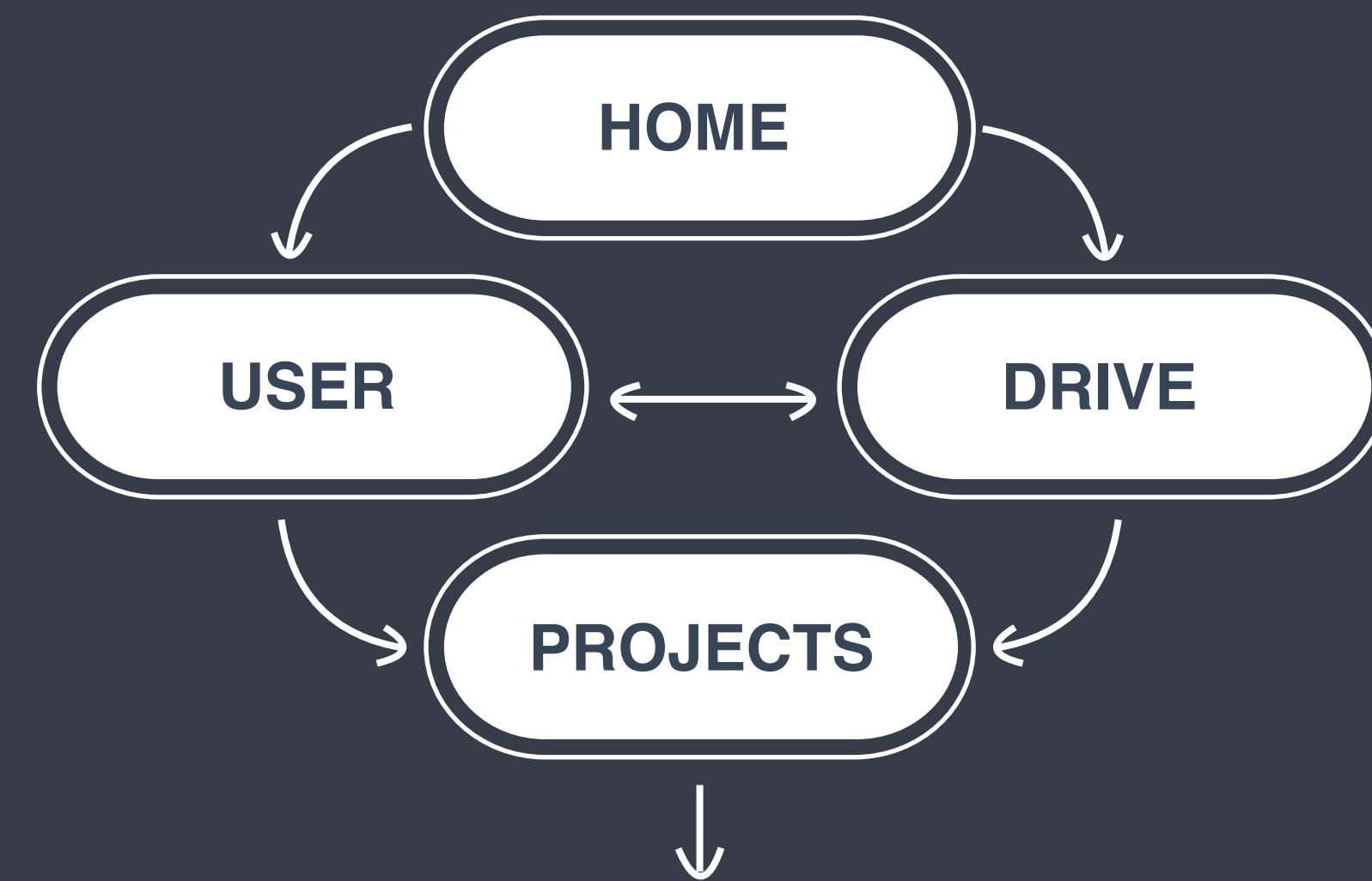
Don't Store Large Data/Files on Your Local Computer



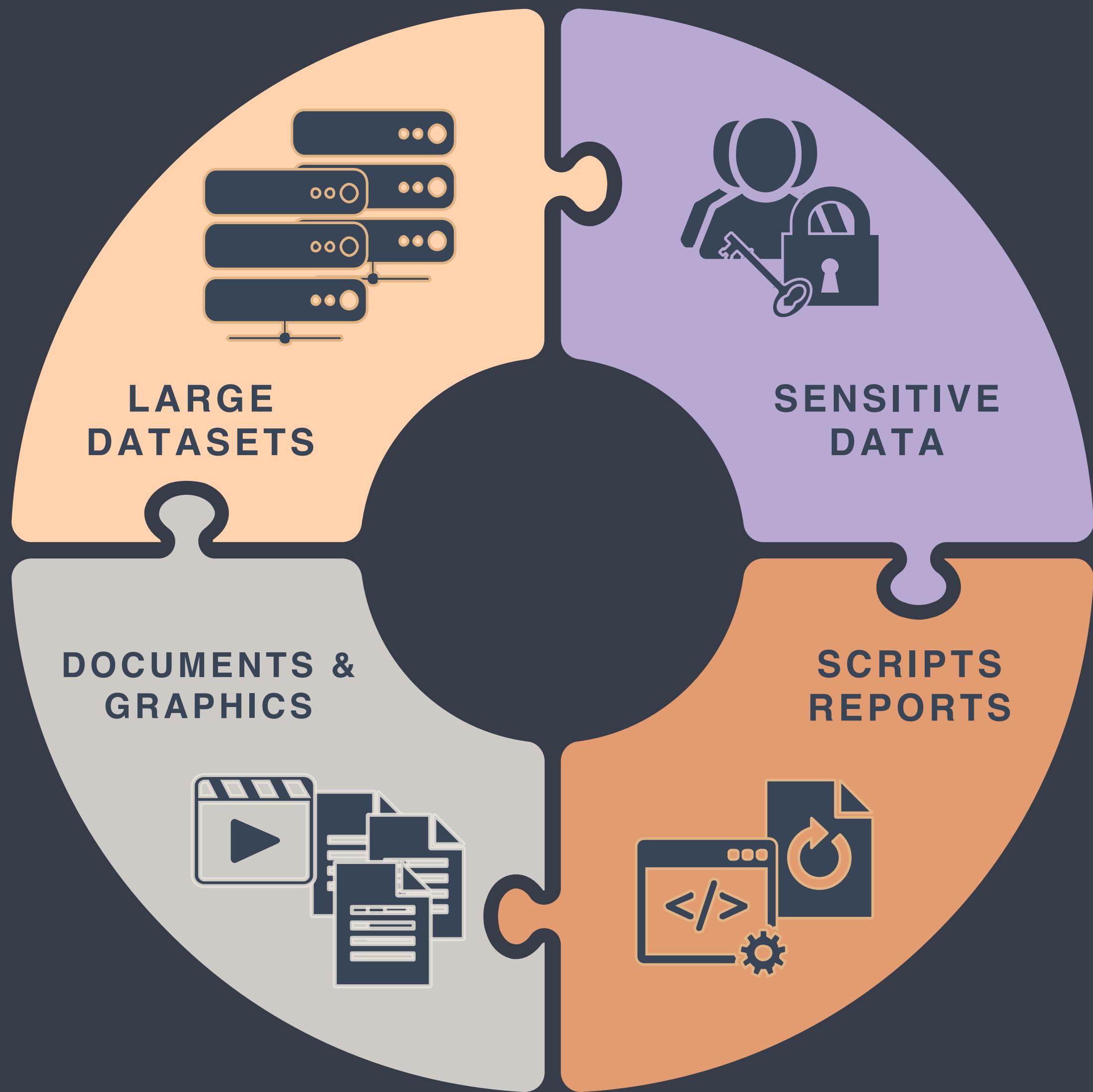
Don't Store Sensitive Datasets on Your Computer

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>

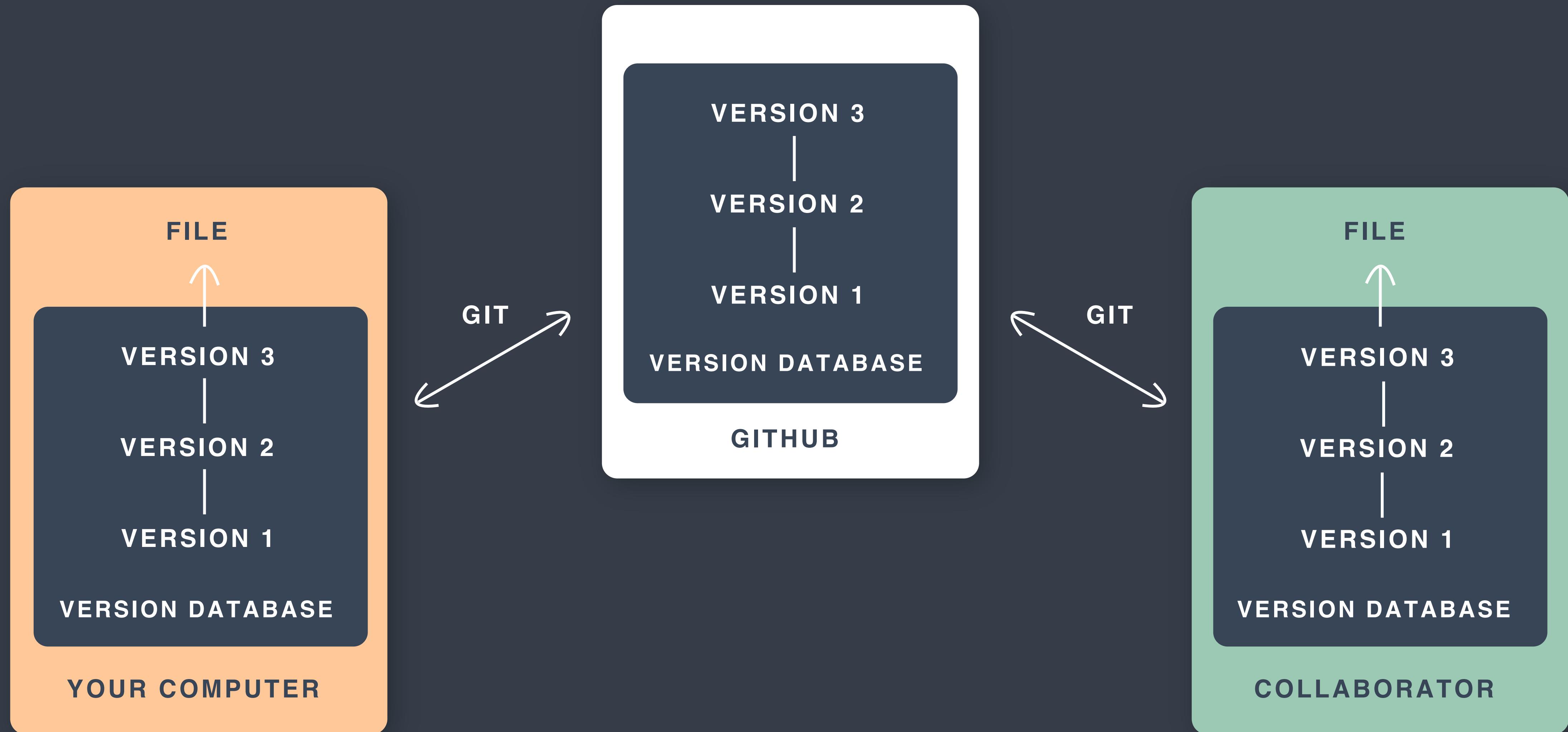
GitHub is a web-based hosting service for version control that uses git

Other services exist.

<https://github.com>



WHAT IS A VERSION CONTROL SYSTEM?

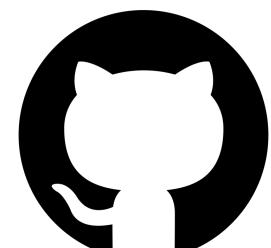


GIT WORKS FROM THE COMMAND LINE

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)`

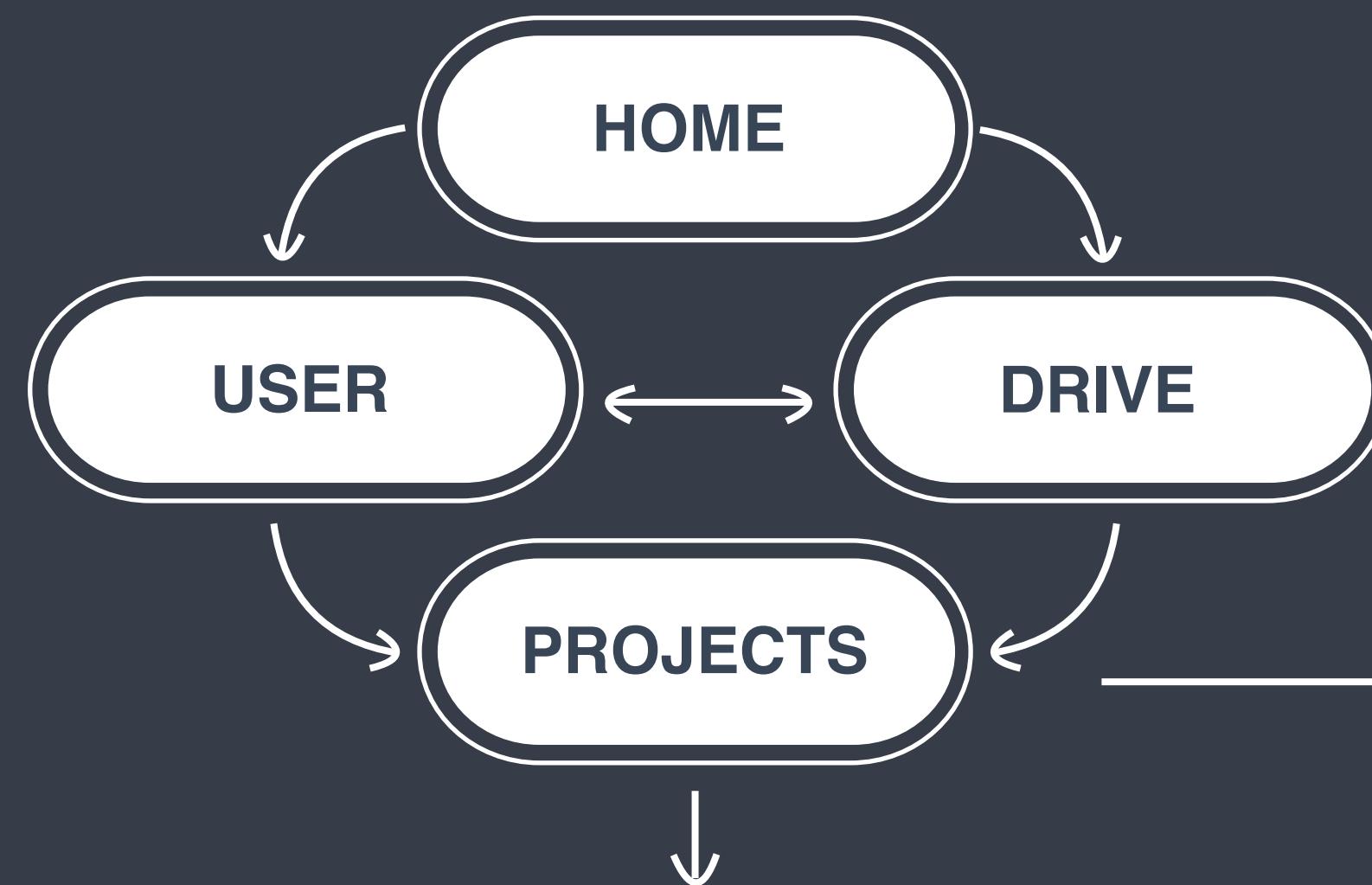
`git add (add new files or changes)`

`git commit -m (commit the changes)`

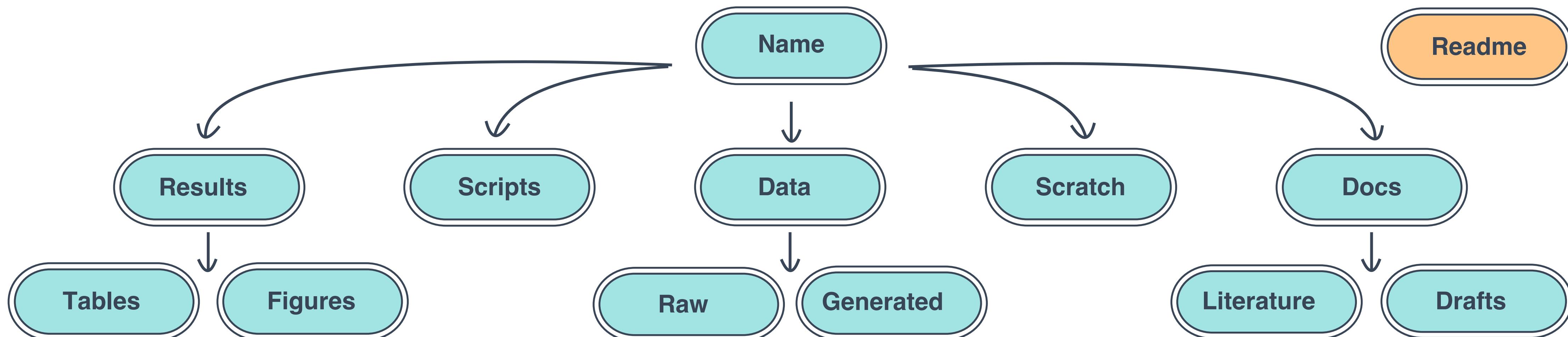
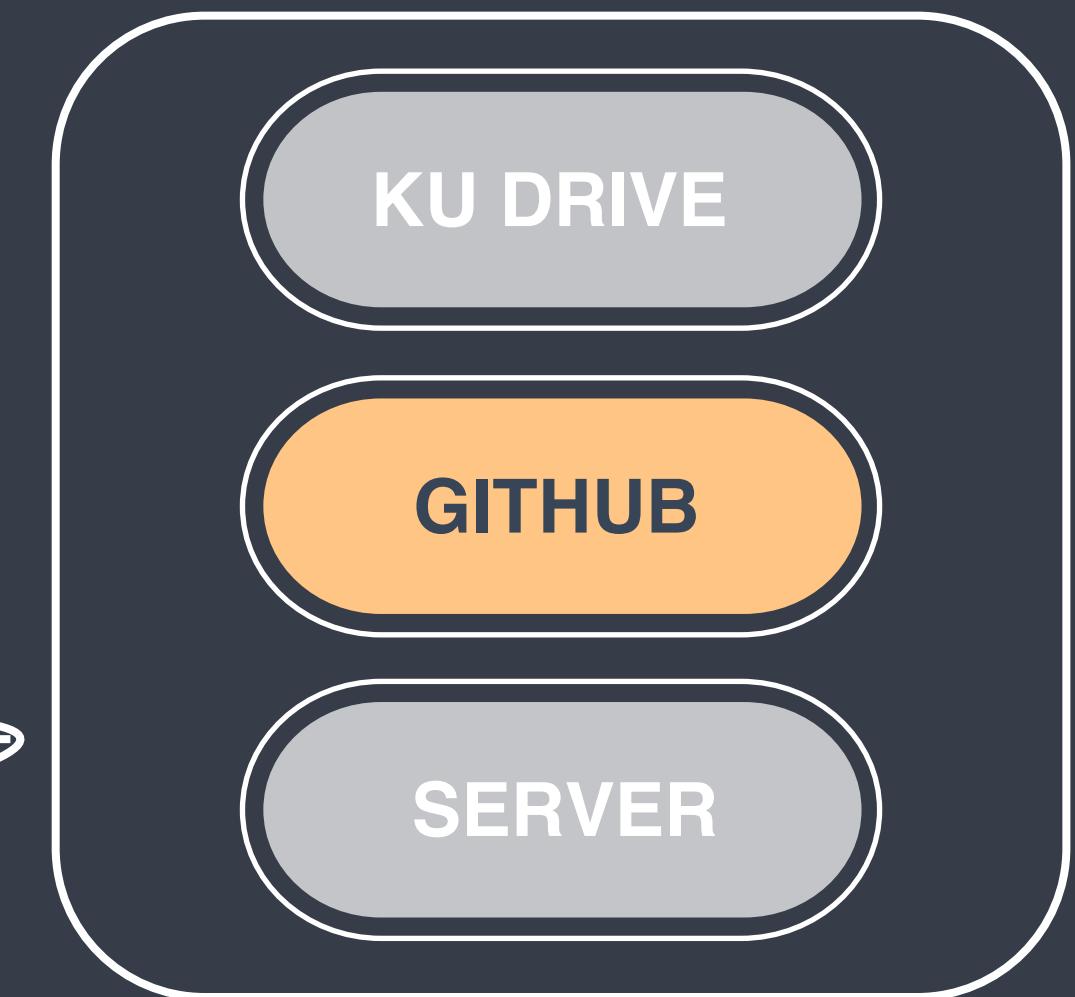
`git push (push the changes to GitHub)`

A SUGGESTION FOR STRUCTURE

LOCAL COMPUTER

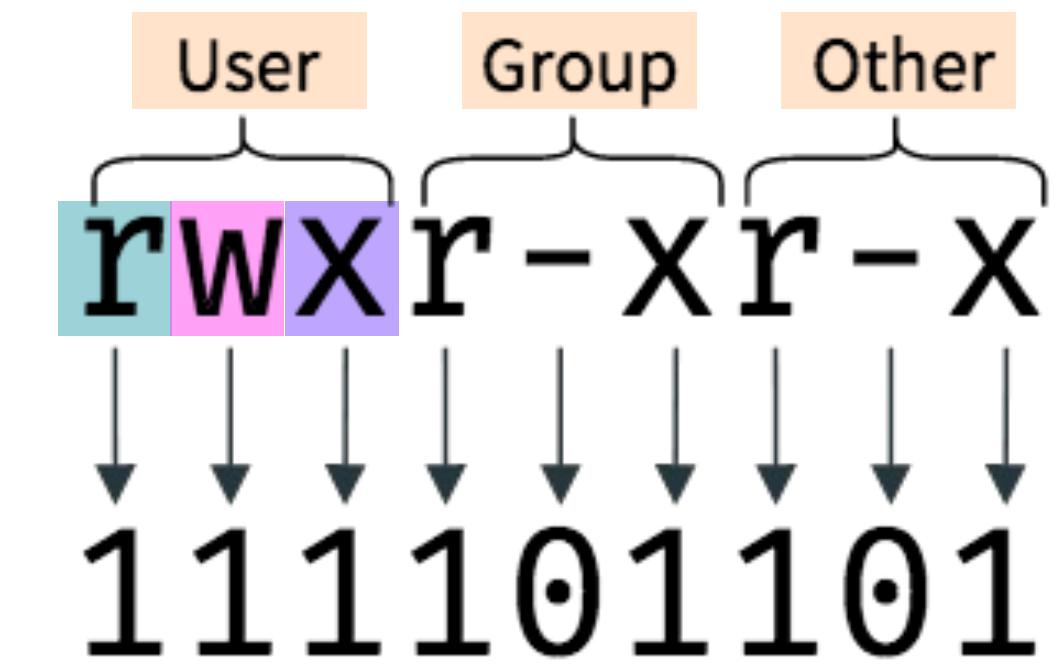


EXTERNAL BACKUP



USERS, GROUPS & PERMISSIONS

- Permission settings denote who can **read**, **write** (**edit**) and **execute** a file/dir
- On UNIX-based systems (Mac, Ubuntu) you can use the `chmod` command to set permissions **IF** you 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
- On Windows `chmod` **does not work** as the file system is set up differently. There are alternatives to set permissions with powershell (you can google this if needed).

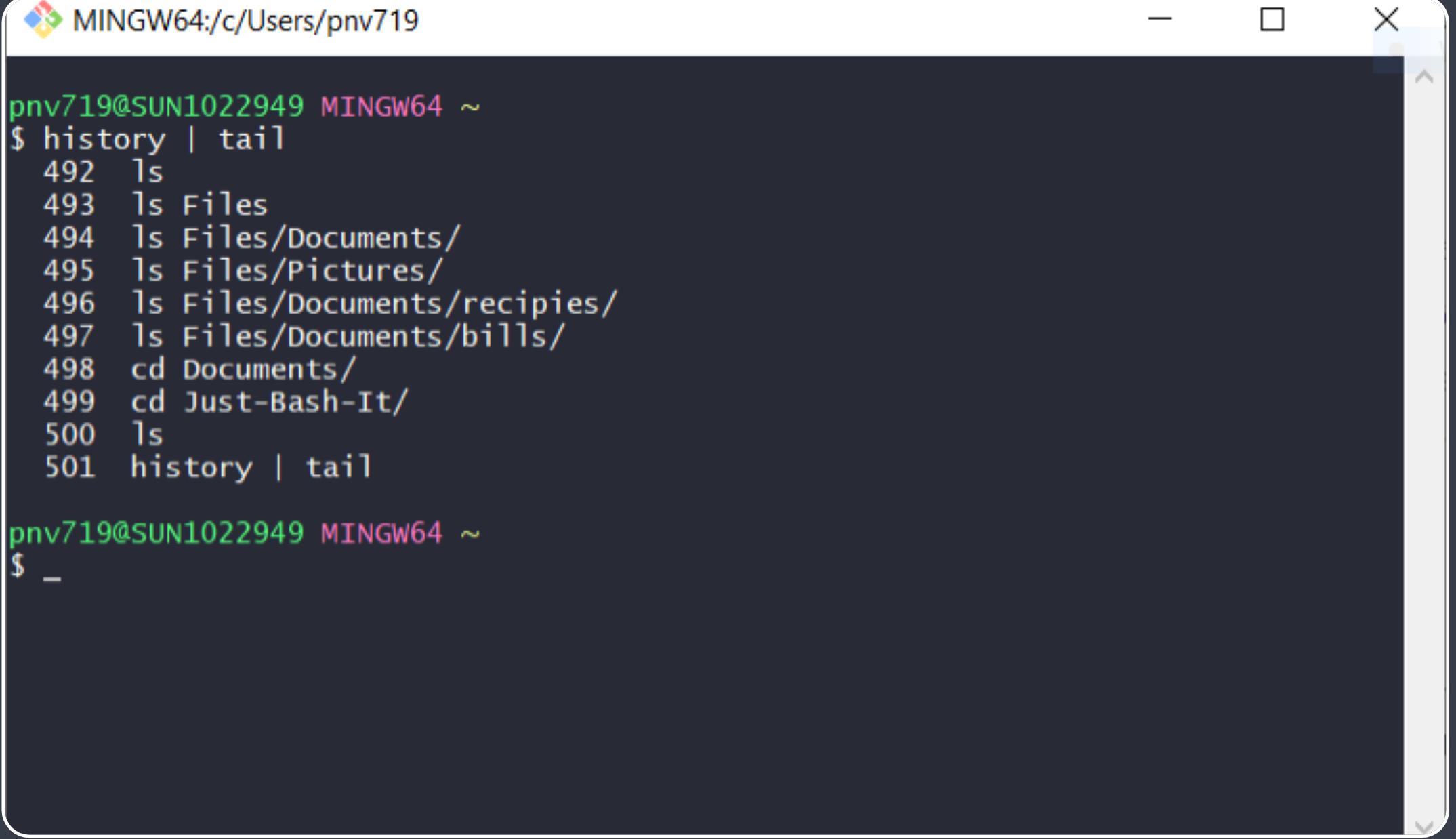


```
chmod +rwx [filename]  
chmod -wx [directoryname]  
  
chmod u+rwx [filename]  
chmod g-w [filename]  
chmod o-rwx [filename]
```

BACK TO THE PAST

“Those who don’t know history are doomed to retype their commands.”

- 'history' command brings up a list of previously used commands
- The most recent command is last, the oldest command is first
- Can use head/tail to only see the beginning/end



The screenshot shows a terminal window titled 'MINGW64:/c/Users/pnv719'. The window contains the following text:

```
pnv719@SUN1022949 MINGW64 ~  
$ history | tail  
492 ls  
493 ls Files  
494 ls Files/Documents/  
495 ls Files/Pictures/  
496 ls Files/Documents/recipies/  
497 ls Files/Documents/bills/  
498 cd Documents/  
499 cd Just-Bash-It/  
500 ls  
501 history | tail
```

Below this, there is another prompt:

```
pnv719@SUN1022949 MINGW64 ~  
$ -
```

Pro tip:
Use the 'grep' command to search your history or bring up the interactive 'reverse-i-search' with ctrl+r.

CHEAT SHEET 3

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+rwx [name] # specify who, add permission
```

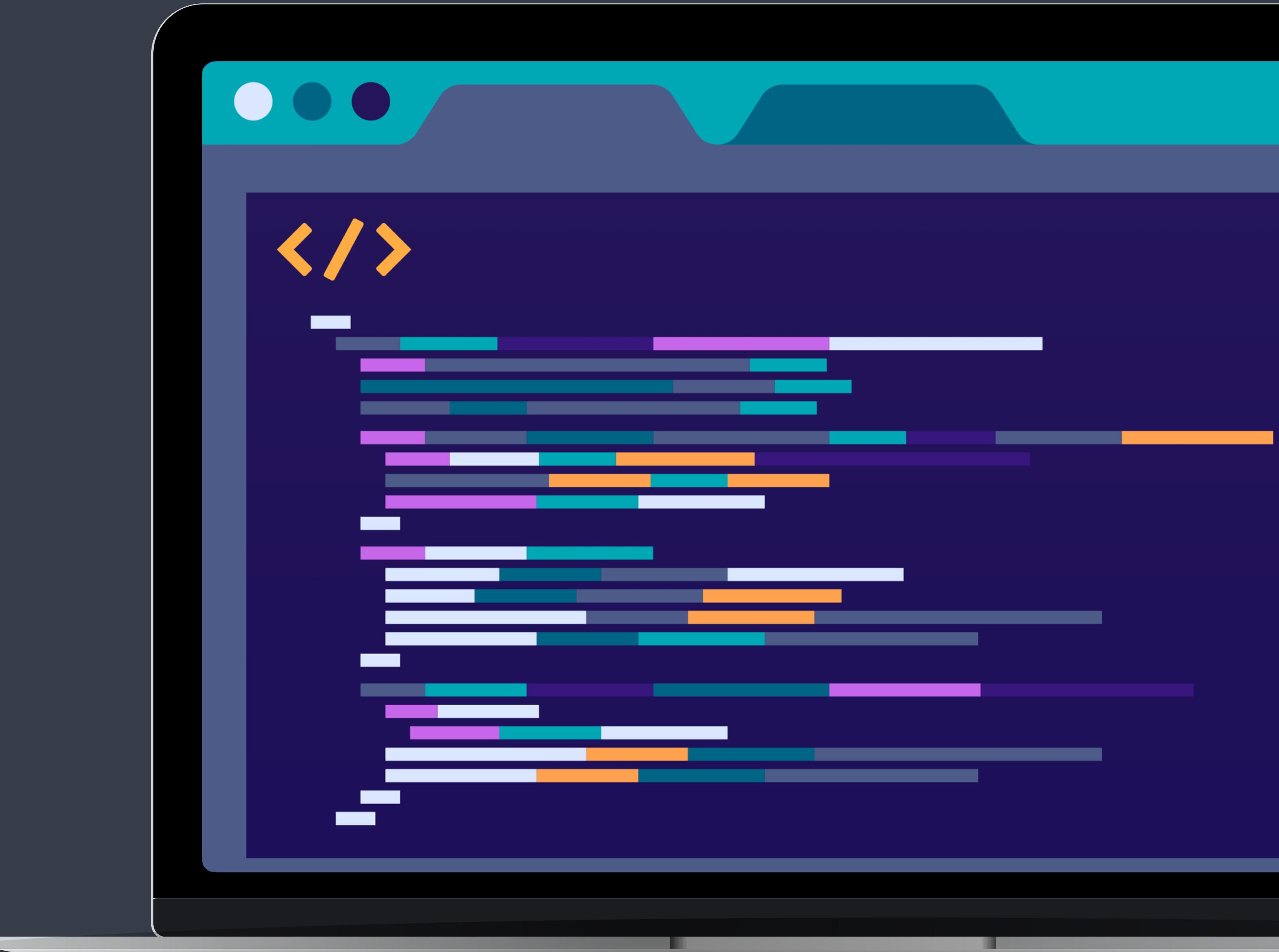


```
Last login: Fri Jun 24 15:05:34 on ttys000
[kgx936@SUN1007442 ~ % echo "Just Bash It"
Just Bash It
kgx936@SUN1007442 ~ % ]
```

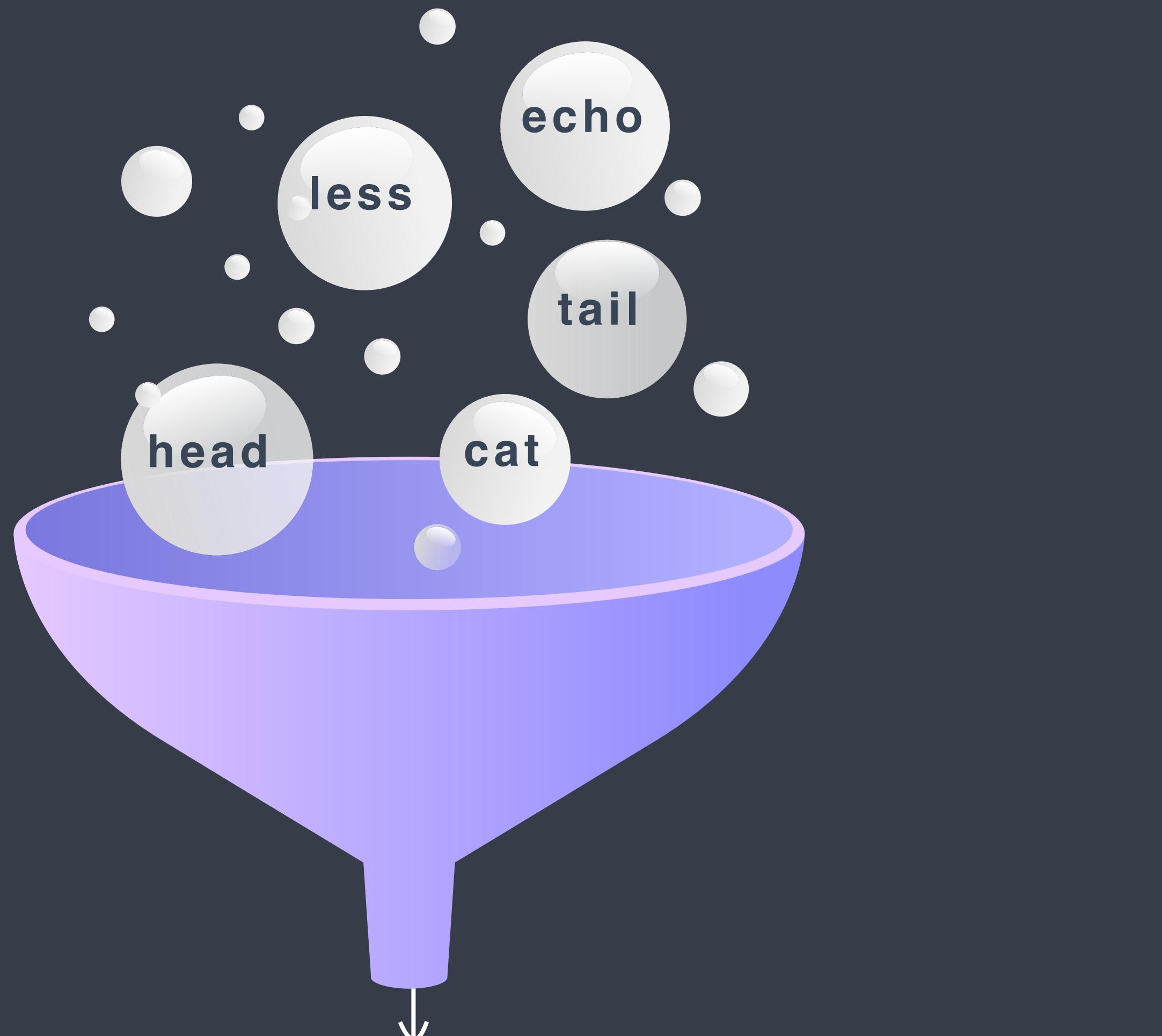
“

Exercise 3, you know the drill...

4. VIEWING AND EDITING 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
```

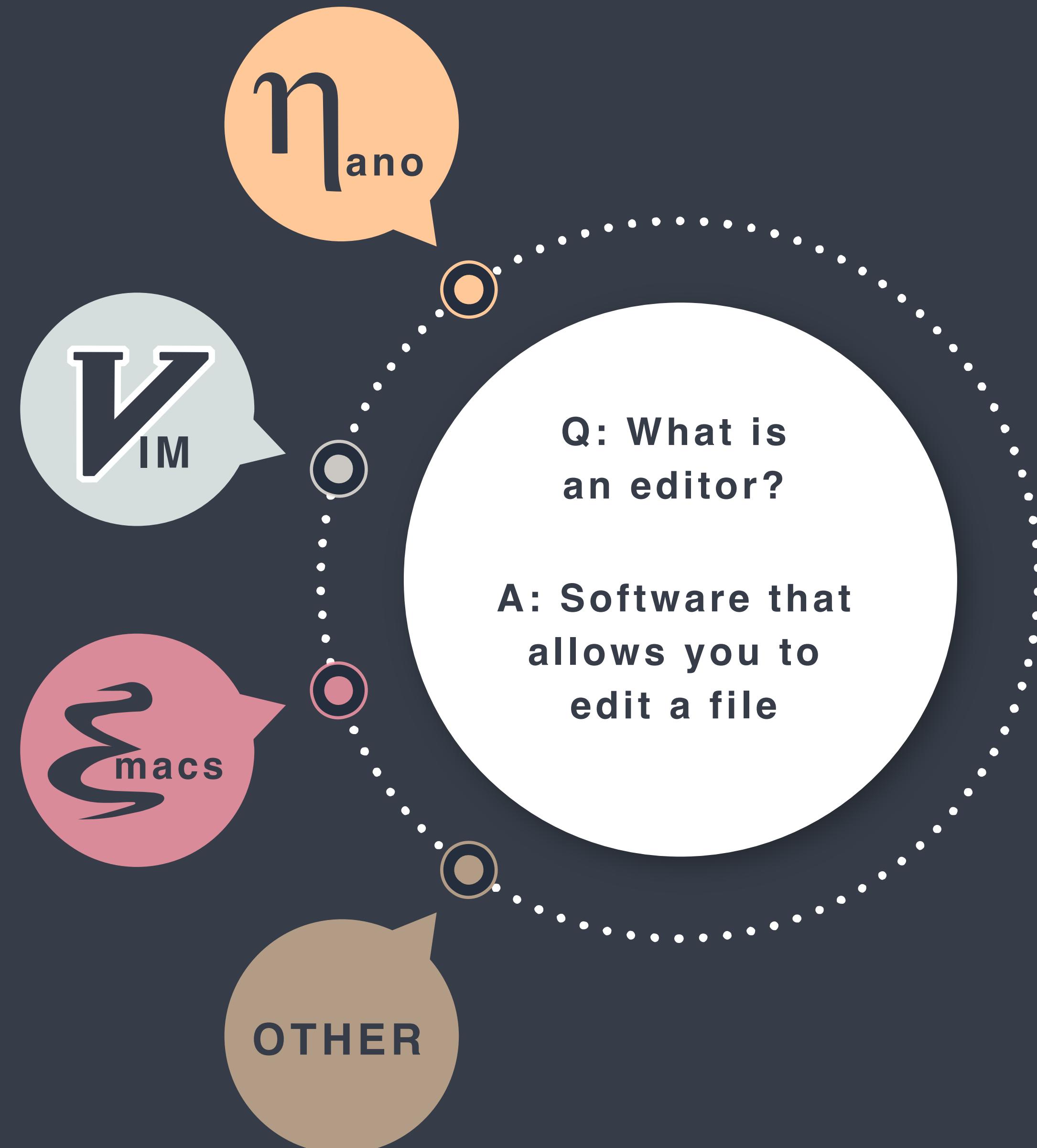
```
head -n 20 myfile.txt > myfile_small.txt
```

> = 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 + c (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. Compresses multiple files and puts them into one, single file archive.



Fun fact: tar stands for tape archive



CHEAT SHEET 4

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/
```

Quit & Save

```
Ctrl + c # quit process  
q # quit less command  
  
:q (:w) # quit (and save) vim editor  
Ctrl + x (Y + enter) # quit (and save) nano editor
```

Compressed Files

```
gzip [file] # Create compressed gzip file  
gunzip [file] # Unpack compressed gzip file  
gunzip -k [file] # Unpack but also keep compressed file  
zip/unzip [file] # create/decompress .zip files  
zless -[f] [file] # view compressed file w/o decompression  
tar -cvf [file.tar.gz] [dir] # Create a compressed tar archive of [dir] at [file.tar.gz]  
tar -xvf [file.tar.gz] # Unpack a compressed tar archive  
tar -tf [file.tar.gz] # List files inside a compressed tar archive
```

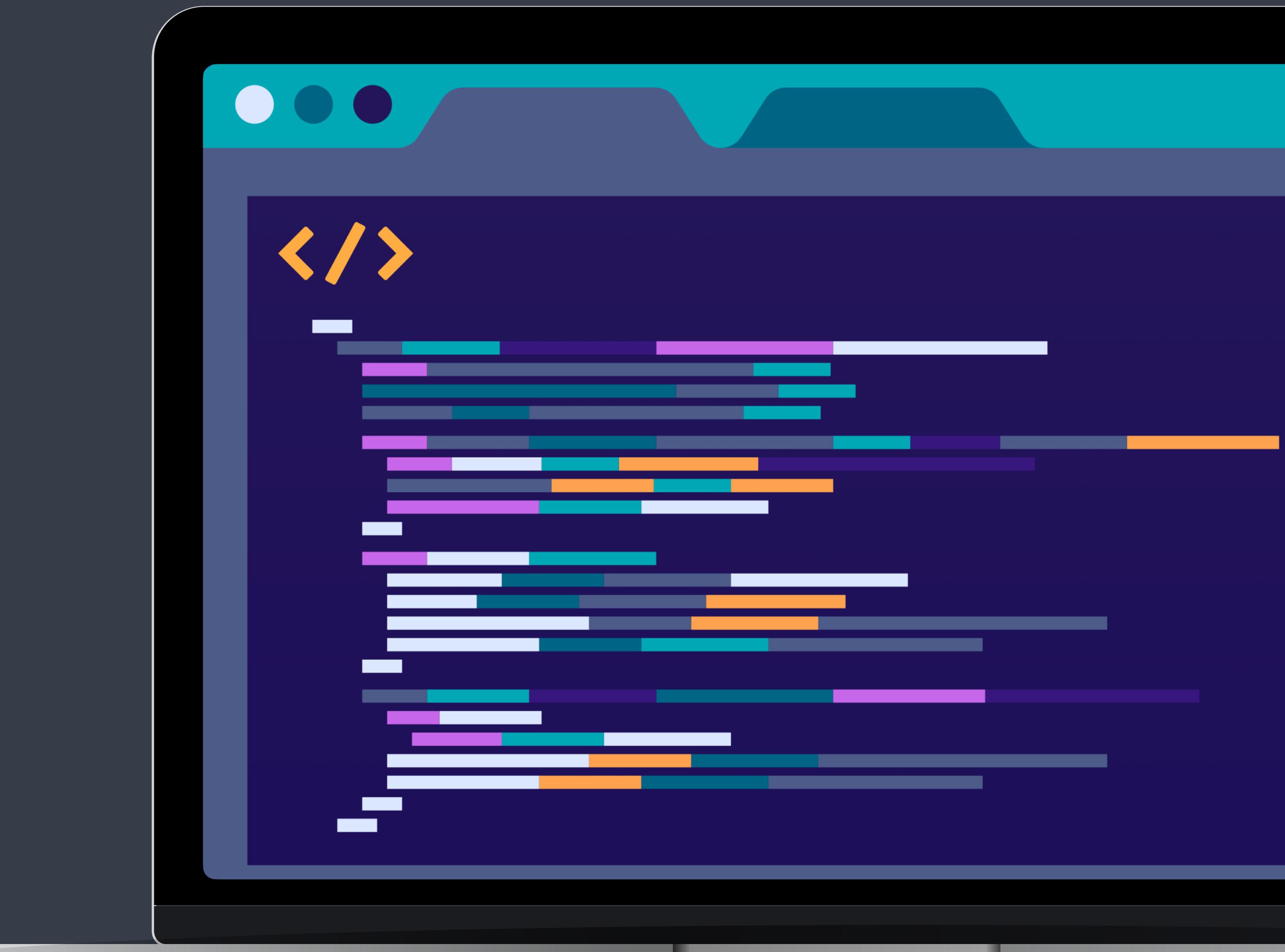


```
Last login: Fri Jun 24 15:05:34 on ttys000
[kgx936@SUN1007442 ~ % echo "Just Bash It"
Just Bash It
kgx936@SUN1007442 ~ % ]
```

“

Refreshed from lunch we do
Exercise 4!

5. DATA WRANGLING 1



FROM FILE NAVIGATION TO MANIPULATION

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



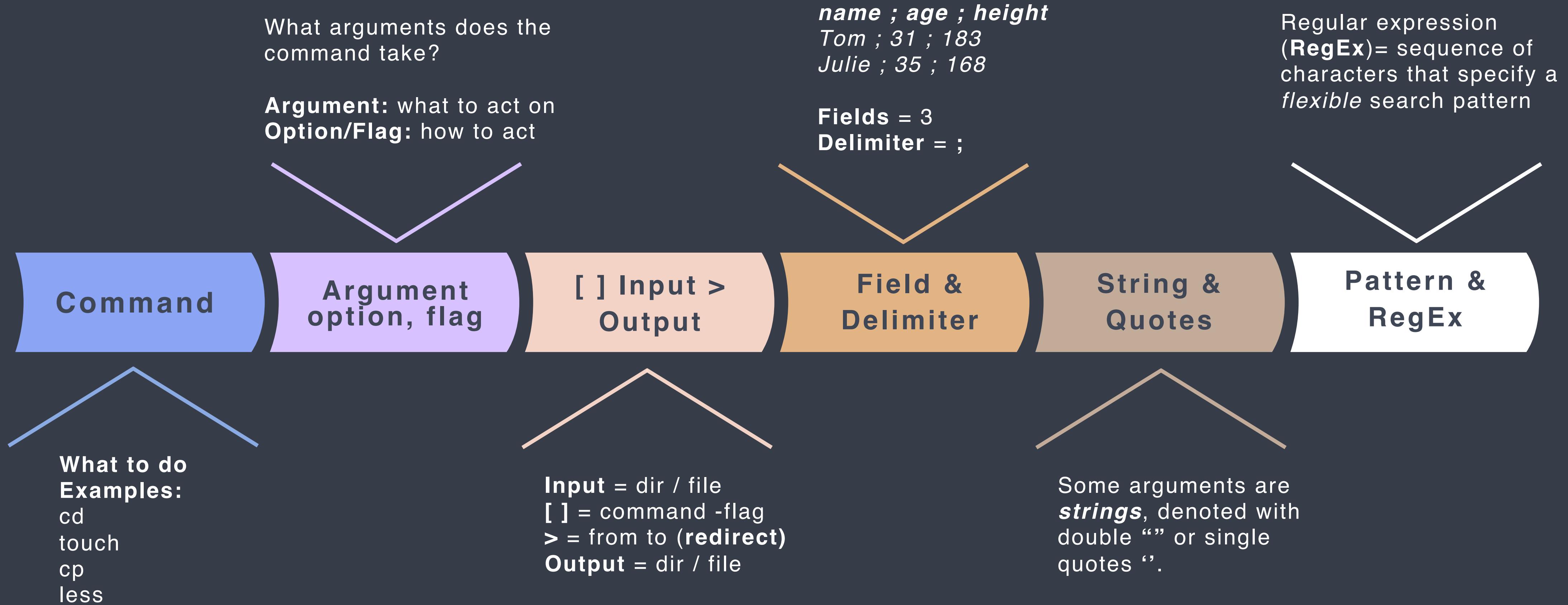
NAVIGATE THE TERMINAL
&
FILE MANAGEMENT

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



FILE MANIPULATION,
WRANGLING, SUMMATION

A LITTLE TERMINOLOGY



BASH COMMANDS

Sort
Sort a file/column
Number, character, mixed

Sort

uniq

Report unique entries only

Search & Extract
pattern in file

grep

Search, replace, extract, manipulate
awk is tool & a command

awk

Paste corresponding or
subsequent lines of files

paste

sed

Find and replace,
insertion or deletion

Select field of each line

cut

wc

Count lines & words

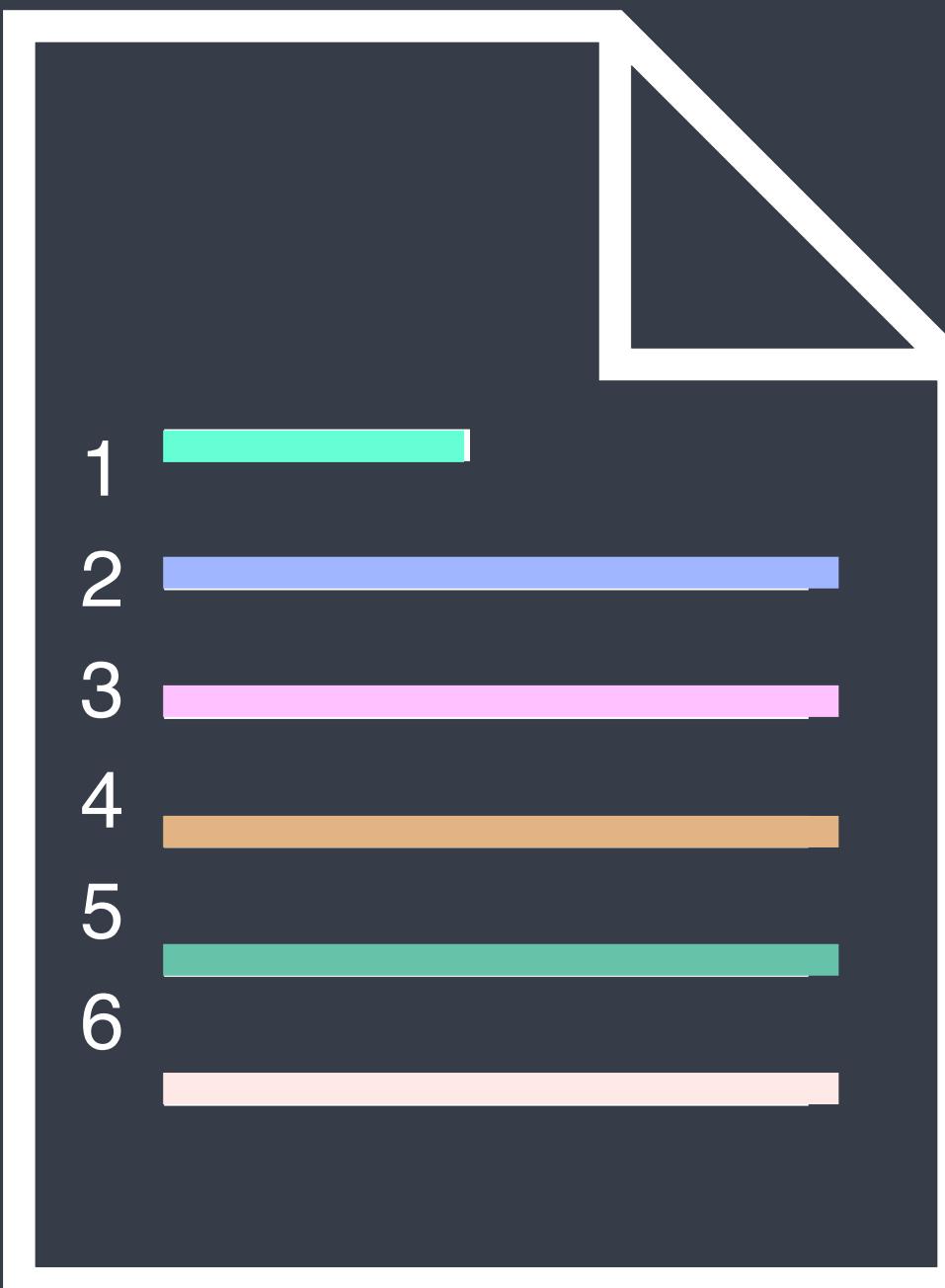
WC – NO..., NOT THAT ONE

- Word count (wc) lets us know how many words, lines and bytes there are in a file.

```
$ wc patients.txt
```

- We can also specify which count we want. For example only the lines:

```
$ wc -l patients.txt
```



CUT – GETTING COLUMNS OUT

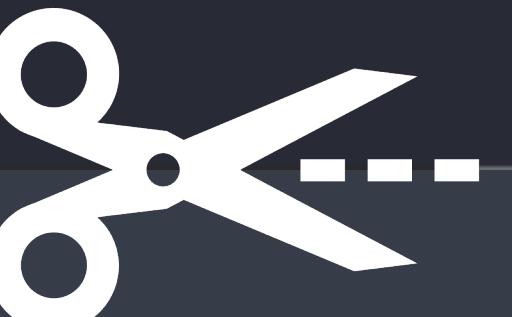
- Cut slices a column out of a file. What a ‘column’ is, depends on the field separator:

```
$ cut -f 1 -d ',' patients.txt
```

- We can also specify a range of columns:

```
$ cut -f 1-3 -d ',' patients.txt
```

- The opposite of **cut** is **paste**, putting columns together.



```
MINGW64:/c/Users/pnv719/Documents/Just-Bash-It/Fi... — □ ×
pnv719@SUN1022949 MINGW64 ~/Documents/Just-Bash-It/Files/Data
(june_2023)
$ head patients.txt
patient_ID,technique,hospital,conv_days,age,satisfaction
402109,A,Rigshospitalet,15,68,3
092070,A,Rigshospitalet,13,74,5
994082,B,Herlev,27,76,2
843094,A,Herlev,30,65,5
369360,B,Rigshospitalet,21,68,5
688213,B,Rigshospitalet,29,77,3
197347,A,Herlev,25,65,5
759063,B,Rigshospitalet,16,75,4
121219,B,Herlev,27,68,4

pnv719@SUN1022949 MINGW64 ~/Documents/Just-Bash-It/Files/Data
(june_2023)
$ cut -f 1 -d ',' patients.txt
patient_ID
402109
092070
994082
843094
369360
688213
```

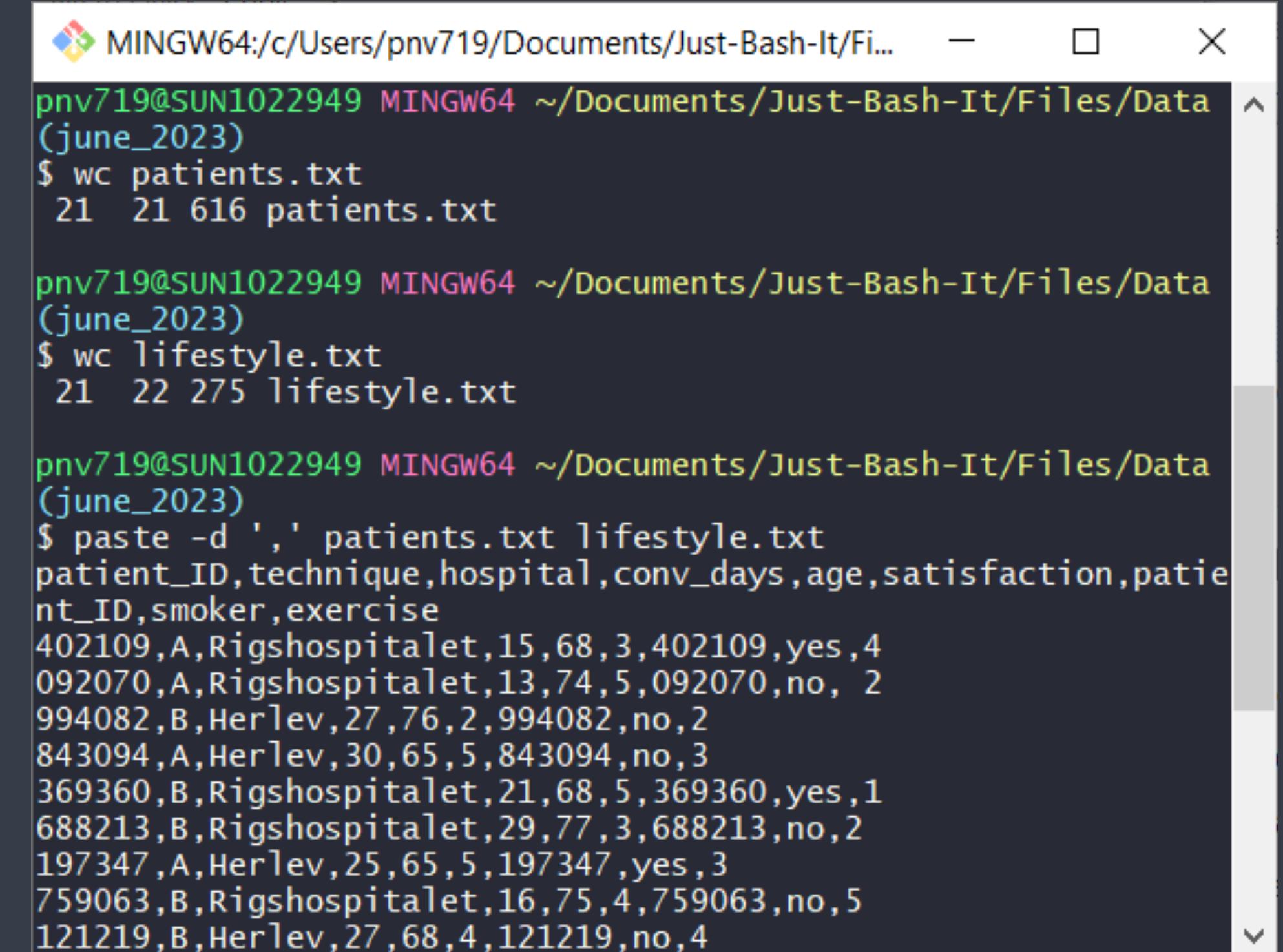
PASTE – GLUE THINGS TOGETHER

- Paste two or more files together line by line. Default delimiter is space:

```
$ paste patients.txt lifestyle.txt
```

- Specify another delimiter for pasting together files:

```
$ paste -d ',' patients.txt lifestyle.txt
```



The screenshot shows a terminal window titled 'MINGW64:/c/Users/pnv719/Documents/Just-Bash-It/Fi...' with a process ID of 'pnv719@SUN1022949 MINGW64 ~/Documents/Just-Bash-It/Files/Data (june_2023)'. It displays the results of two 'wc' commands: one for 'patients.txt' showing 21 lines, 21 words, and 616 characters; and another for 'lifestyle.txt' showing 21 lines, 22 words, and 275 characters. Below these, the output of the 'paste' command is shown, consisting of 10 lines of patient data. Each line contains 8 fields separated by commas: patient_ID, technique, hospital, conv_days, age, satisfaction, patient_ID, smoker, and exercise. The data includes various values such as 'A', 'B', and numerical values like '15,68,3,402109, yes, 4'.

patient_ID	technique	hospital	conv_days	age	satisfaction	patient_ID	smoker	exercise
402109	A	Rigshospitalet	15,68,3,402109	yes, 4				
092070	A	Rigshospitalet	13,74,5,092070	no, 2				
994082	B	Herlev	27,76,2,994082	no, 2				
843094	A	Herlev	30,65,5,843094	no, 3				
369360	B	Rigshospitalet	21,68,5,369360	yes, 1				
688213	B	Rigshospitalet	29,77,3,688213	no, 2				
197347	A	Herlev	25,65,5,197347	yes, 3				
759063	B	Rigshospitalet	16,75,4,759063	no, 5				
121219	B	Herlev	27,68,4,121219	no, 4				

SED – THE STEAM EDITOR

Sed is a powerful tool used to manipulate files without the manual labor of a text editor.

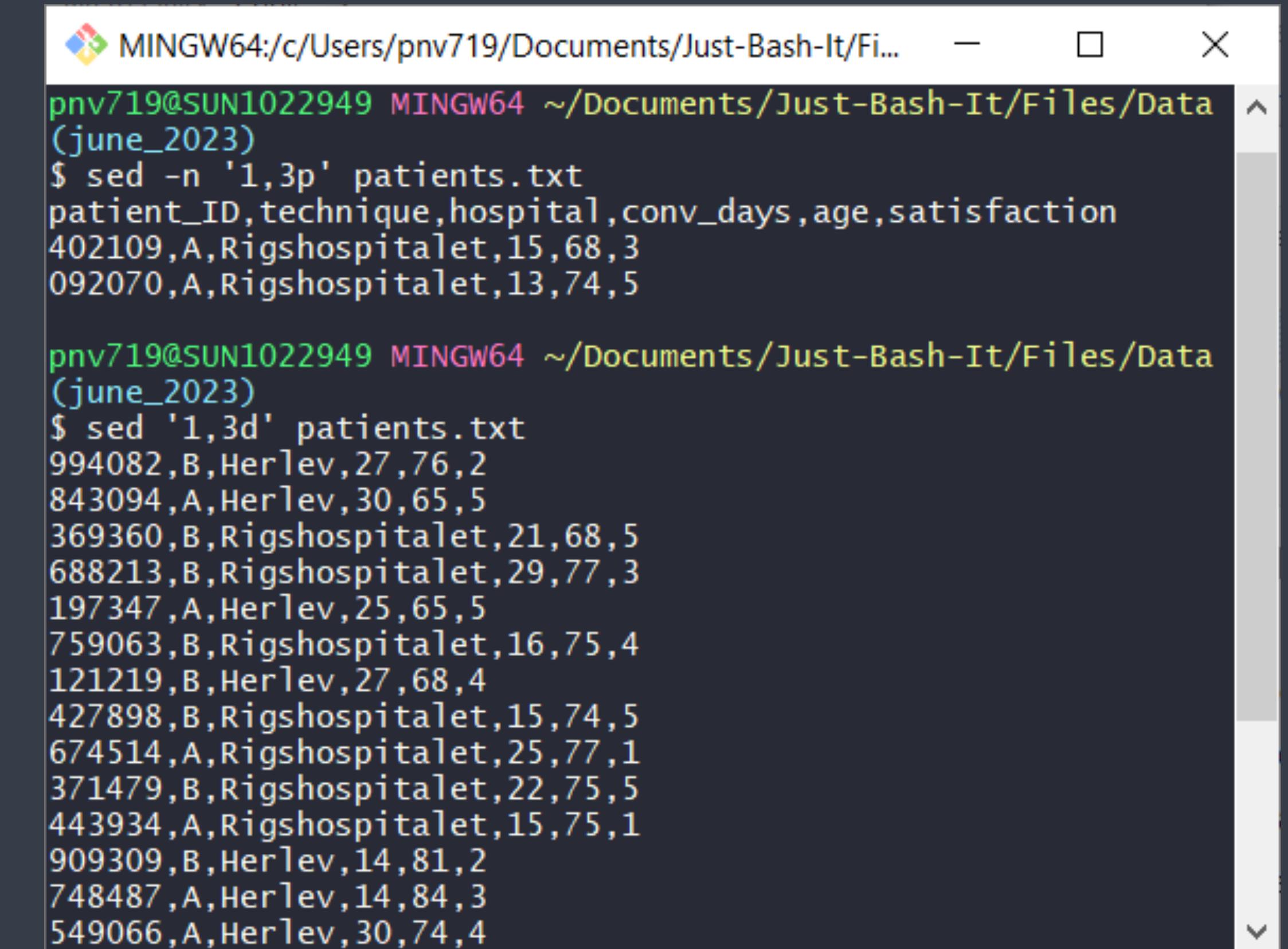
- A common use it to display only specific lines:

```
$ sed -n '1,3p' patients.txt
```

- Or to delete them!

```
$ sed '1,3d' patients.txt
```

NB! We haven't saved the output, it is just displayed.



The screenshot shows a terminal window titled 'MINGW64;c/Users/pnv719/Documents/Just-Bash-It/Fi...'. The command \$ sed -n '1,3p' patients.txt is run, displaying the first three lines of the 'patients.txt' file. The command \$ sed '1,3d' patients.txt is run, deleting the first three lines of the file. The terminal window has a dark background with light-colored text and a scroll bar on the right.

```
pnv719@SUN1022949 MINGW64 ~/Documents/Just-Bash-It/Files/Data(june_2023)$ sed -n '1,3p' patients.txt
patient_ID,technique,hospital,conv_days,age,satisfaction
402109,A,Rigshospitalet,15,68,3
092070,A,Rigshospitalet,13,74,5

pnv719@SUN1022949 MINGW64 ~/Documents/Just-Bash-It/Files/Data(june_2023)$ sed '1,3d' patients.txt
994082,B,Herlev,27,76,2
843094,A,Herlev,30,65,5
369360,B,Rigshospitalet,21,68,5
688213,B,Rigshospitalet,29,77,3
197347,A,Herlev,25,65,5
759063,B,Rigshospitalet,16,75,4
121219,B,Herlev,27,68,4
427898,B,Rigshospitalet,15,74,5
674514,A,Rigshospitalet,25,77,1
371479,B,Rigshospitalet,22,75,5
443934,A,Rigshospitalet,15,75,1
909309,B,Herlev,14,81,2
748487,A,Herlev,14,84,3
549066,A,Herlev,30,74,4
```

One very powerful use of **sed** is to edit files with *regular expressions*

REGULAR EXPRESSIONS

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

Anchors				Quantifiers				Groups and Ranges	
^	Start of string, or start of line in multi-line pattern	*	0 or more	{3}	Exactly 3	.	Any character except new line (\n)		
\A	Start of string	+	1 or more	{3,}	3 or more	(a b)	a or b		
\$	End of string, or end of line in multi-line pattern	?	0 or 1	{3,5}	3, 4 or 5	(...)	Group		
\Z	End of string	Add a ? to a quantifier to make it ungreedy.				(?:...)	Passive (non-capturing) group		
\b	Word boundary					[abc]	Range (a or b or c)		
\B	Not word boundary					[^abc]	Not (a or b or c)		
\<	Start of word					[a-q]	Lower case letter from a to q		
\>	End of word					[A-Q]	Upper case letter from A to Q		
Character Classes								[0-7]	Digit from 0 to 7
\c	Control character					\x	Group/subpattern number "x"		
"Escaping" is a way of treating characters which have a special meaning in regular expressions literally, rather than as special characters.					Ranges are inclusive.				
https://cheatography.com/davechild/cheat-sheets/regular-expressions/					Regex Testers : https://regexr.com/ https://regex101.com/				

SED – THE STEAM EDITOR

Search and replace patterns with **sed**.

- Changing ‘pat’ to ‘bat’:

```
$ sed 's/pat/bat/g' patients.txt
```

- Using *anchors* we can be specific about the exact pattern we want to hit:

```
$ sed 's/^40/xx/g' patients.txt
```

You should be as specific as possible to avoid changing the wrong part.

You can find many examples here:

<https://www.howtogeek.com/666395/how-to-use-the-sed-command-on-linux/>

The screenshot shows a terminal window titled 'MINGW64:/c/Users/pnv719/Documents/Just-Bash-It/Files/Data (june_2023)'. The user has run the command '\$ head patients.txt -n 5' which displays the first five lines of the 'patients.txt' file. The file contains patient data with columns: patient_ID, technique, hospital, conv_days, age, and satisfaction. The terminal then shows the result of running '\$ sed 's/pat/bat/g' patients.txt', where the word 'pat' has been replaced by 'bat' in all occurrences across the entire file.

patient_ID	technique	hospital	conv_days	age	satisfaction
402109	A	Rigshospitalet	15	68	3
092070	A	Rigshospitalet	13	74	5
994082	B	Herlev	27	76	2
843094	A	Herlev	30	65	5

patient_ID	technique	hospital	conv_days	age	satisfaction
402109	A	Rigshospitalet	15	68	3
092070	A	Rigshospitalet	13	74	5
994082	B	Herlev	27	76	2
843094	A	Herlev	30	65	5
369360	B	Rigshospitalet	21	68	5
688213	B	Rigshospitalet	29	77	3
197347	A	Herlev	25	65	5
759063	B	Rigshospitalet	16	75	4
121219	B	Herlev	27	68	4
427898	B	Rigshospitalet	15	74	5
674514	A	Rigshospitalet	25	77	1

CHEAT SHEET 5

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] # (de)compress (.tar).gz files  
unzip -[f] [file] # decompress .zip files  
zless -[f] [file] # view .gz file w/o decompression  
gunzip -[f] [file] # uncompress and keep original  
  
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
```



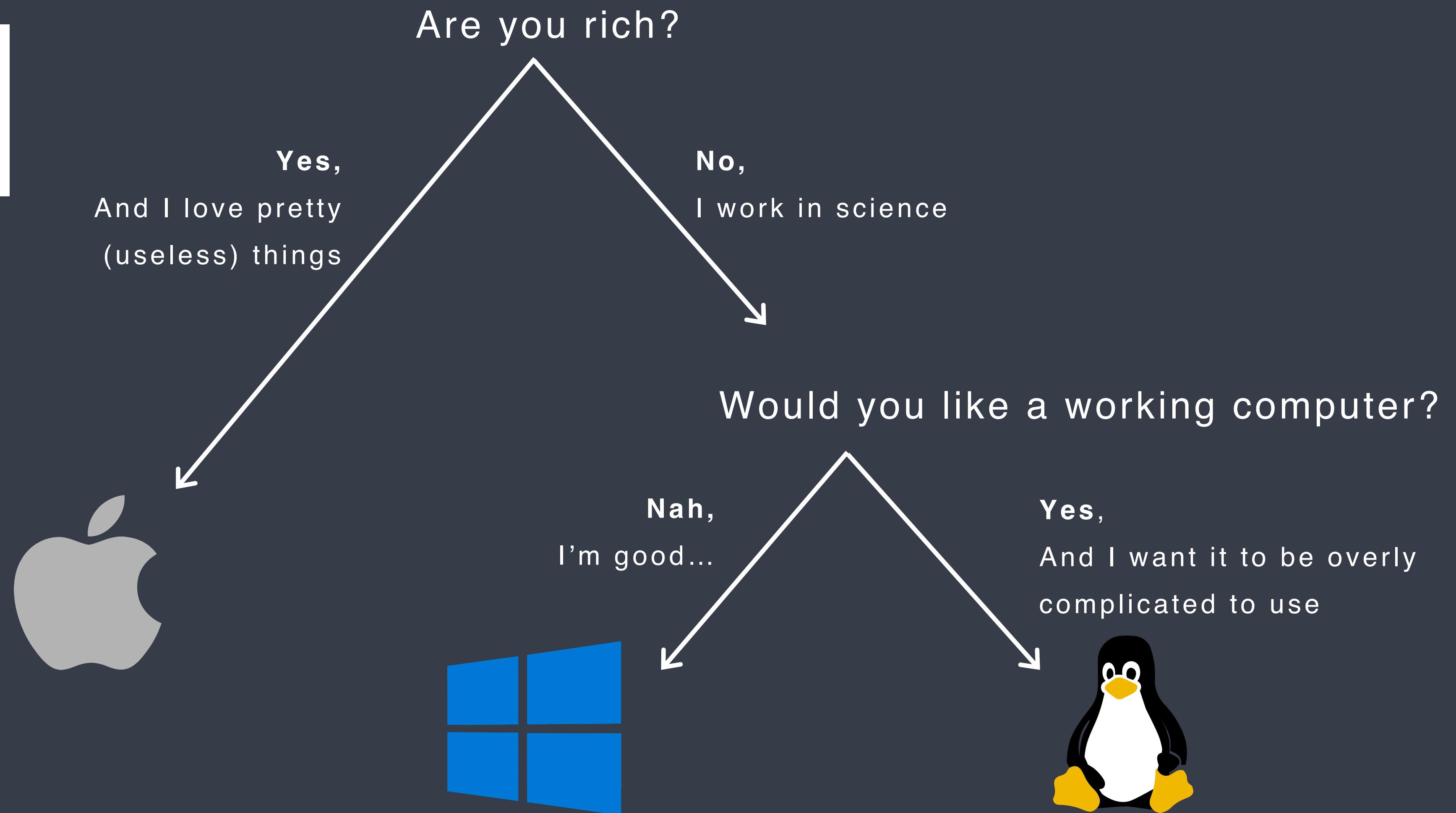
```
Last login: Fri Jun 24 15:05:34 on ttys000
[kgx936@SUN1007442 ~ % echo "Just Bash It"
Just Bash It
kgx936@SUN1007442 ~ % ]
```

“

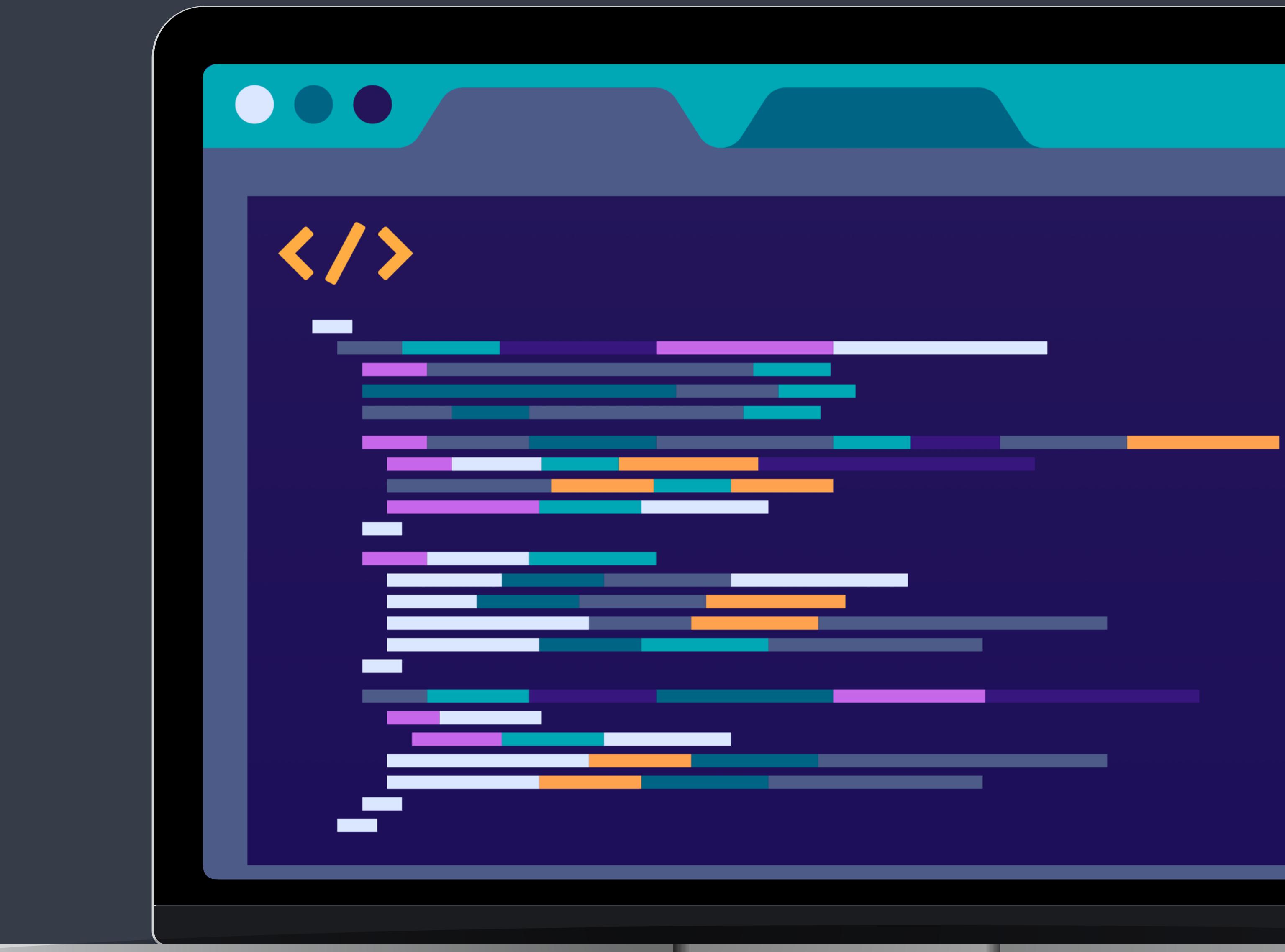
Last exercise for today,
Exercise 5 let's go!

SEE YOU TOMORROW

HOW TO PICK AN OPERATING SYSTEM



6. DATA WRANGLING 2



SORT - GUESS WHAT, IT SORTS!

Sort produces a sorted output.

We need to specify the field separator as the default separator in sort is a space.

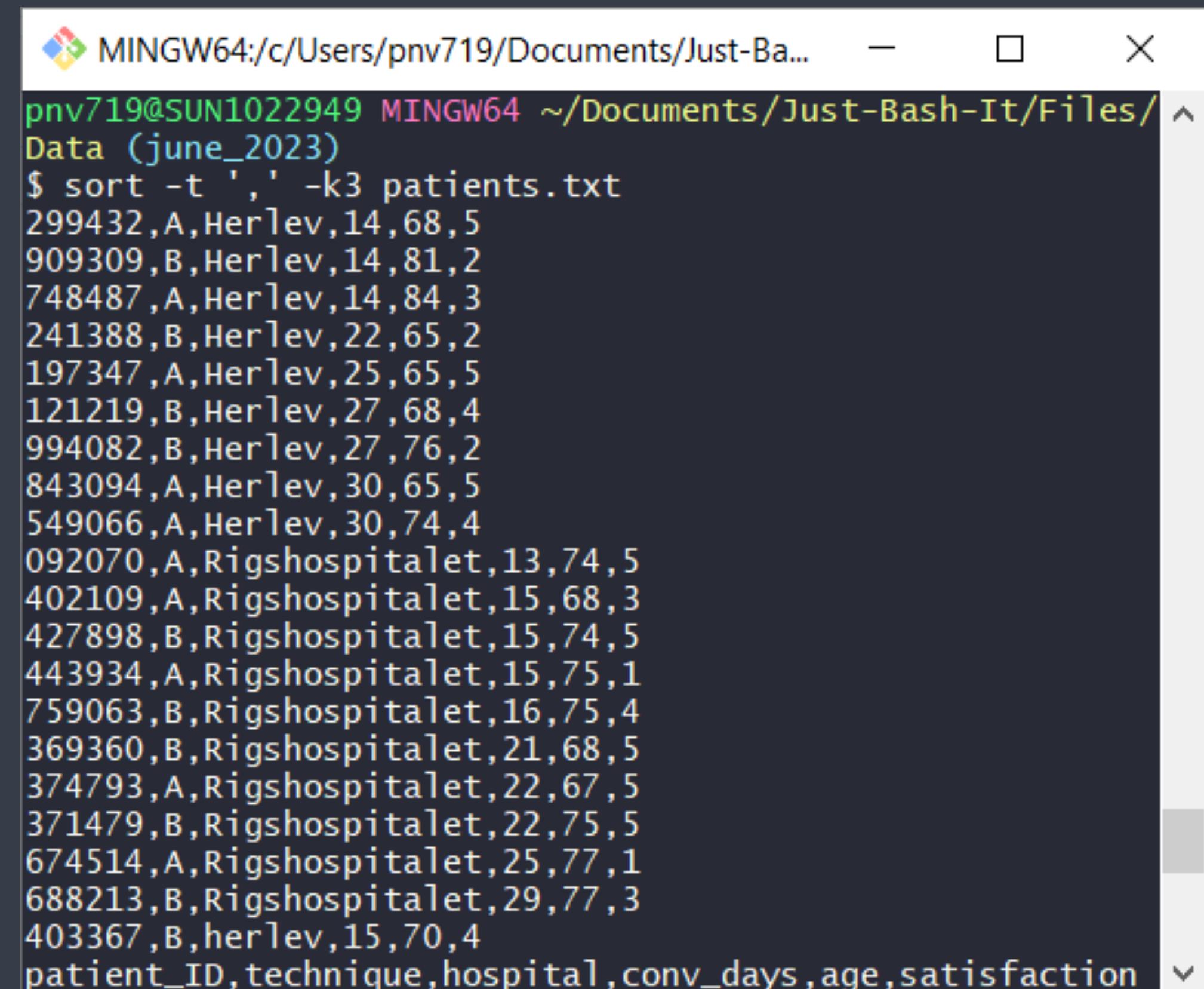
- The column to sort on is given by -k.

```
$ sort -t ',' -k3 patients.txt
```

Sort is alphanumeric, i.e. 10 is smaller than 9 since $1 < 9$.

- To switch to numeric sort, use the -n option:

```
$ sort -t ',' -n -k4 patients.txt
```



The screenshot shows a terminal window titled 'MINGW64:/c/Users/pnv719/Documents/Just-Bash-It/Files/'. The command run is '\$ sort -t ',' -k3 patients.txt'. The output is a list of patient records, each consisting of five fields separated by commas. The records are sorted primarily by the third field ('age') in ascending order, and secondarily by the first field ('patient_ID') in lexicographical order. The last two columns ('age' and 'satisfaction') show numerical values. A scroll bar is visible on the right side of the terminal window.

patient_ID	technique	hospital	conv_days	age	satisfaction
299432	A	Herlev	14	68	5
909309	B	Herlev	14	81	2
748487	A	Herlev	14	84	3
241388	B	Herlev	22	65	2
197347	A	Herlev	25	65	5
121219	B	Herlev	27	68	4
994082	B	Herlev	27	76	2
843094	A	Herlev	30	65	5
549066	A	Herlev	30	74	4
092070	A	Rigshospitalet	13	74	5
402109	A	Rigshospitalet	15	68	3
427898	B	Rigshospitalet	15	74	5
443934	A	Rigshospitalet	15	75	1
759063	B	Rigshospitalet	16	75	4
369360	B	Rigshospitalet	21	68	5
374793	A	Rigshospitalet	22	67	5
371479	B	Rigshospitalet	22	75	5
674514	A	Rigshospitalet	25	77	1
688213	B	Rigshospitalet	29	77	3
403367	B	herlev	15	70	4

GREP – LOOKING FOR SOMETHING

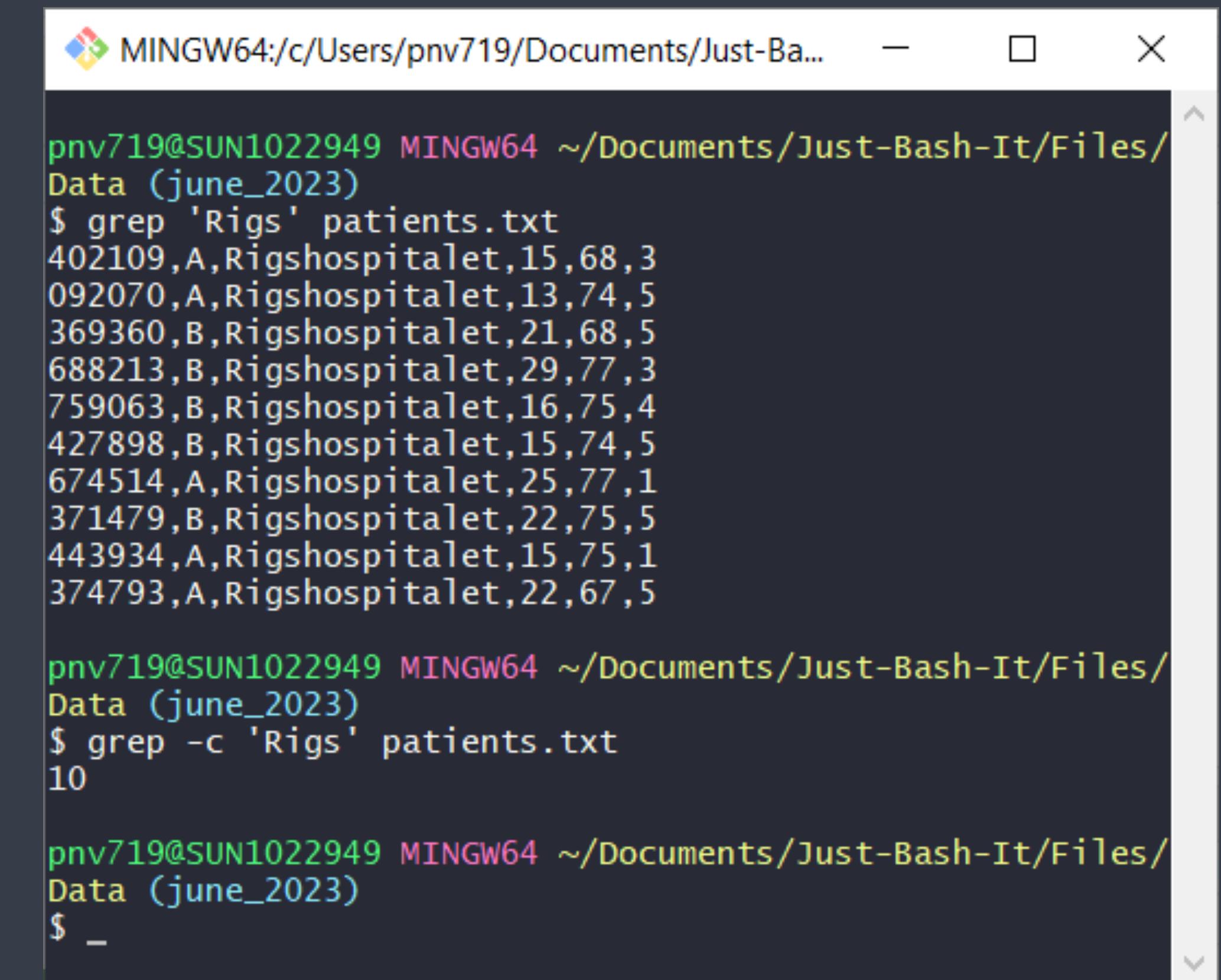
- The **grep** searches files for matches to a pattern and by default returns the entire line:

```
$ grep 'Rigs' patients.txt
```

- Has option for counting the number of matching lines:

```
$ grep -c 'Rigs' patients.txt
```

grep understands **regex** (like **sed**) so you can make exact patterns. You may need the **-P** option to get grep to compile the expression in Perl style



A screenshot of a terminal window titled "MINGW64:/c/Users/pnv719/Documents/Just-Bash-It/Files/Data (june_2023)". The terminal shows two commands being run: "grep 'Rigs' patients.txt" which outputs ten lines of patient data, and "grep -c 'Rigs' patients.txt" which outputs the number 10.

```
pnv719@SUN1022949 MINGW64 ~/Documents/Just-Bash-It/Files/
Data (june_2023)
$ grep 'Rigs' patients.txt
402109,A,Rigshospitalet,15,68,3
092070,A,Rigshospitalet,13,74,5
369360,B,Rigshospitalet,21,68,5
688213,B,Rigshospitalet,29,77,3
759063,B,Rigshospitalet,16,75,4
427898,B,Rigshospitalet,15,74,5
674514,A,Rigshospitalet,25,77,1
371479,B,Rigshospitalet,22,75,5
443934,A,Rigshospitalet,15,75,1
374793,A,Rigshospitalet,22,67,5

pnv719@SUN1022949 MINGW64 ~/Documents/Just-Bash-It/Files/
Data (june_2023)
$ grep -c 'Rigs' patients.txt
10
```

AWK – THE FIX EVERYTHING OR DIE TRYING

awk is a domain specific language made of one-liners that is often used for file manipulation.

It can be used for selecting lines, cutting and pasting files together and even perform arithmetic on file content!

```
$ awk '/,5$/ {print}' patients.txt
```

```
$ awk -F ';' '{print $2,$3}' patients.txt
```

```
$ awk -F ',' '{if ($4 >15) {print}}' patients.txt
```

Why use awk instead a general-purpose programming language?
Because, you can directly invoke it on the command line without needing a script and input/output file handling. It's convenient!

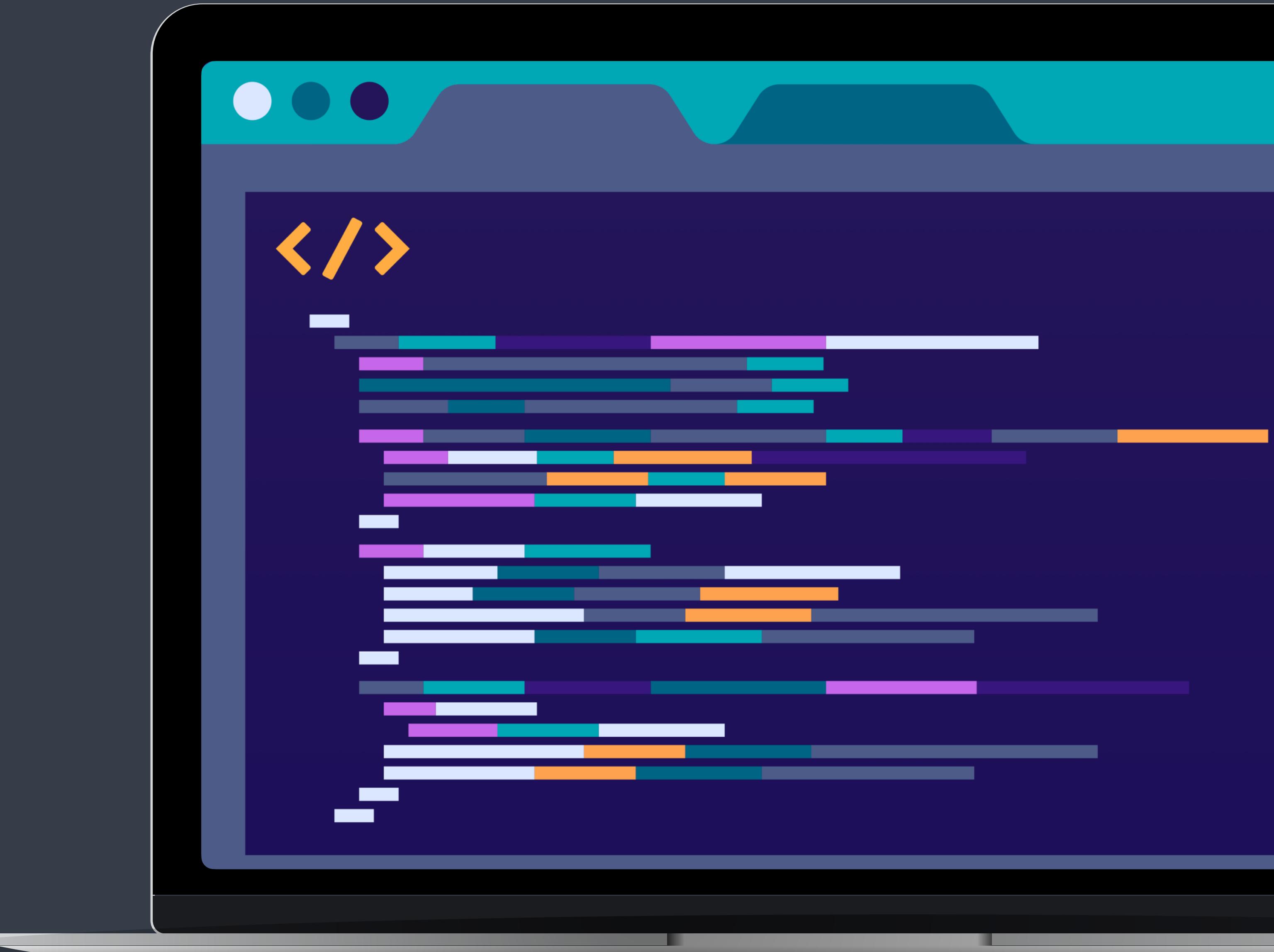


```
Last login: Fri Jun 24 15:05:34 on ttys000
[kgx936@SUN1007442 ~ % echo "Just Bash It"
Just Bash It
kgx936@SUN1007442 ~ % ]
```

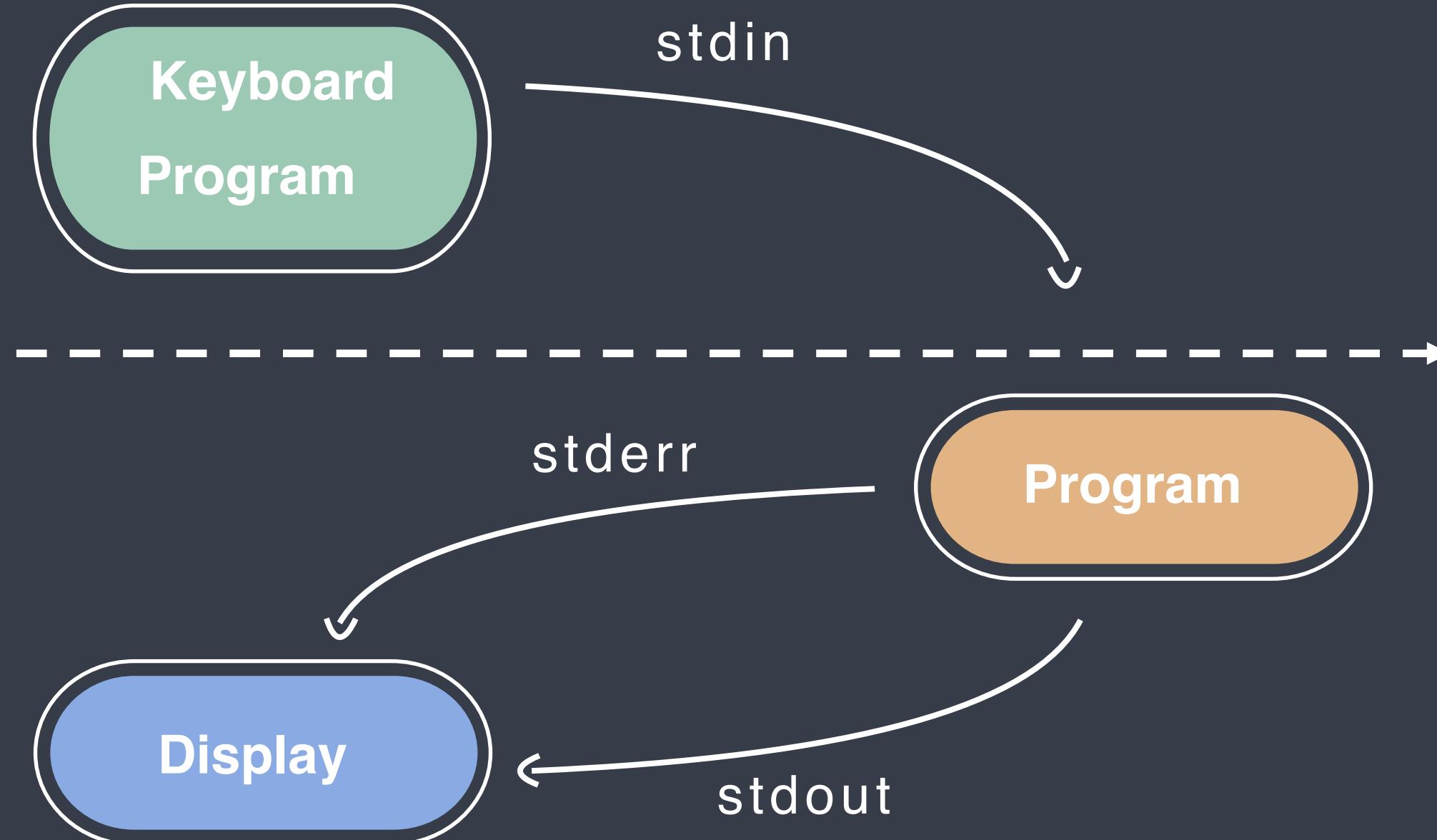
“

First exercise of today,
Exercise 6 let's go!

7. REDIRECTION & PIPES

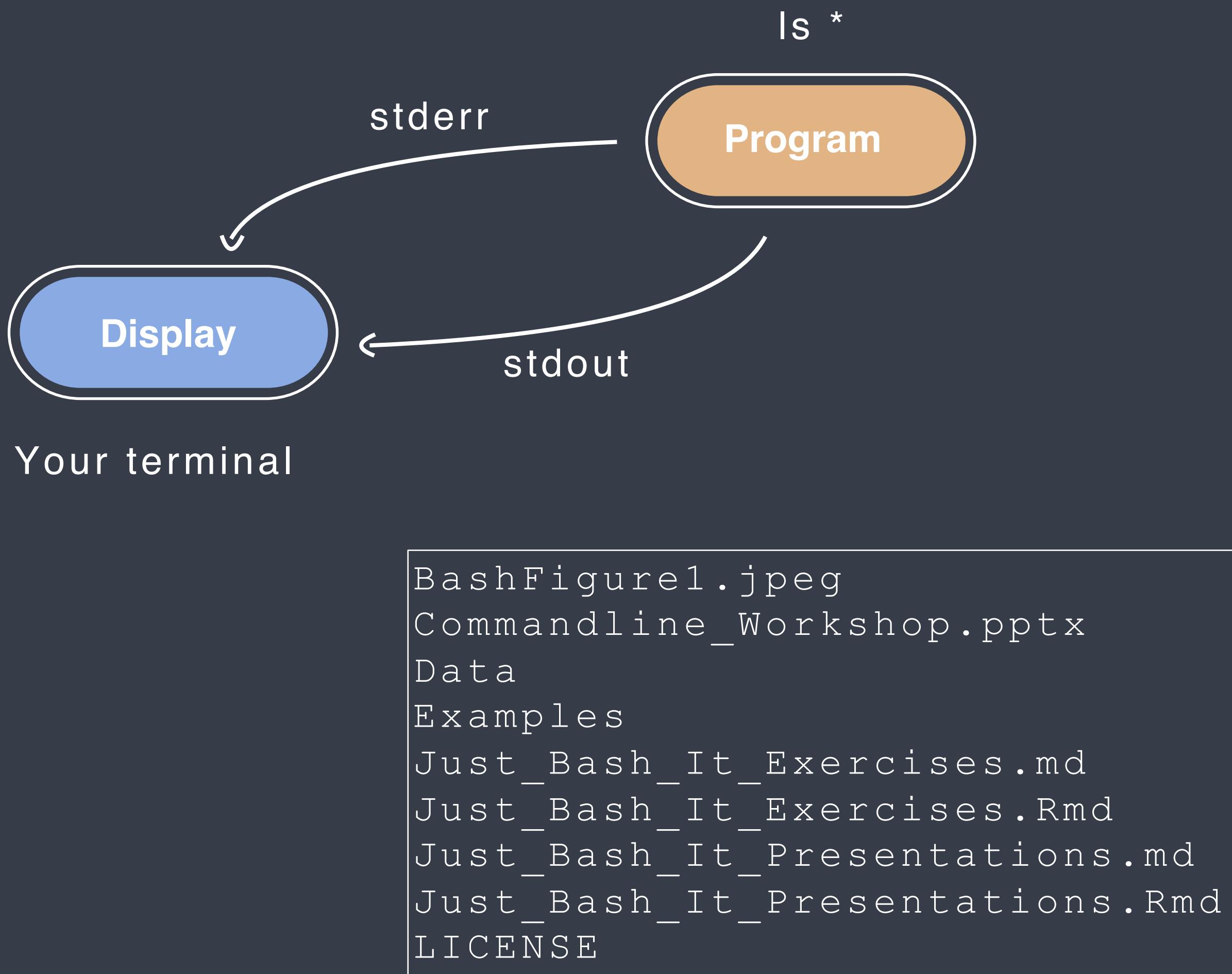


A TALE OF THREE STREAMS



- 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.

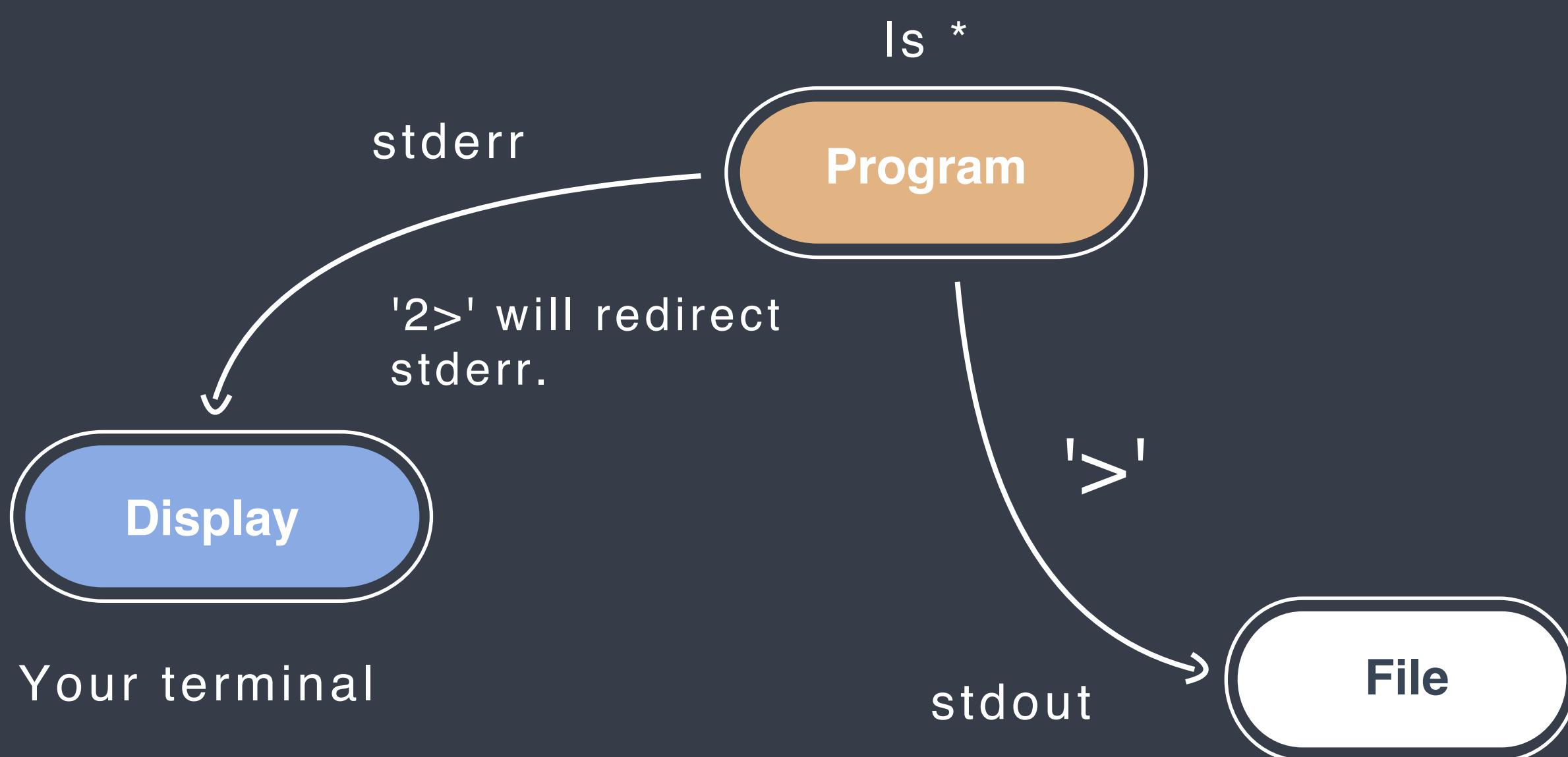
A TALE OF THREE STREAMS



- **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
```

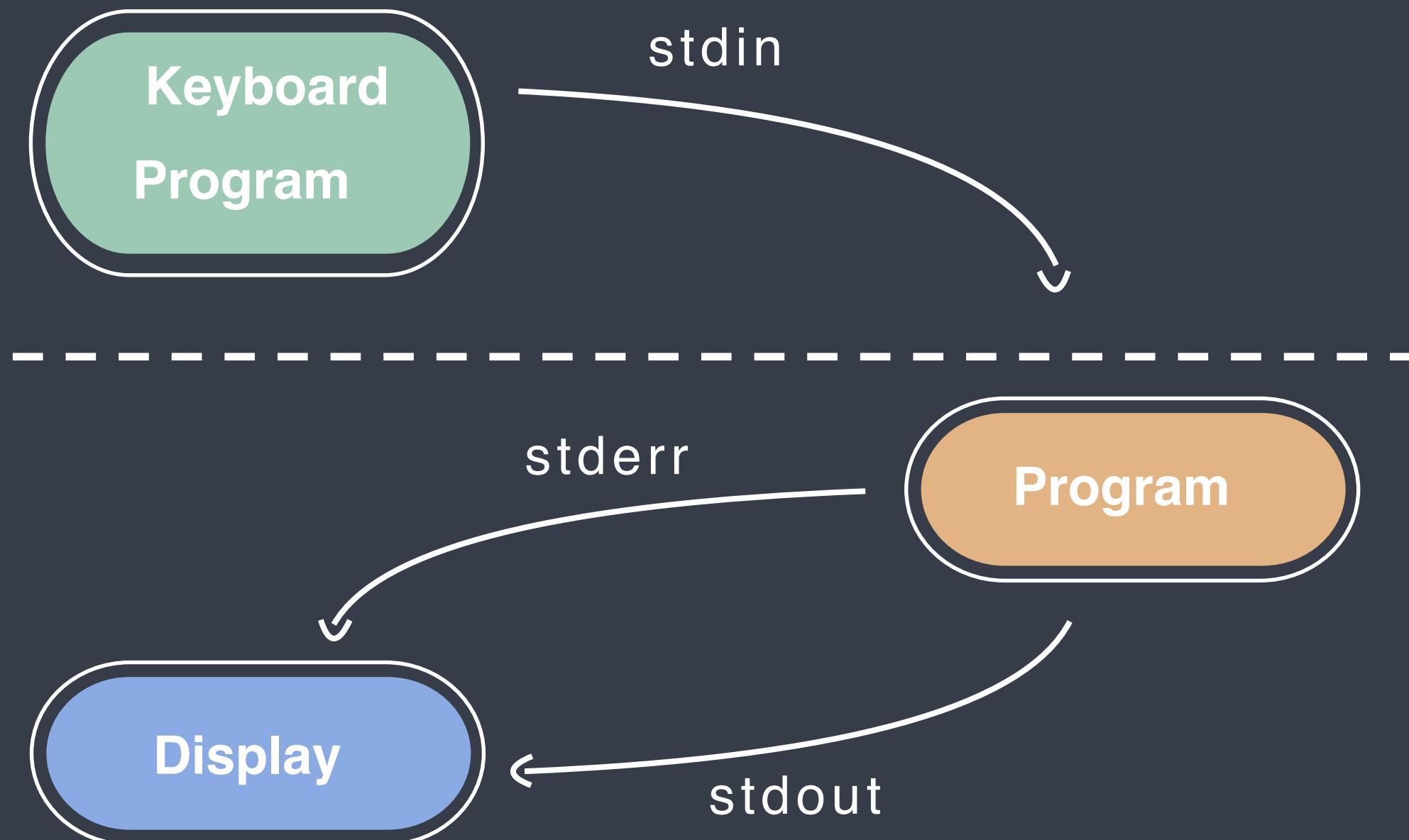
Redirect both into different files:

```
$ 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



File1.txt	10
File2.txt	5
File3.txt	8

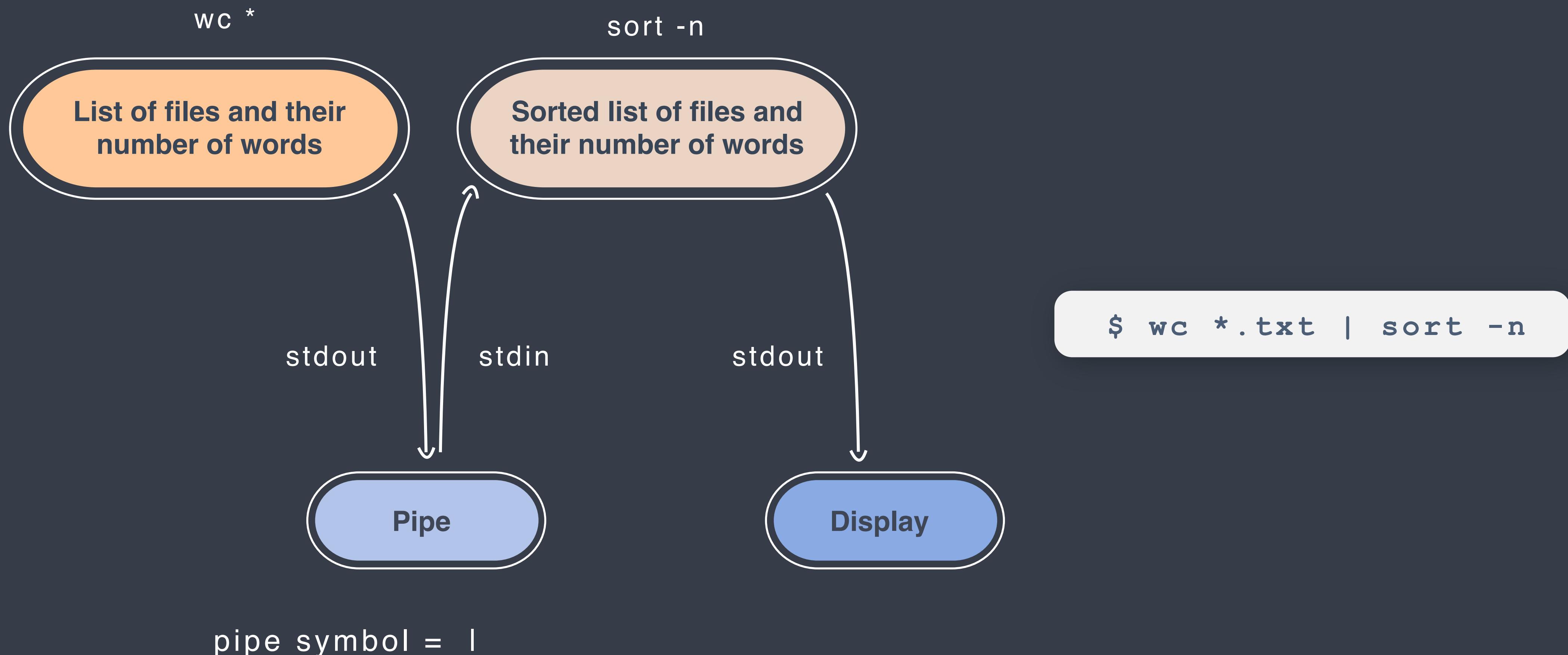
sort

Instead we can use a **pipe** :

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

pipe symbol = |

CHAINING (PIPING) COMMANDS



CHAINING (PIPING) COMMANDS

Some examples:

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

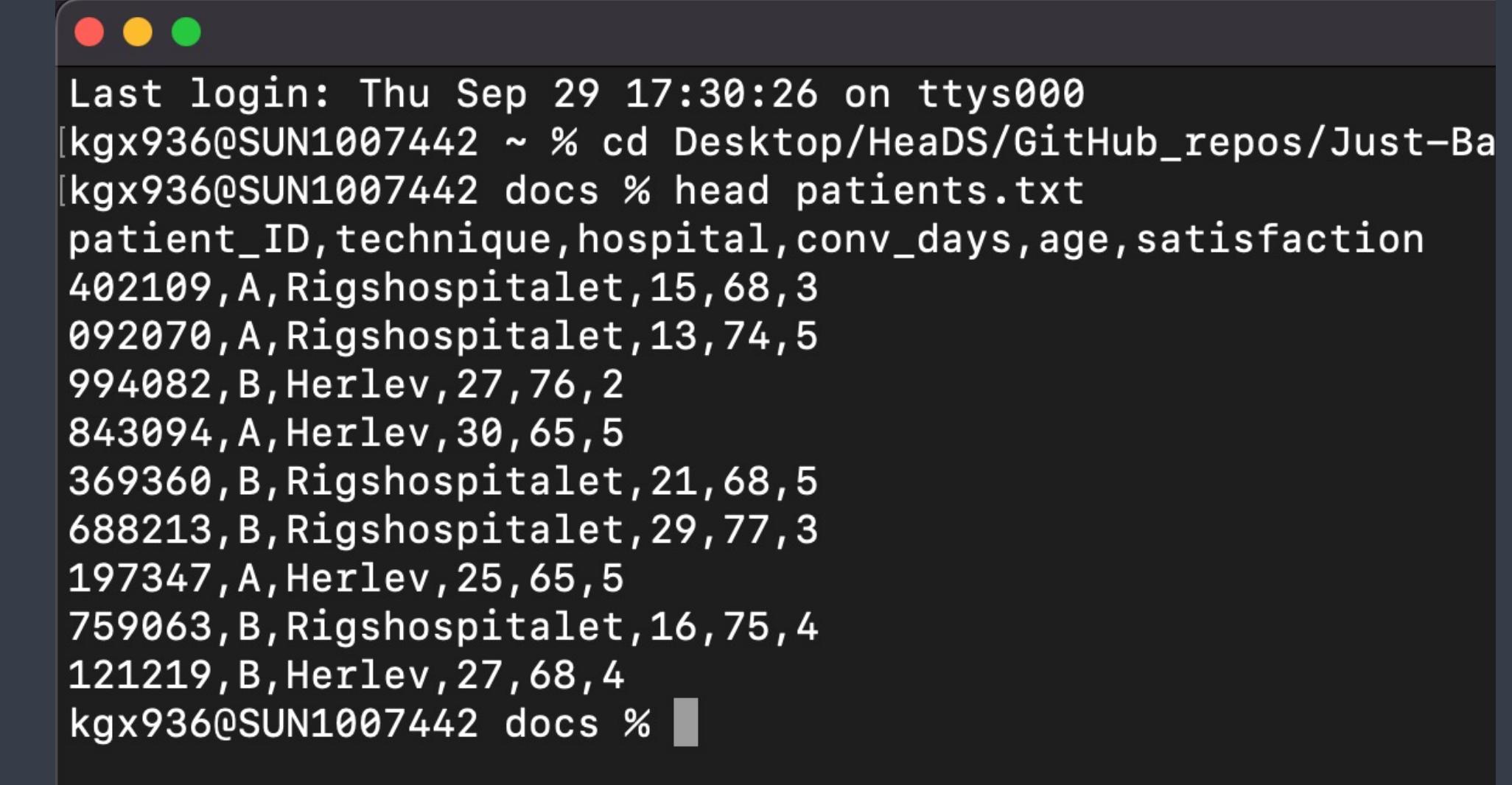
```
$ cut -d ',' -f 2 patients.txt | sort | uniq -c
```

- 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
```



```
Last login: Thu Sep 29 17:30:26 on ttys000
[kgx936@SUN1007442 ~ % cd Desktop/HeaDS/GitHub_repos/Just-Ba
[kgx936@SUN1007442 docs % head patients.txt
patient_ID,technique,hospital,conv_days,age,satisfaction
402109,A,Rigshospitalet,15,68,3
092070,A,Rigshospitalet,13,74,5
994082,B,Herlev,27,76,2
843094,A,Herlev,30,65,5
369360,B,Rigshospitalet,21,68,5
688213,B,Rigshospitalet,29,77,3
197347,A,Herlev,25,65,5
759063,B,Rigshospitalet,16,75,4
121219,B,Herlev,27,68,4
kgx936@SUN1007442 docs %
```

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
```

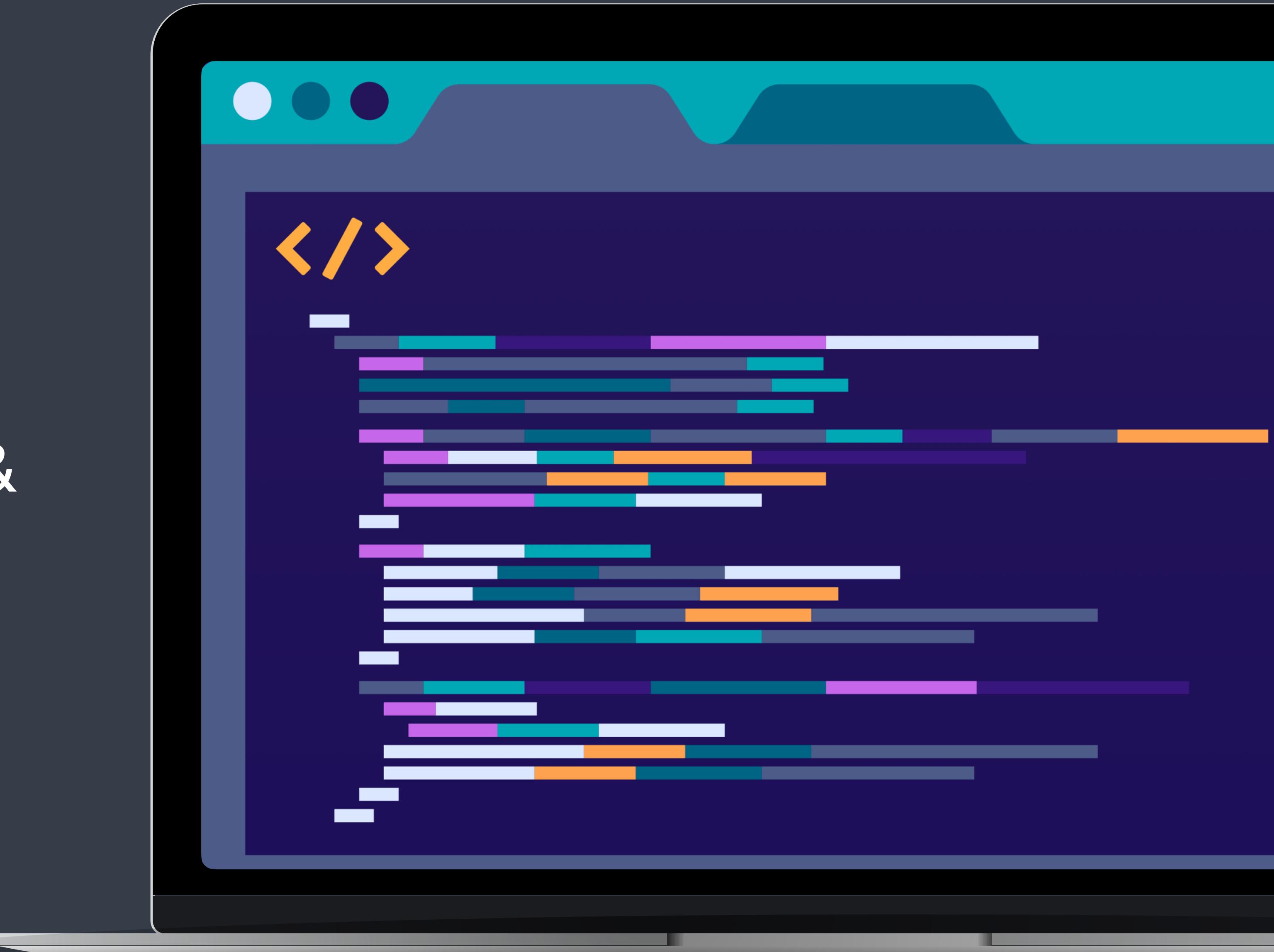


```
Last login: Fri Jun 24 15:05:34 on ttys000
[kgx936@SUN1007442 ~ % echo "Just Bash It"
Just Bash It
kgx936@SUN1007442 ~ % ]
```

“

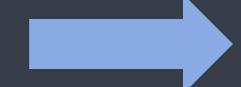
Now we are getting ‘fancy’
and chaining some
commands, **Exercise 7!**

8. SHELL SCRIPTS & LOOPS



WRITING SCRIPTS

```
(base) henrike@henrike-ThinkPad-X390:~/Documents/Heads_center_management/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_management/courses/Just-Bash-It/Examples/docs$ head -n 1 patients.dat > herlev.dat
(base) henrike@henrike-ThinkPad-X390:~/Documents/Heads_center_management/courses/Just-Bash-It/Examples/docs$ grep 'Herlev' patients.dat | sort -n -k 5 -t ',' >> herlev.dat
```



```
my_script.sh (~/Documents/Heads_center_management/courses)
Open + ~/Documents/Heads_center_management/courses/Just-... Save
1 cut -f 3 -d ',' patients.dat | sort | uniq -c
2 head -n 1 patients.dat > herlev.dat
3 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_management/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_management/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 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.
- Use **-x** when you execute the script to show the commands and progression

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

sort -n $1
```

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

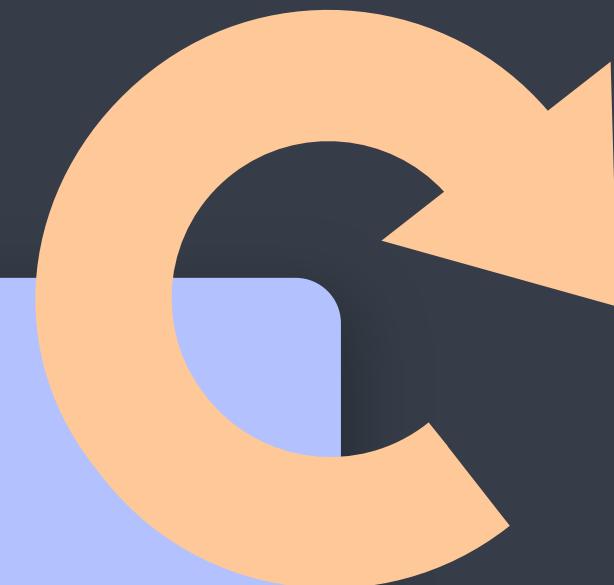
```
$ bash sort_this.sh my_textfile.txt
```

```
# Comment line script 2
# Usage: cut_col.sh [file_name]
[column_number]

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

```
$ bash -x cut_col.sh patients.txt 3
```

LOOP IN BASH



```
#example of a for loop in bash
```

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

v

```
#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:

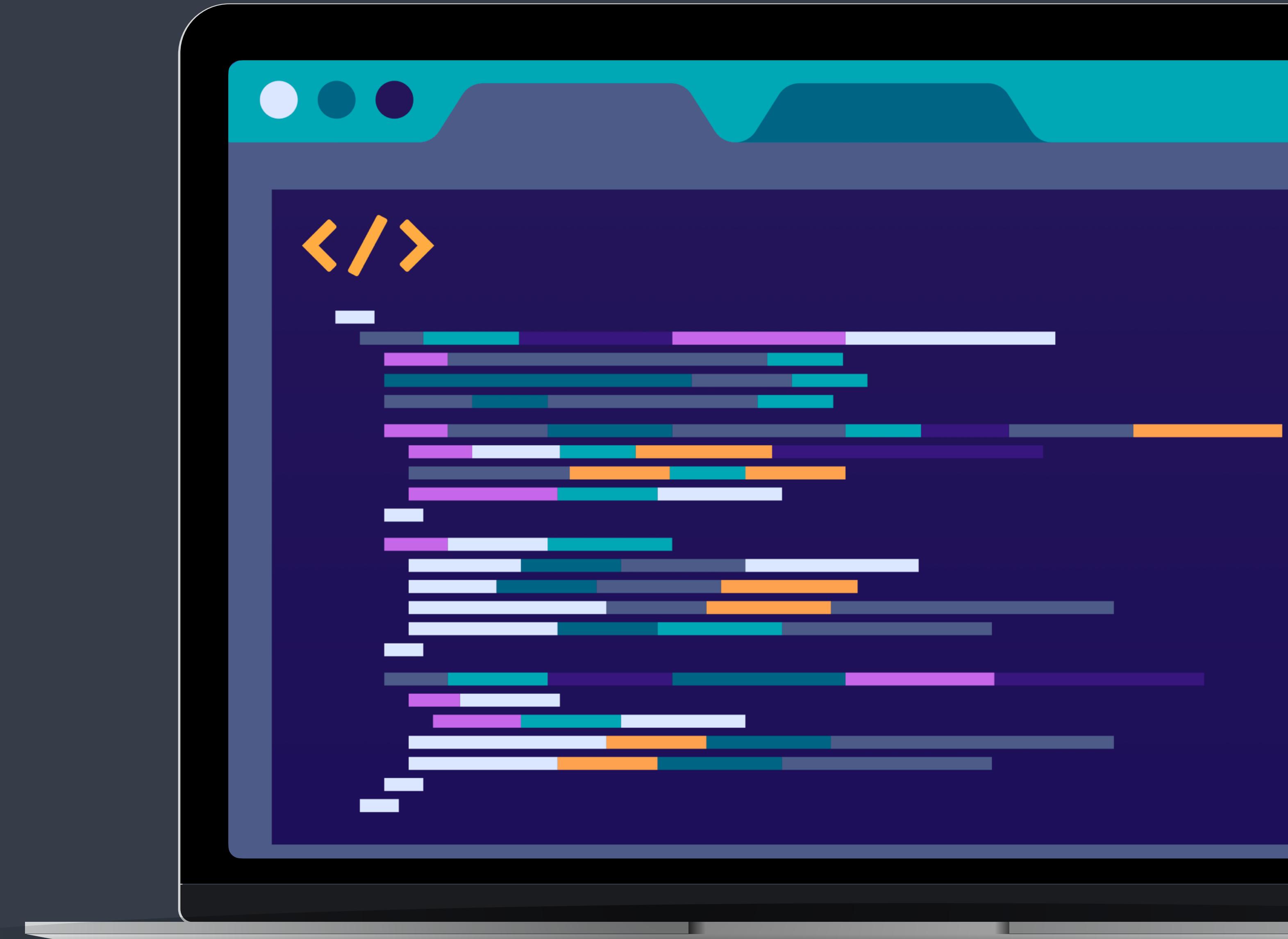
```
$ python align_reads.py
```

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

```
$ Rscript deSEQ_cancer_Data.R
```

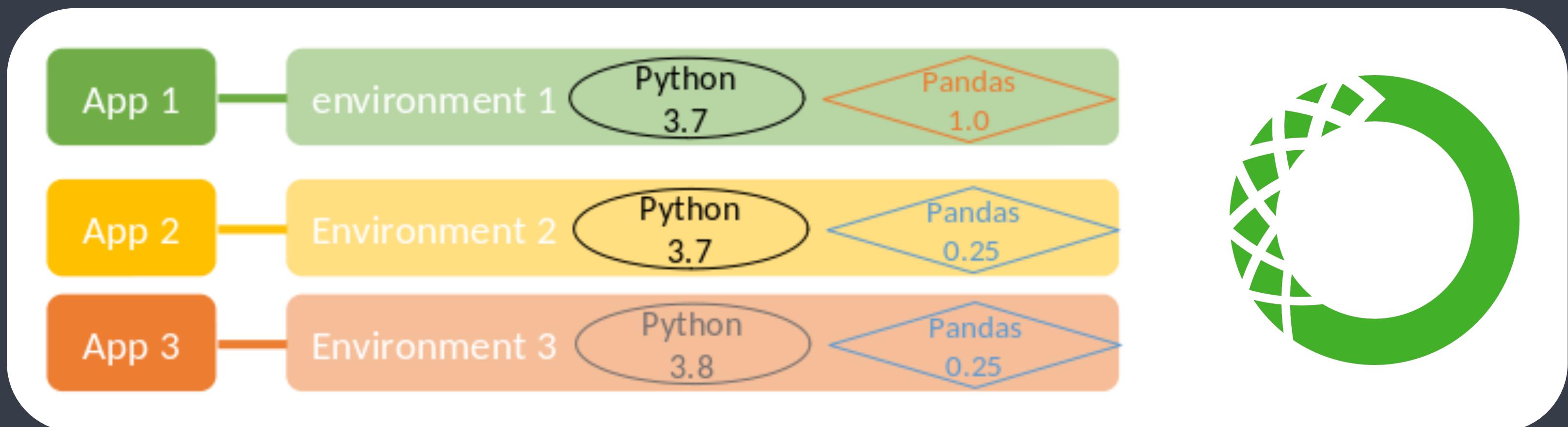
- You need to have R and python installed on your computer in order to be able to call them on the command line.

9. 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 OS X, 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

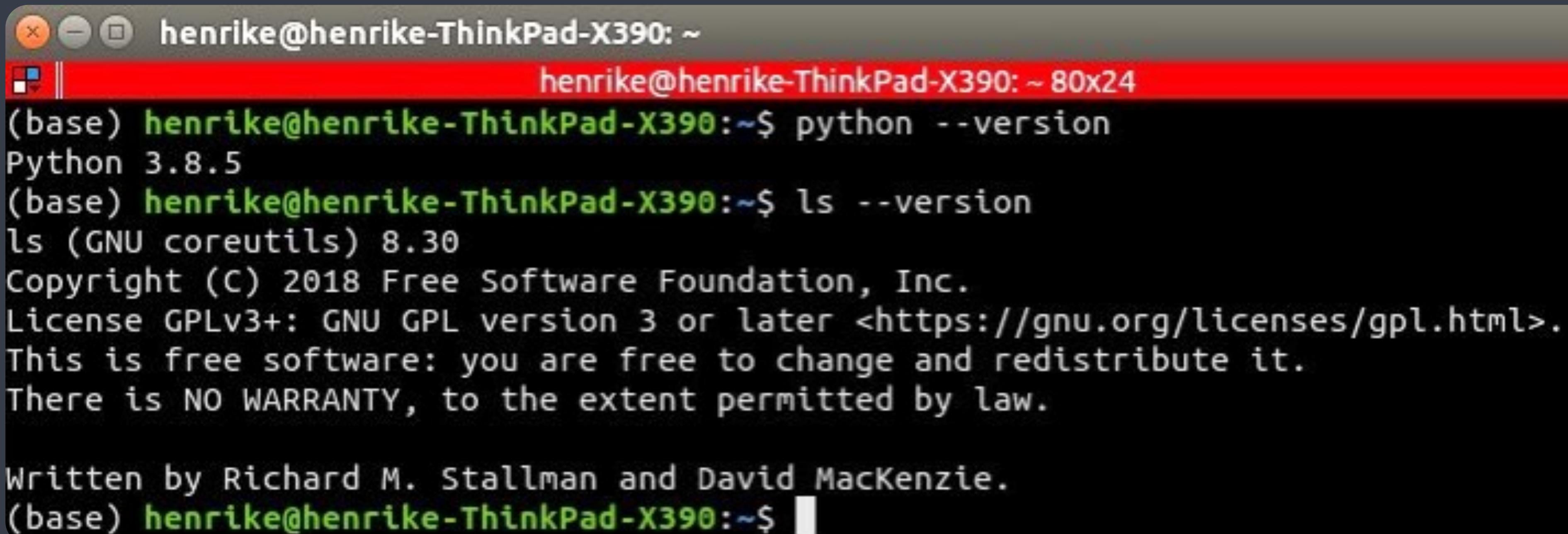
CONS

Packages, NOT environments

SOFTWARE VERSION

You can generally check the version of installed software with:

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



The screenshot shows a terminal window with a dark background and light-colored text. The window title bar says "henrike@henrike-ThinkPad-X390: ~". The terminal prompt "(base)" appears twice. The first prompt is followed by the command "python --version" which outputs "Python 3.8.5". The second prompt is followed by the command "ls --version" which outputs the GNU coreutils license information, including the copyright notice from 2018, the GPL license, and the statement that it is free software with no warranty. The bottom of the terminal shows the text "Written by Richard M. Stallman and David MacKenzie." and the prompt "(base)" again.

```
henrike@henrike-ThinkPad-X390: ~
henrike@henrike-ThinkPad-X390: ~ 80x24
(base) henrike@henrike-ThinkPad-X390:~$ python --version
Python 3.8.5
(base) henrike@henrike-ThinkPad-X390:~$ ls --version
ls (GNU coreutils) 8.30
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <https://gnu.org/licenses/gpl.html>.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Written by Richard M. Stallman and David MacKenzie.
(base) henrike@henrike-ThinkPad-X390:~$
```

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]
```

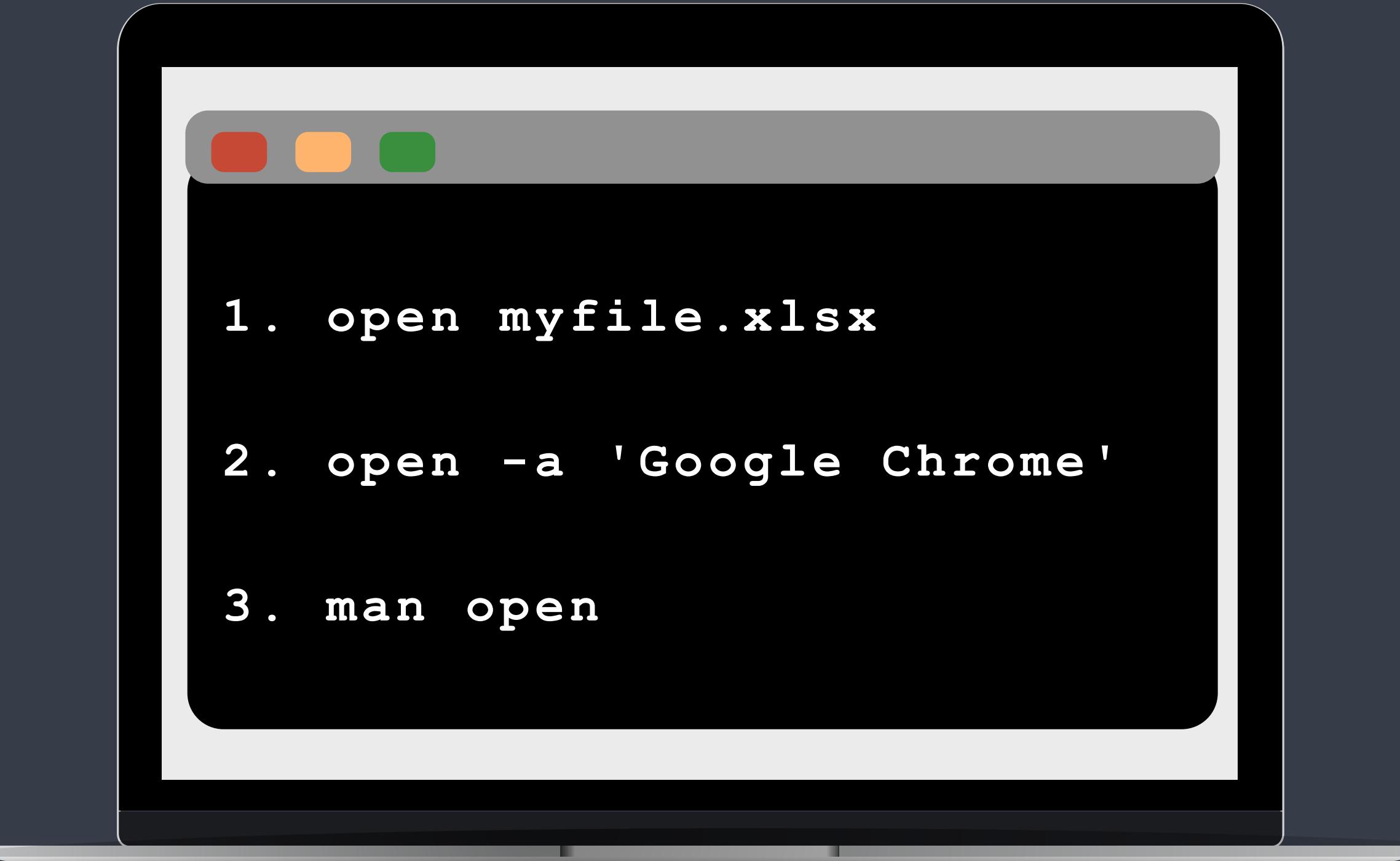


- With **Windows App store**:

*Update via
Windows App store*



OPENING APPS & RUNNING SOFTWARE



1. Some (not all) **files** can be **opened** without specifying an app.
2. Many **apps** can be **launched** with `open -a`.
3. Some **software** are designed to run on both the command line and in a GUI, others are not.

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

```
$ top (htop)  
$ ps aux (processes for all users)  
  
$ ps aux | grep 'Google Chrome'  
$ kill [pid]
```

CHEAT SHEET 3

```
top # display running processes in 'real time'  
ps (aux) [file] # snapshot of processes  
open -a [application] # open an application  
kill [pid] # kill/stop a process using pid  
  
bash [file.sh] or sh [file.sh] # Execute/run bash script  
  
[software] --version # check software version  
brew install [software] # OS X install software on Mac  
apt-get[software] # OS X install software on Ubuntu/Linux  
  
conda # software and environment management system for Windows, Ubuntu & OS X
```

Applications & Scripts



```
Last login: Fri Jun 24 15:05:34 on ttys000
[kgx936@SUN1007442 ~ % echo "Just Bash It"
Just Bash It
kgx936@SUN1007442 ~ % ]
```

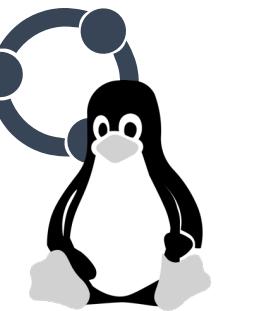
“

Last exercise, let's install
and run a software using the
command line, **Exercise 8!**

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.

UBUNTU - BASH SHELL



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

OS X - 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 bash scripts.

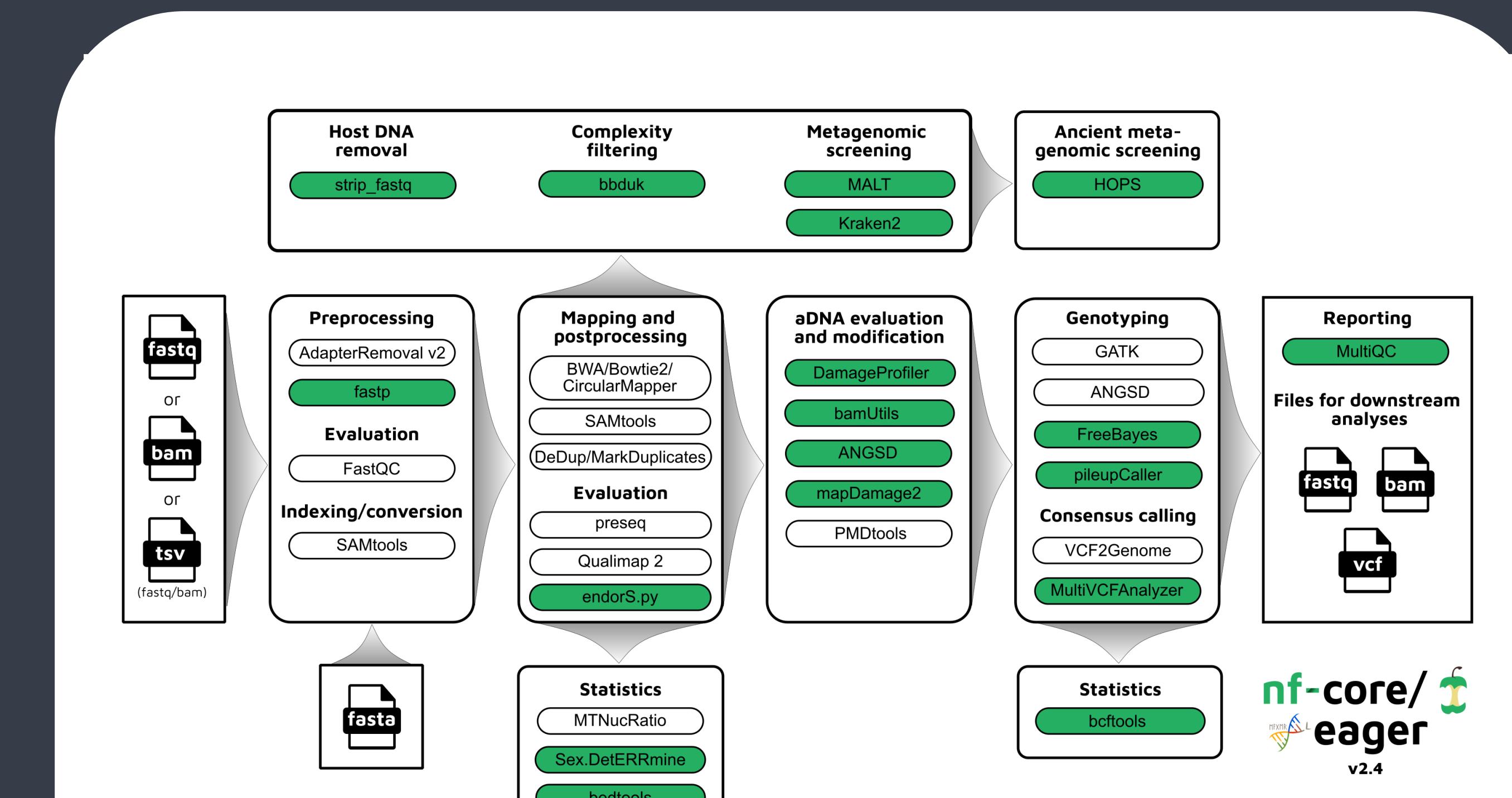
Workflow languages allow you to:

- Control the flow of your pipeline
- Restart at various defined points
- Ensure previous commands have executed without error
- Logging and reporting jobs



snakemake

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

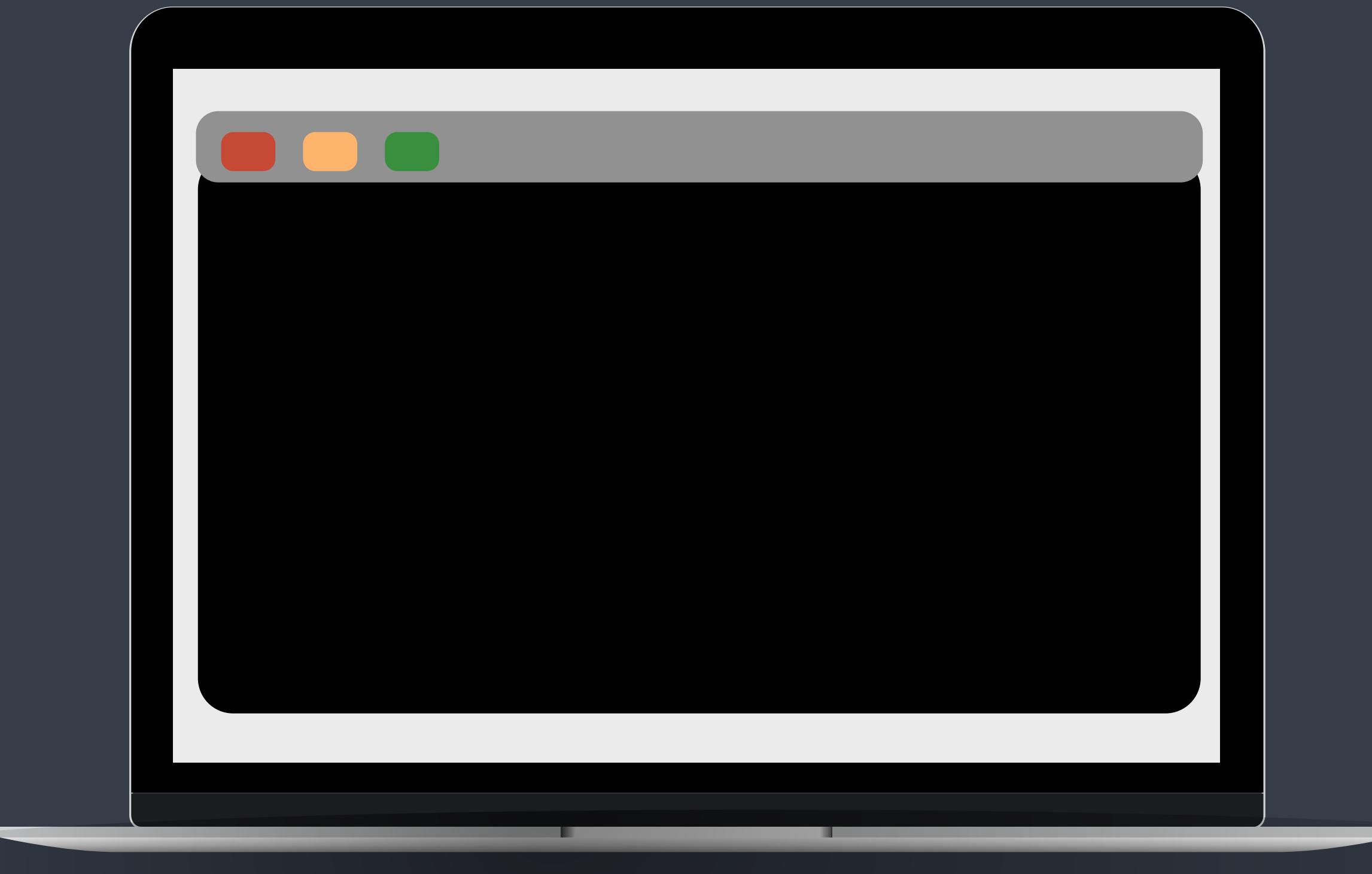


nextflow

<https://www.nextflow.io/>

nf-core/eager
v2.4

SUDO THE ULTIMATE POWER MOVE - DON'T USE IT!



THANK YOU FOR TODAY

