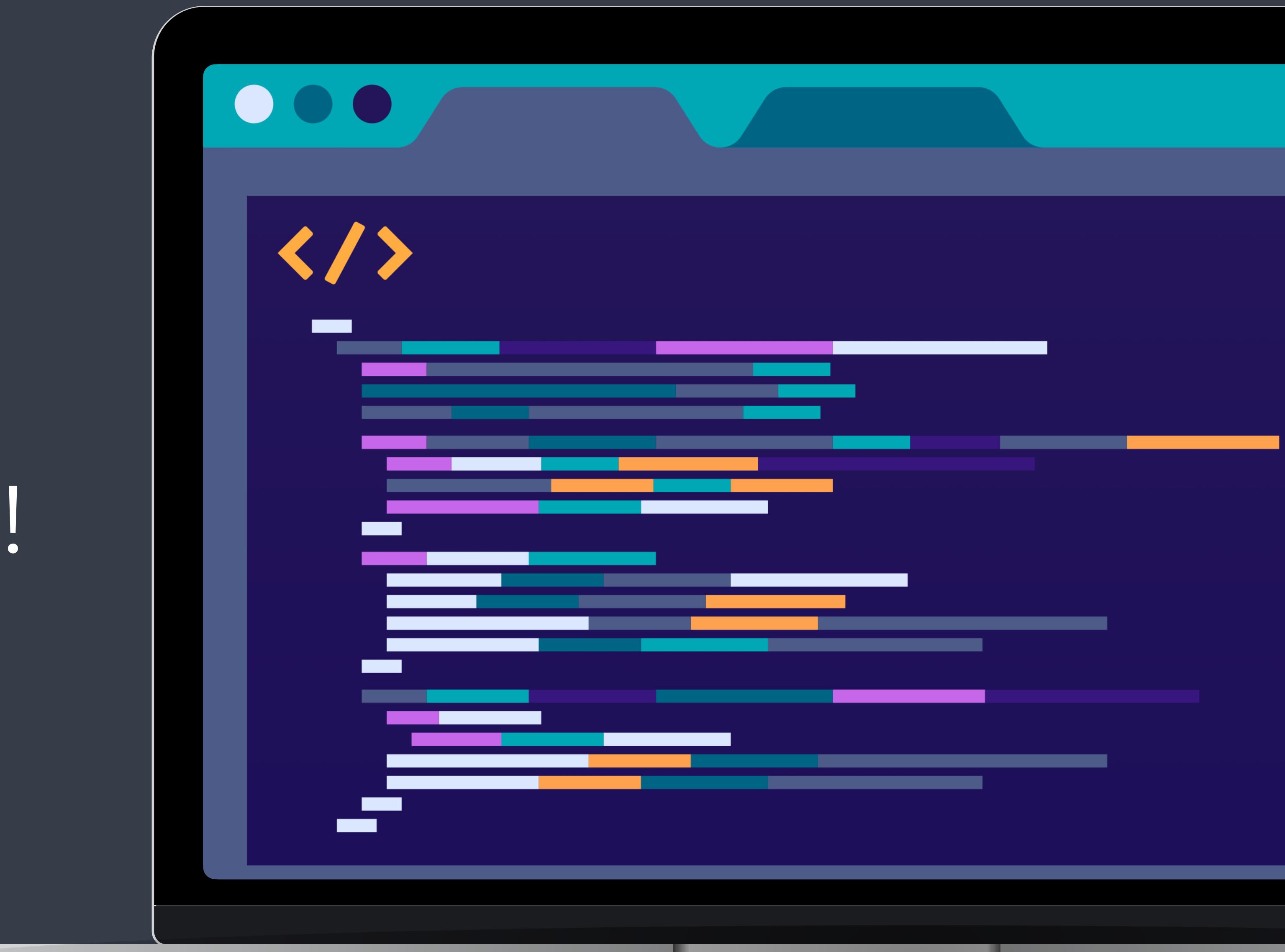
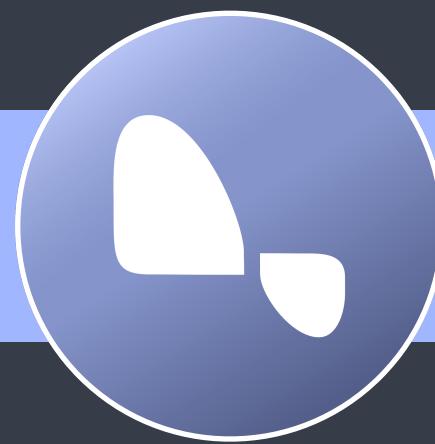


JUST BASH IT!





CENTER FOR HEALTH DATA SCIENCE (HEADS)

- Data Lab Supports Researchers at the Faculty
- Consultation & Commission:
 - Data science and bioinformatics analyses
- Teaching: Courses & Workshops, Seminars.



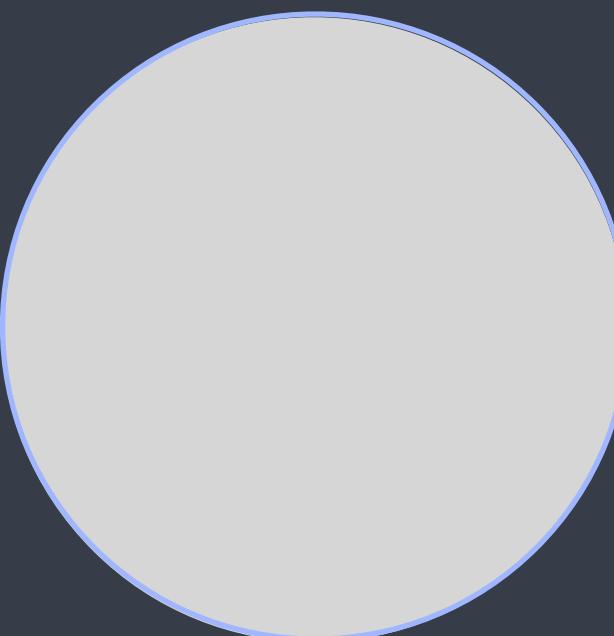
Henrike Zschach



Thilde Terkelsen

COPENHAGEN UNIVERSITY LIBRARY (KUB)

- XXXXXX XXXXXX
- XXXXXX XXXXXX
- XXXXXX XXXXXX



XXXXXX XXXXXX



XXXXXX XXXXXX

JUST BASH IT!

PROGRAM

XX:XX-XX:XX - INTRODUCTION TO COMMANDLINE

XX:XX-XX:XX - NAVIGATING FILES AND DIRECTORIES

XX:XX-XX:XX - PROJECT ORGANIZATION & BACKUP

XX:XX-XX:XX - WORKING WITH FILES

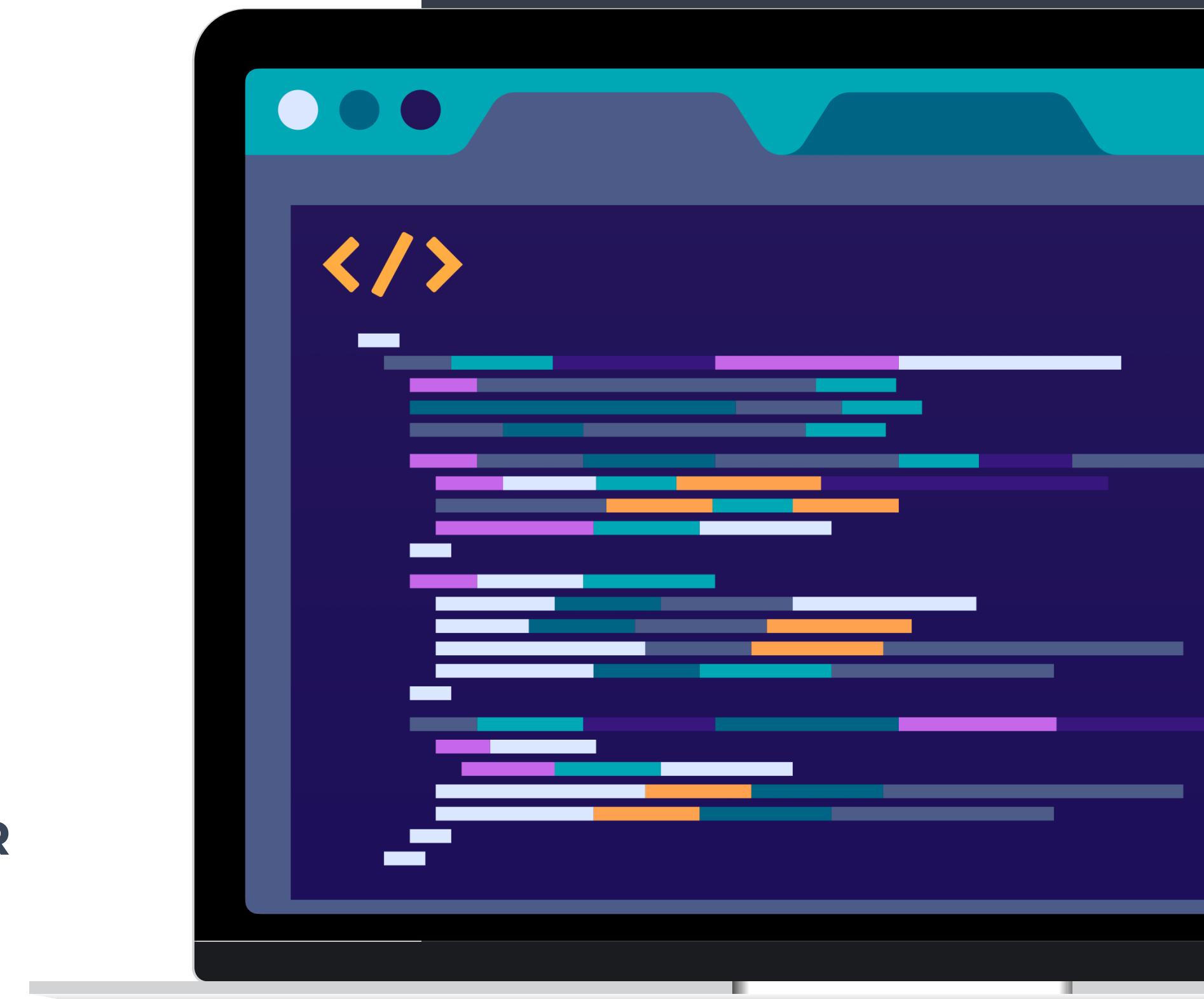
XX:XX-XX:XX - MORE BASH COMMANDS

XX:XX-XX:XX - REDIRECTION & PIPES

XX:XX-XX:XX - SHELL SCRIPTS & LOOPS

XX:XX-XX:XX - SOFTWARE INSTALLATION UPKEEP & MORE

XX:XX-XX:XX - WORKFLOW LANGUAGE & COMPUTE POWER



COURSE MATERIALS:

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



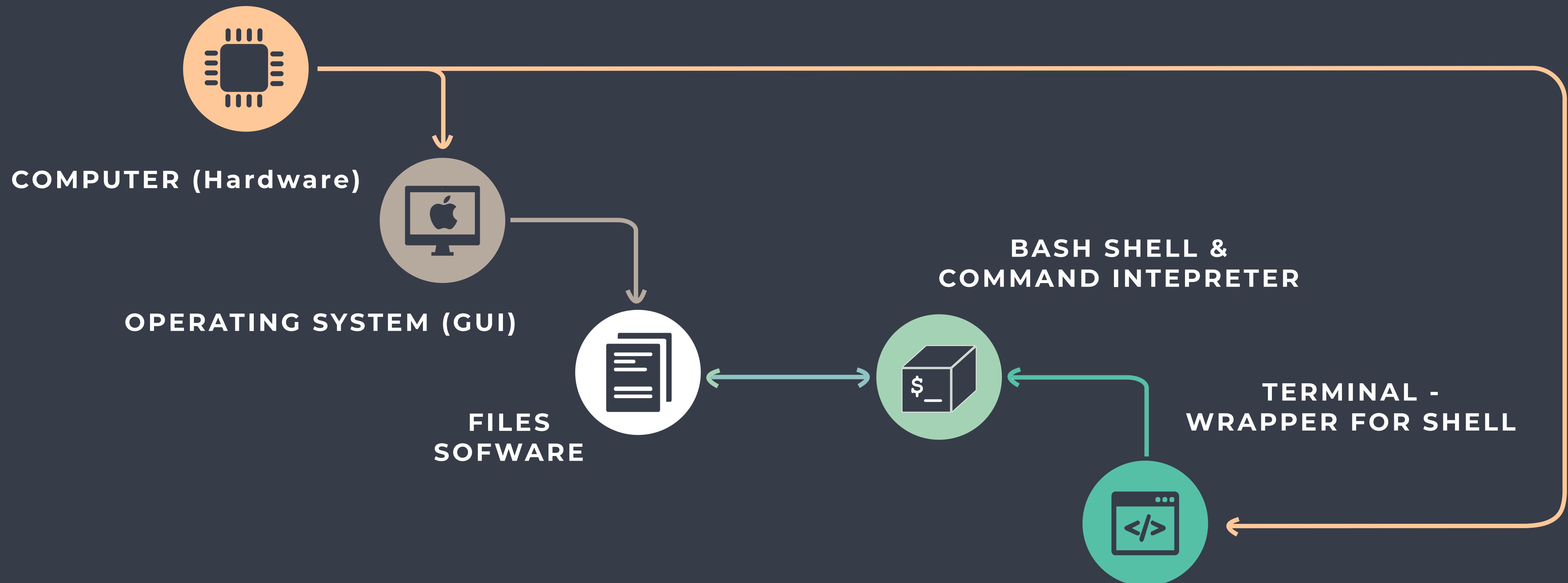
```
Last login: Fri Jun 24 15:05:34 on ttys000  
[kgx936@SUN1007442 ~ % echo "Just Bash It"  
Just Bash It  
kgx936@SUN1007442 ~ % █
```

“

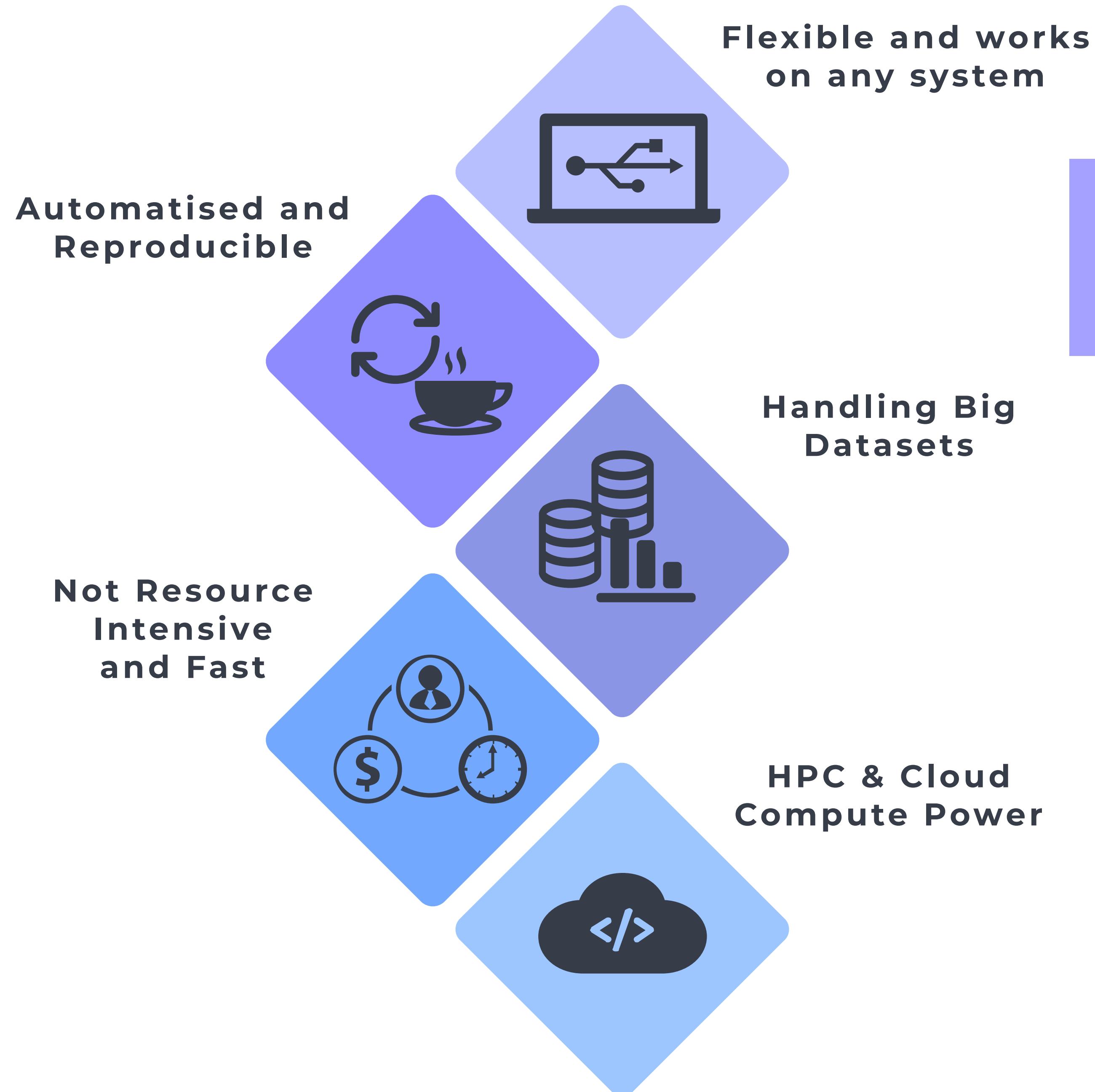
“What is a shell, a terminal, a command-line, and what is bash?”

“How do these concepts connected to my computer?”

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:

- Do a lot with minimum effort on any system.
- Your analysis will be reproducible, automated and parallelized.
- Fast processes, less computationally intensive.
- 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.



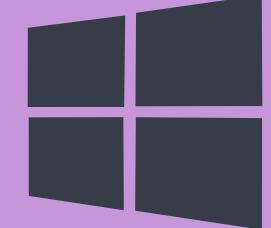
OS X or UBUNTU

In this course Windows users will be working on **MobaXterm**.

You should have **installed this shell** via. instructions in the introduction email.

YOU NEED TO INSTALL A BASH SHELL/TERMINAL:

- **MobaXterm:** <https://mobaxterm.mobatek.net/download.html>
- **Cmder:** <https://cmder.net/>
- **GOW (Cygwin):** <https://github.com/bmatzelle/gow>
- **PuTTY:** <https://www.putty.org/>



WINDOWS

WHAT WILL I LEARN TODAY?



The BASICS OF BASH

- The terminal
- 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

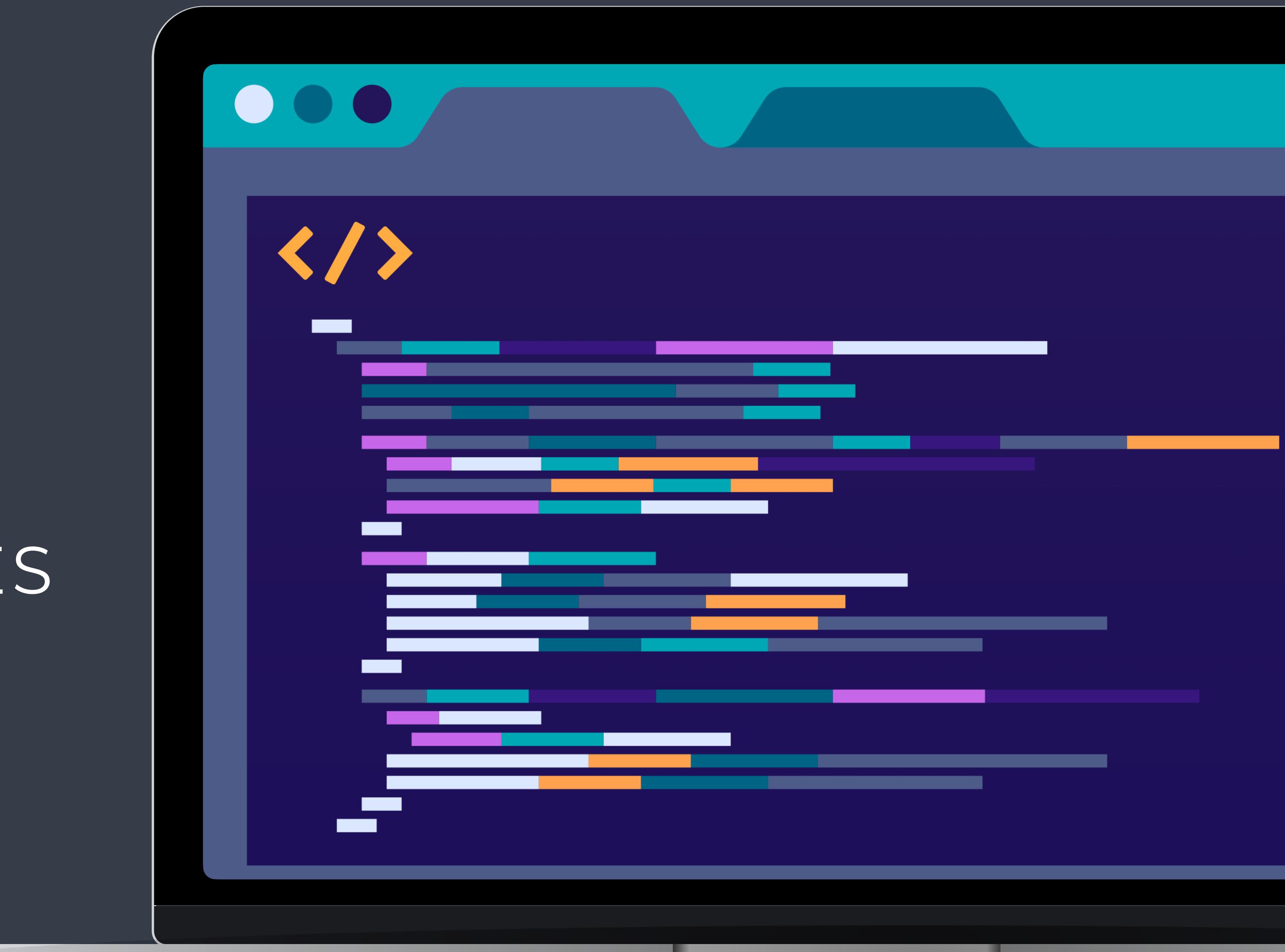
4

STORAGE, BACKUP & MORE...

- Manage Software
- Workflows
- Backups
- Computing Power

5

1. NAVIGATING FILES & DIRECTORIES



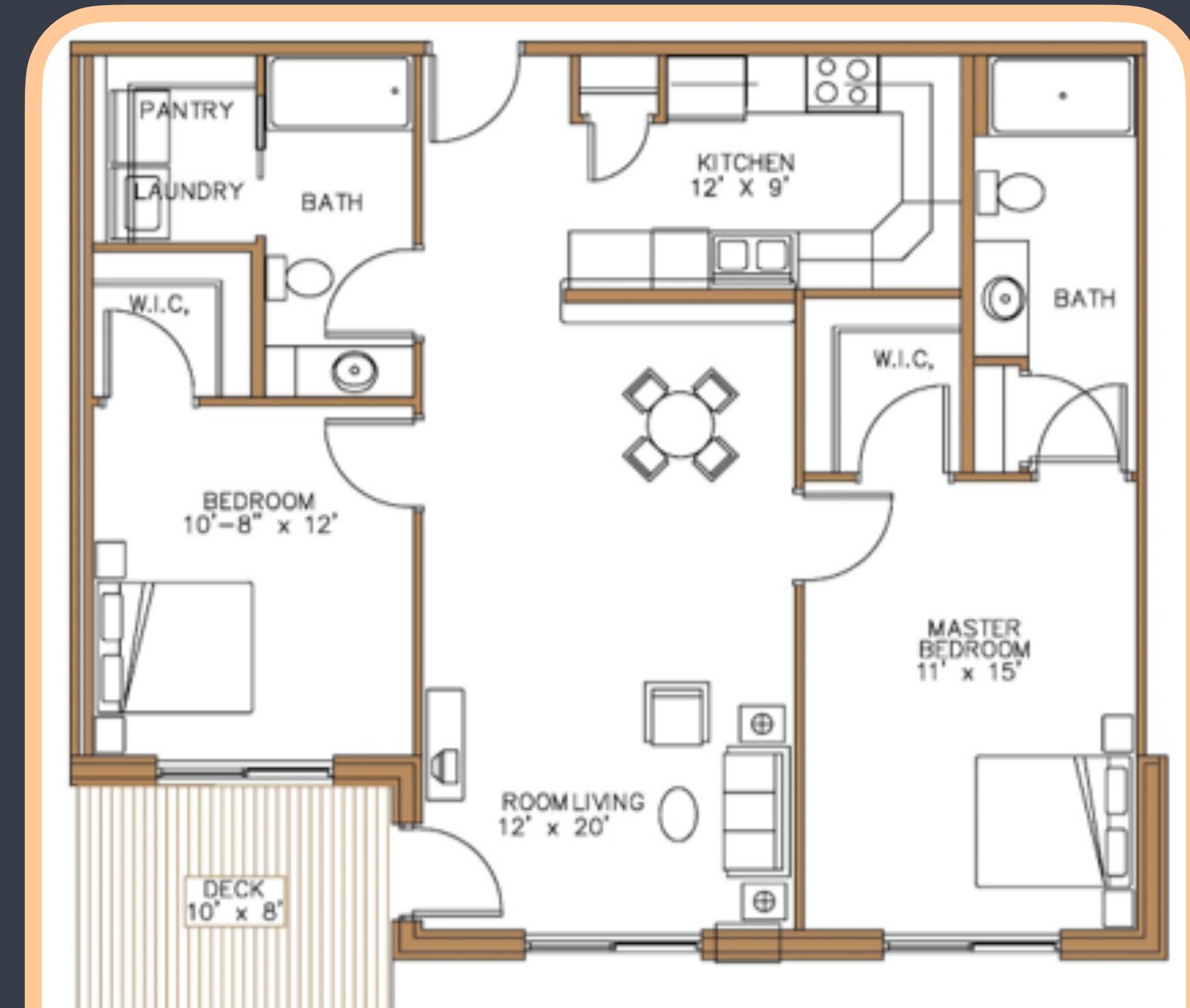
NAVIGATING FILES AND DIRECTORIES

- **Where am I on my computer? (And what does that mean?)**
- **How can I move around on my computer?**
- **What are files and directories?**
- **How can I see what files and directories I have?**
- **How can I specify the location of a file or directory on my computer?**

THE DIRECTORY TREE

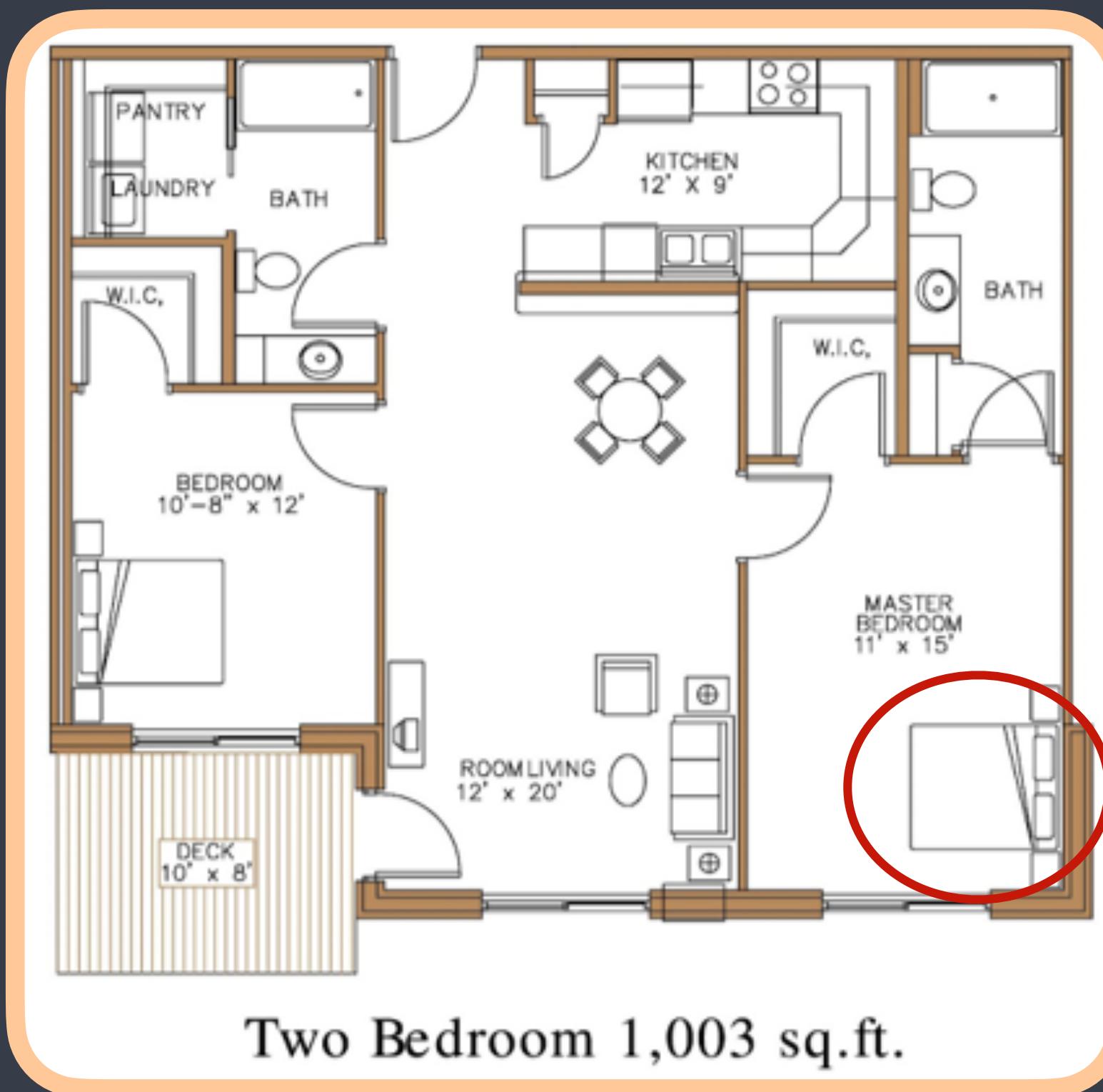
Your computer is **like a house**:

- 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.

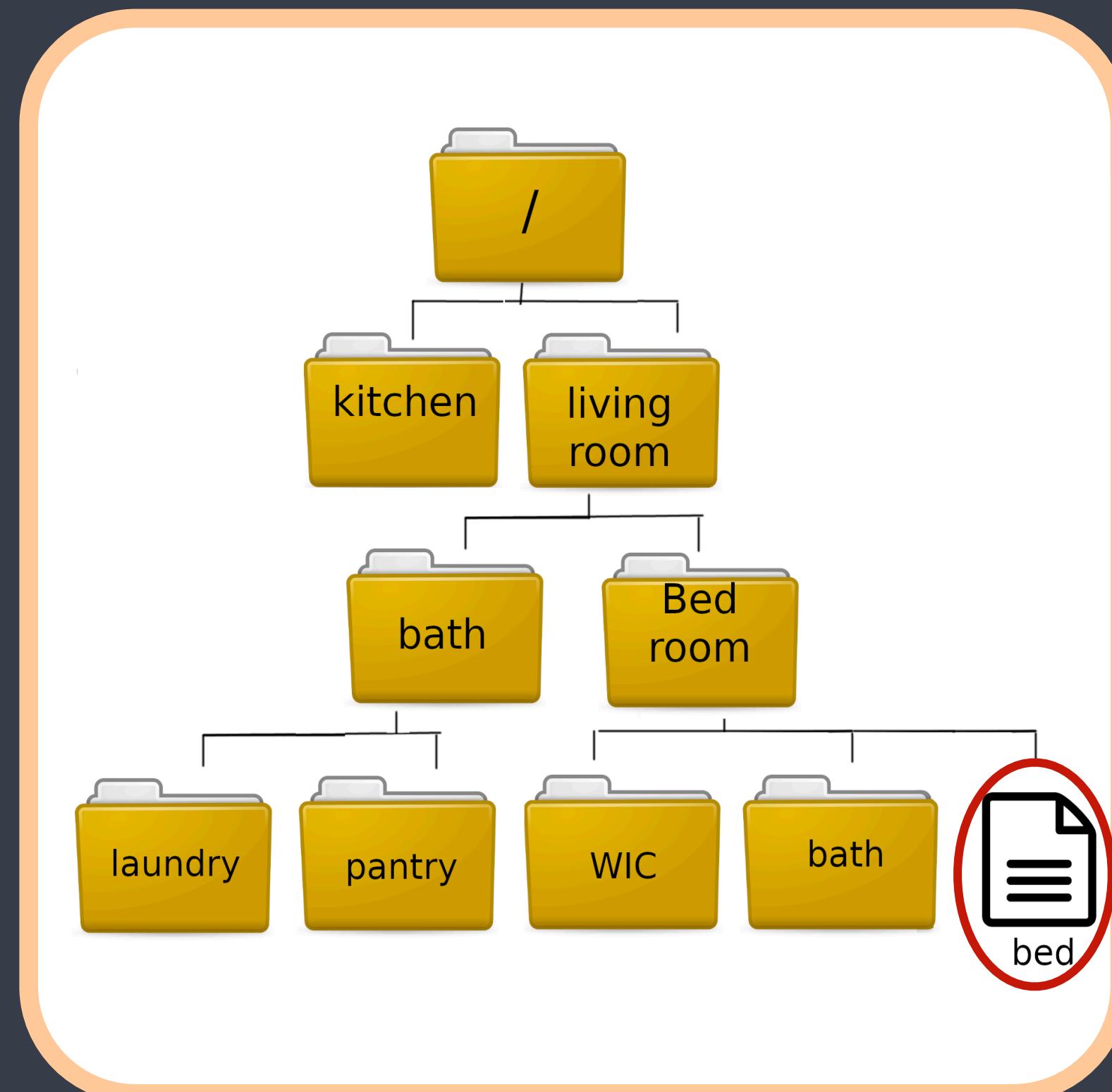


THE DIRECTORY TREE

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



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

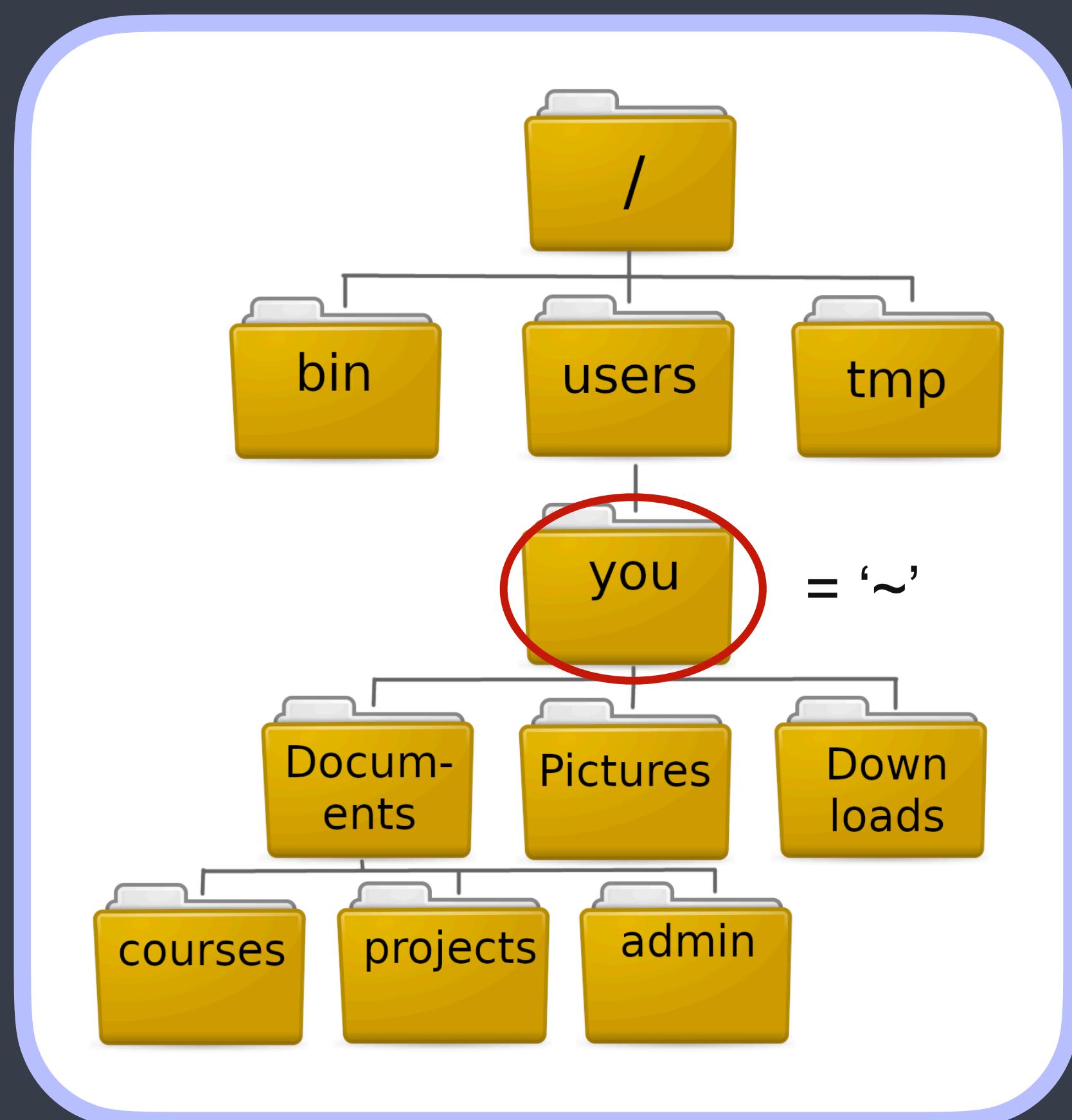


THE HOME DIRECTORY

- 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!**
- 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

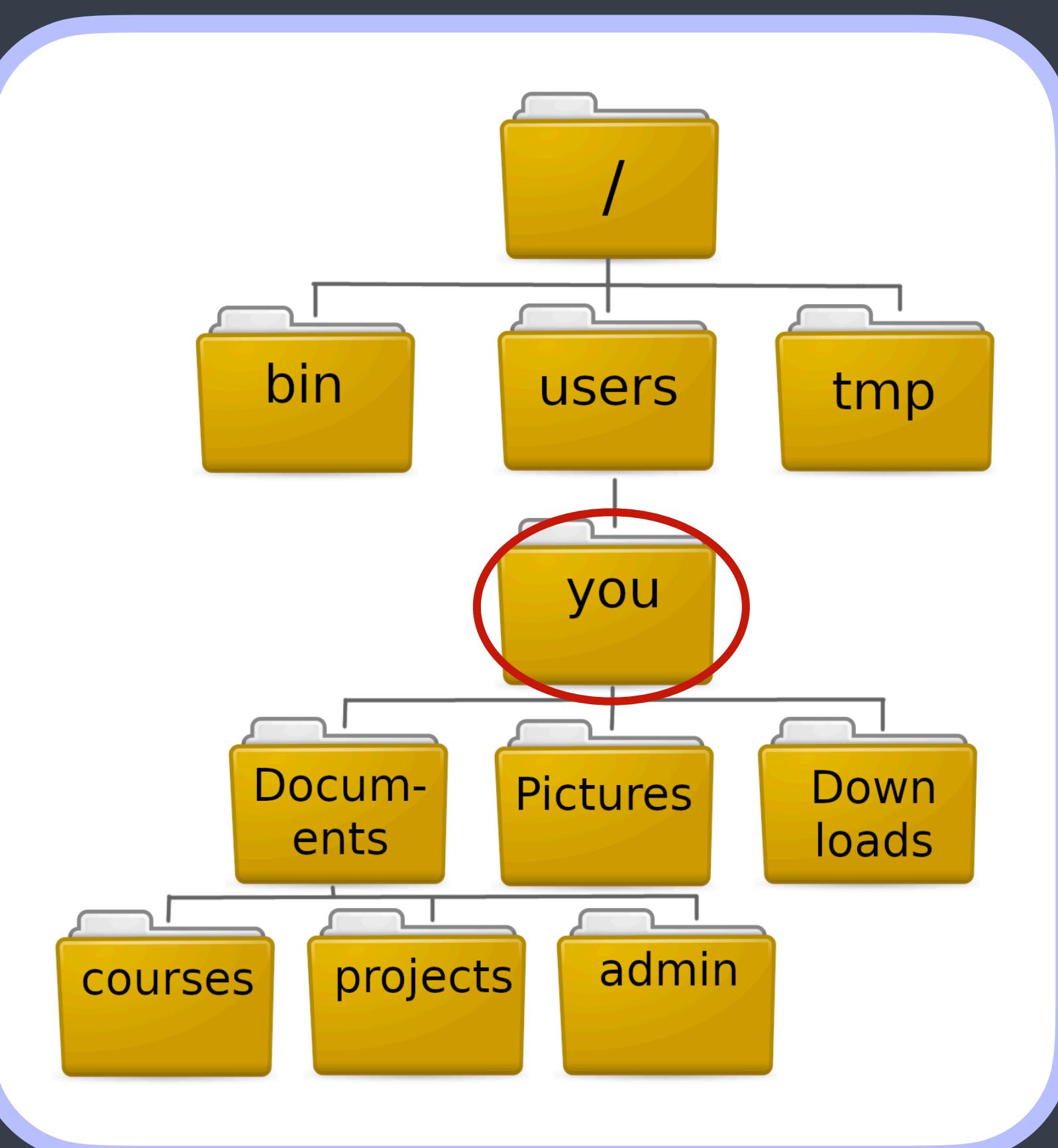
- To **list all contents of a directory**, including other directories, use the command:

```
$ ls
```

- ls will list what is in your current **wd**.
- You can list other directories by specifying the **path** to them:

```
$ ls /home/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
```

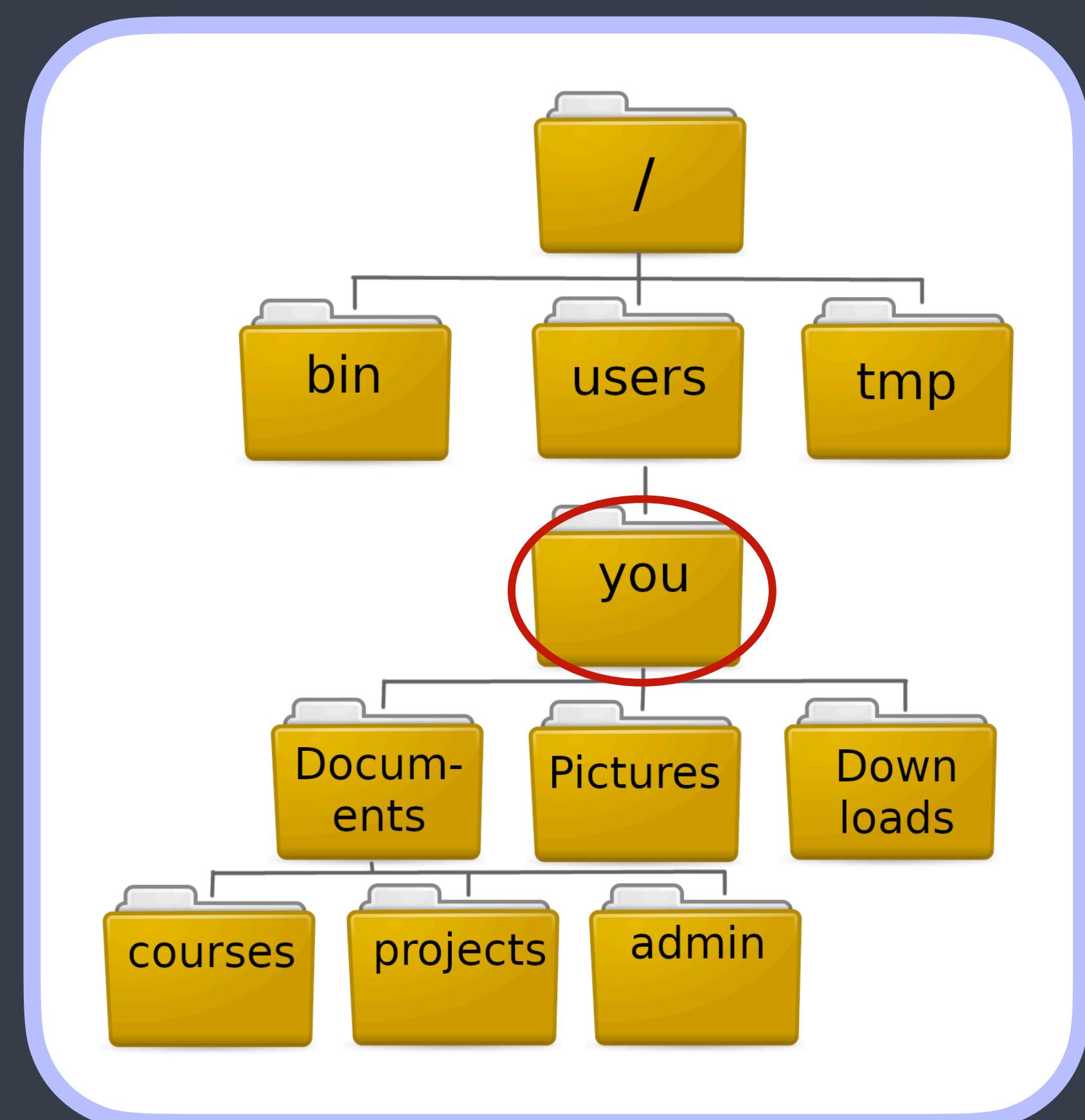
- You can also move several directories down:

```
$ 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
```

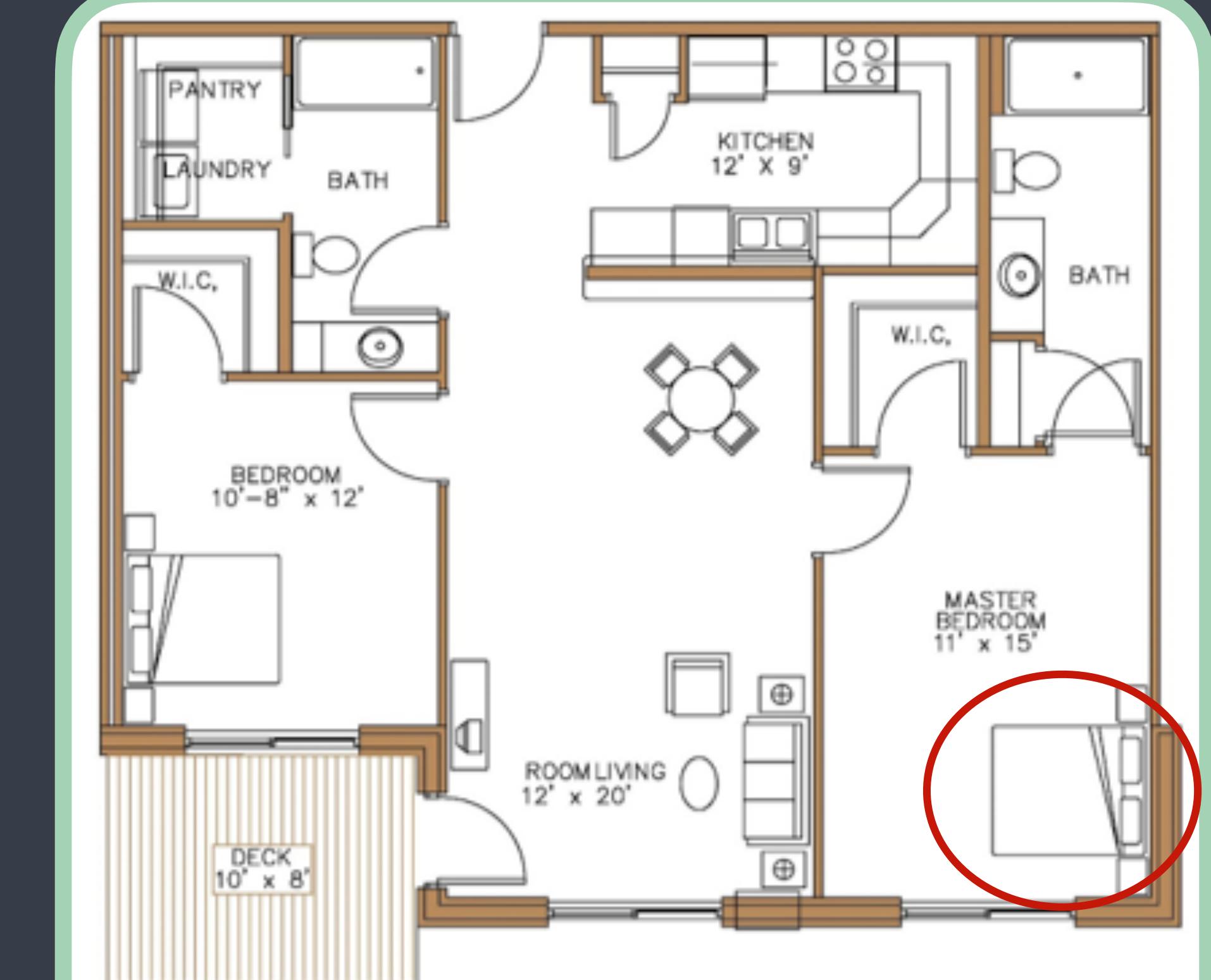
- '**cd**' without any arguments brings you back to your home directory. Try it out!



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 your bed would be:

/entrance/living_room/Bedroom/bved.txt



Two Bedroom 1,003 sq.ft.

PATHS

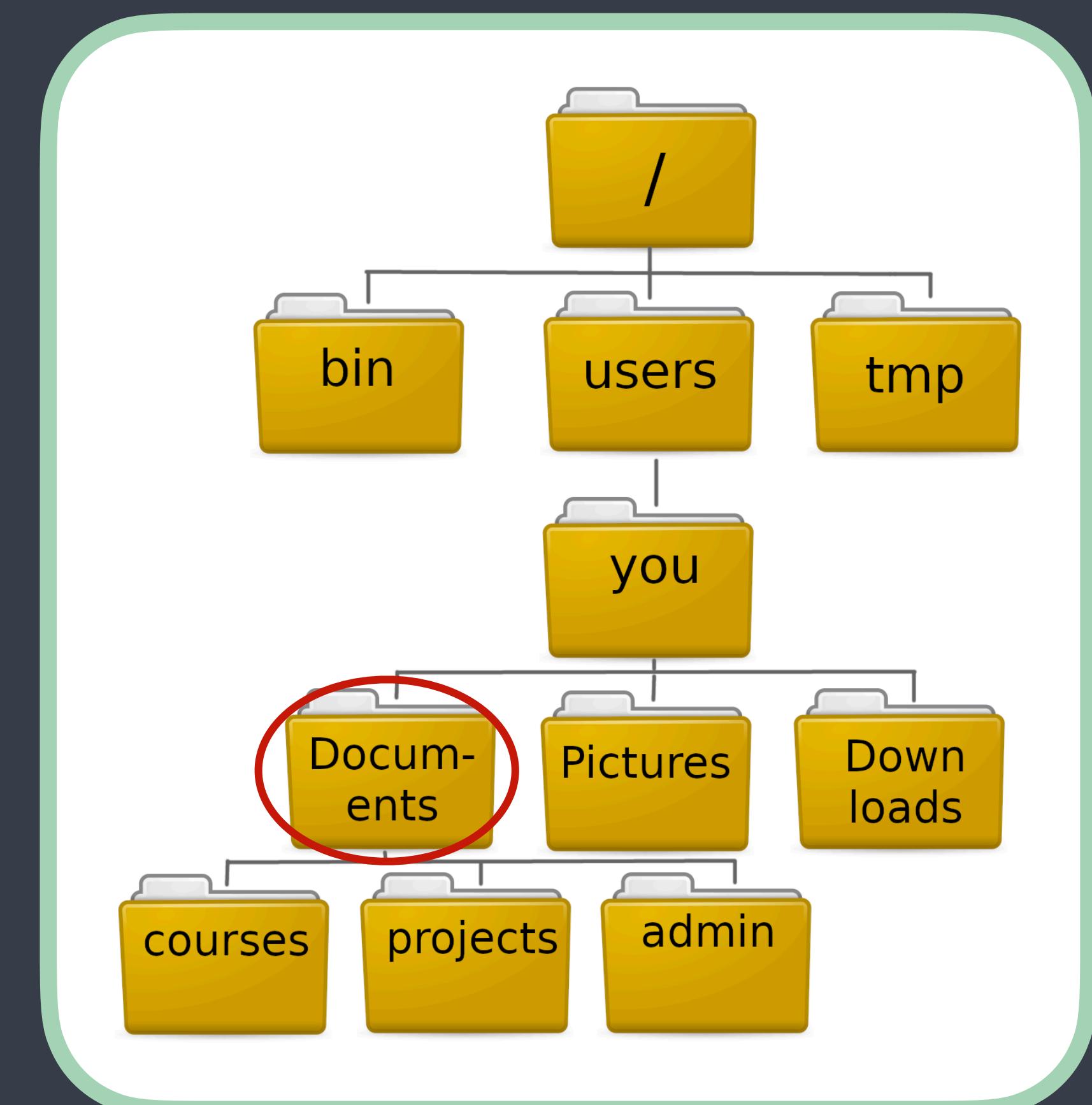
- Paths can be full/absolute or relative.
- Full paths always start at the root. Since ~ represent the path to your home directory it naturally includes the root. So

/home/user/[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

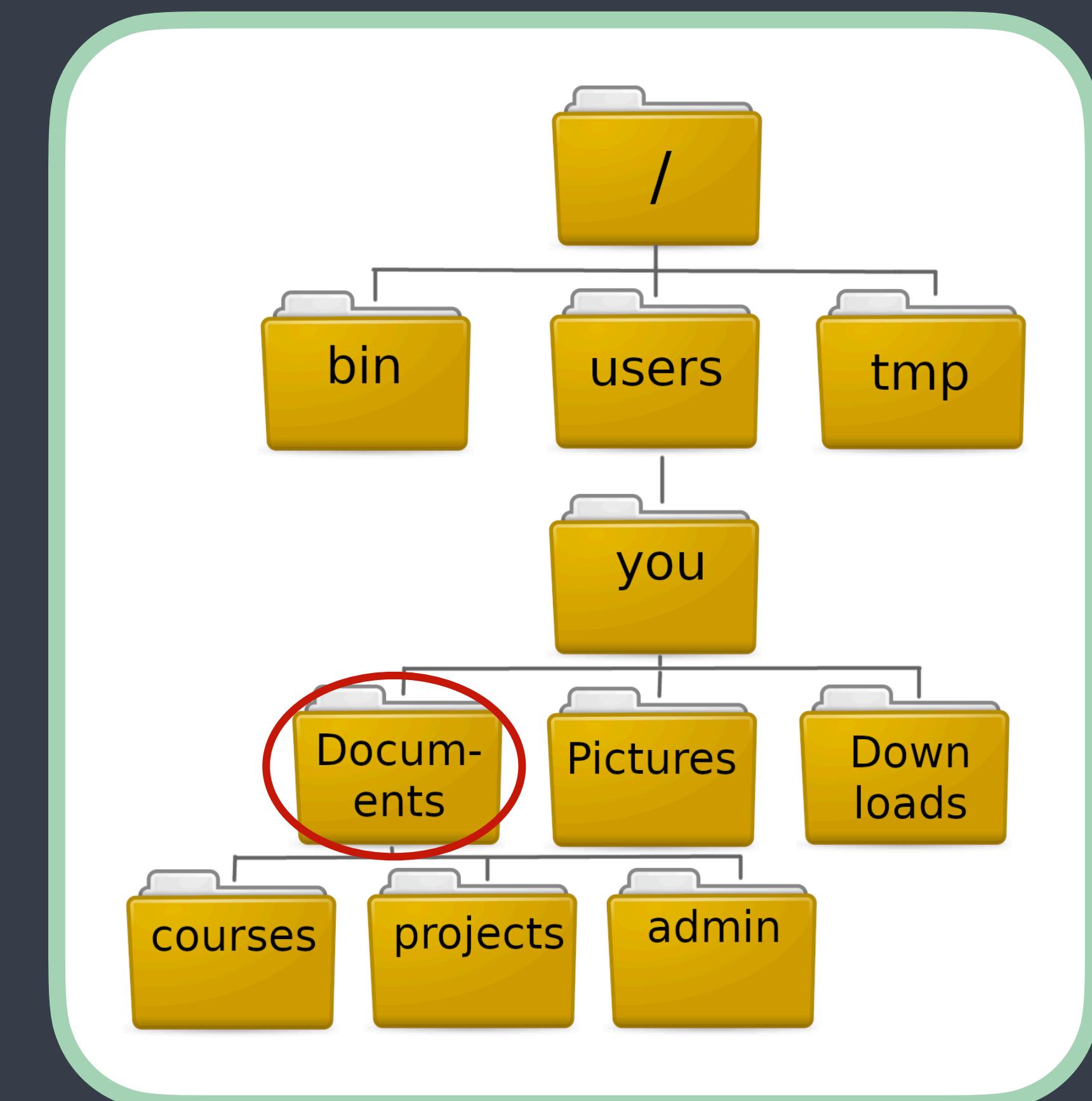
- A **relative paths** is an address relative to the current working directory.
- If my working directory is 'Documents', I can address this presentation as:

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**:

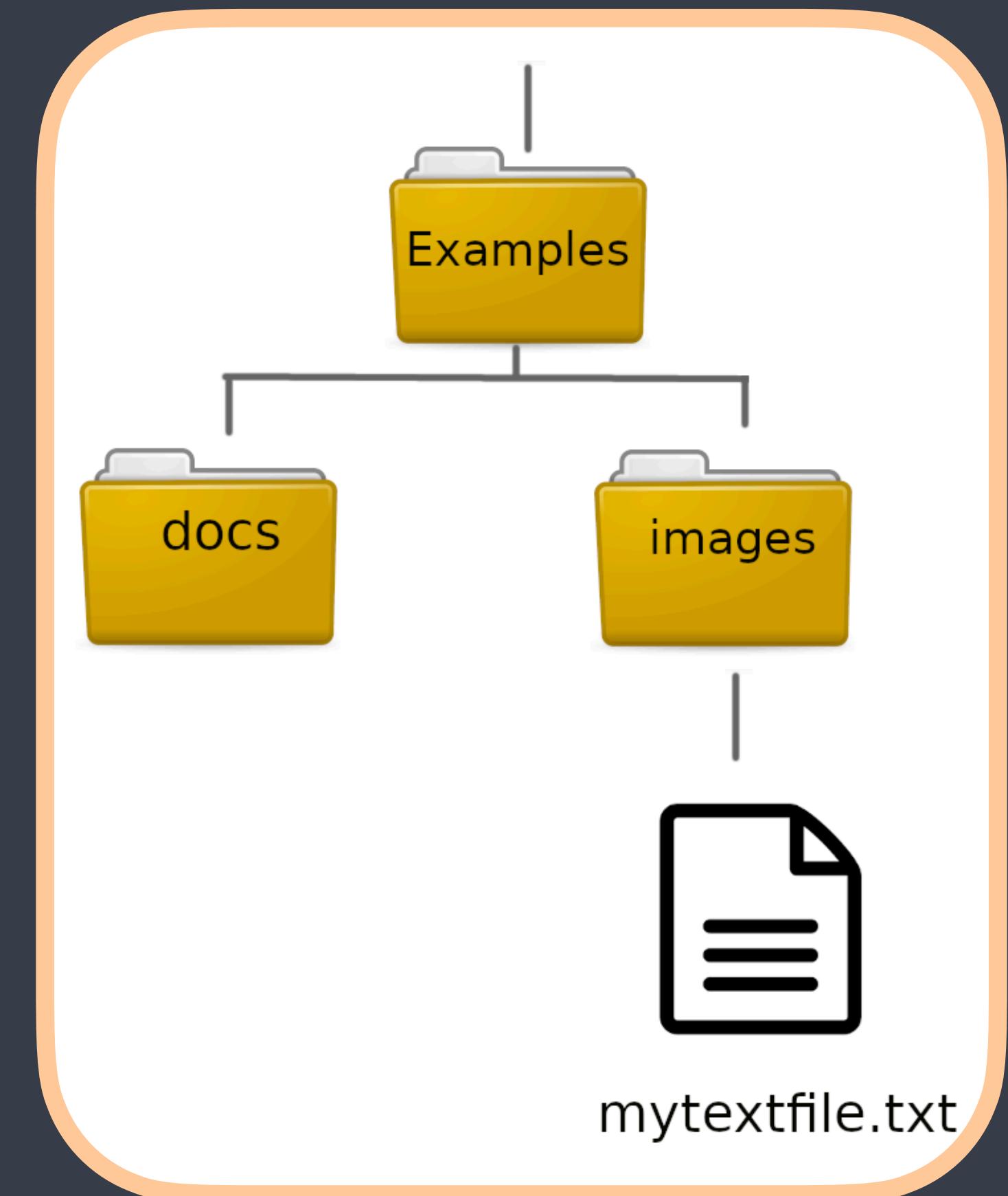
```
$ mv mytextfile.txt ..docs
```

- **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

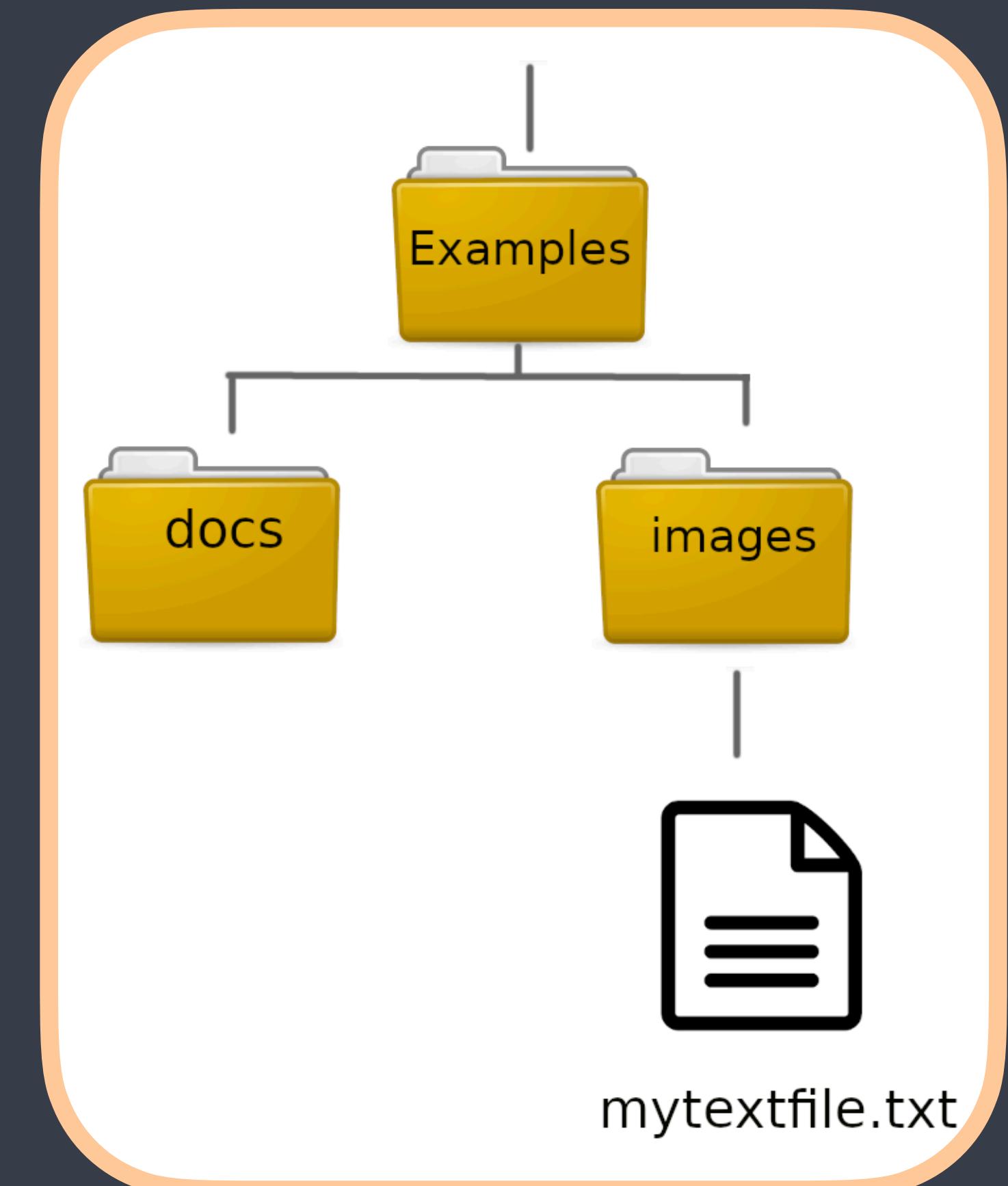
- Files can also be duplicated with the **cp** command:

```
$ 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

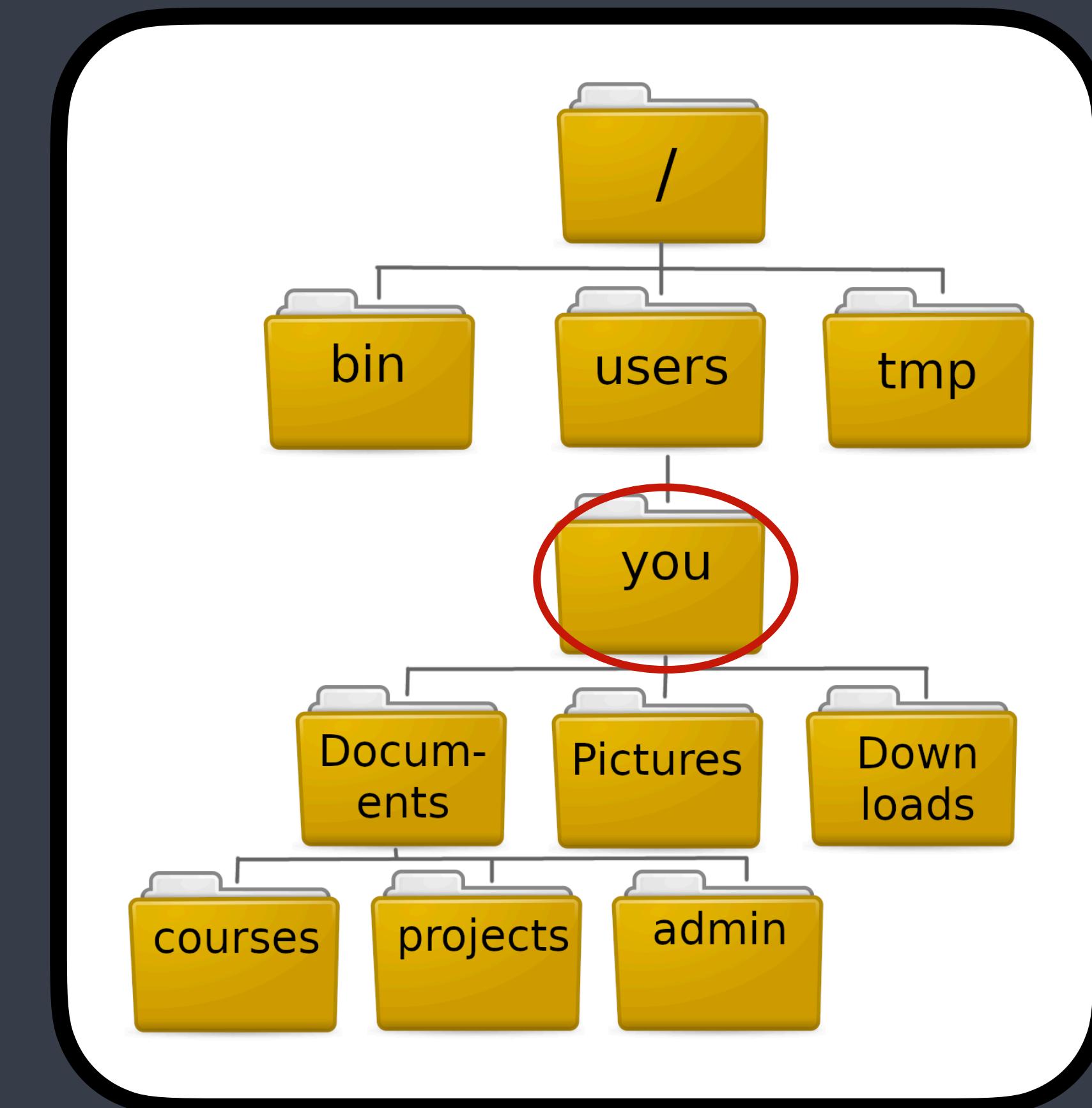
WHEN THINGS GO WRONG

Assuming I am in my home directory and I write:

```
$ cd courses
```

What will happen?

```
henrike@henrike-Swift-SF314-42: ~
henrike@henrike-Swift-SF314-42: ~ 56x24
henrike@henrike-Swift-SF314-42:~$ pwd
/home/henrike
henrike@henrike-Swift-SF314-42:~$ ls
Desktop  Downloads  Pictures  snap      Videos
Documents  Music    Public     Templates
henrike@henrike-Swift-SF314-42:~$ ls Documents/
courses  planets1.xcf  tap-fix.sh
henrike@henrike-Swift-SF314-42:~$ cd courses
bash: cd: courses: No such file or directory
henrike@henrike-Swift-SF314-42:~$
```



WHEN THINGS GO WRONG

1

Don't panic!

2

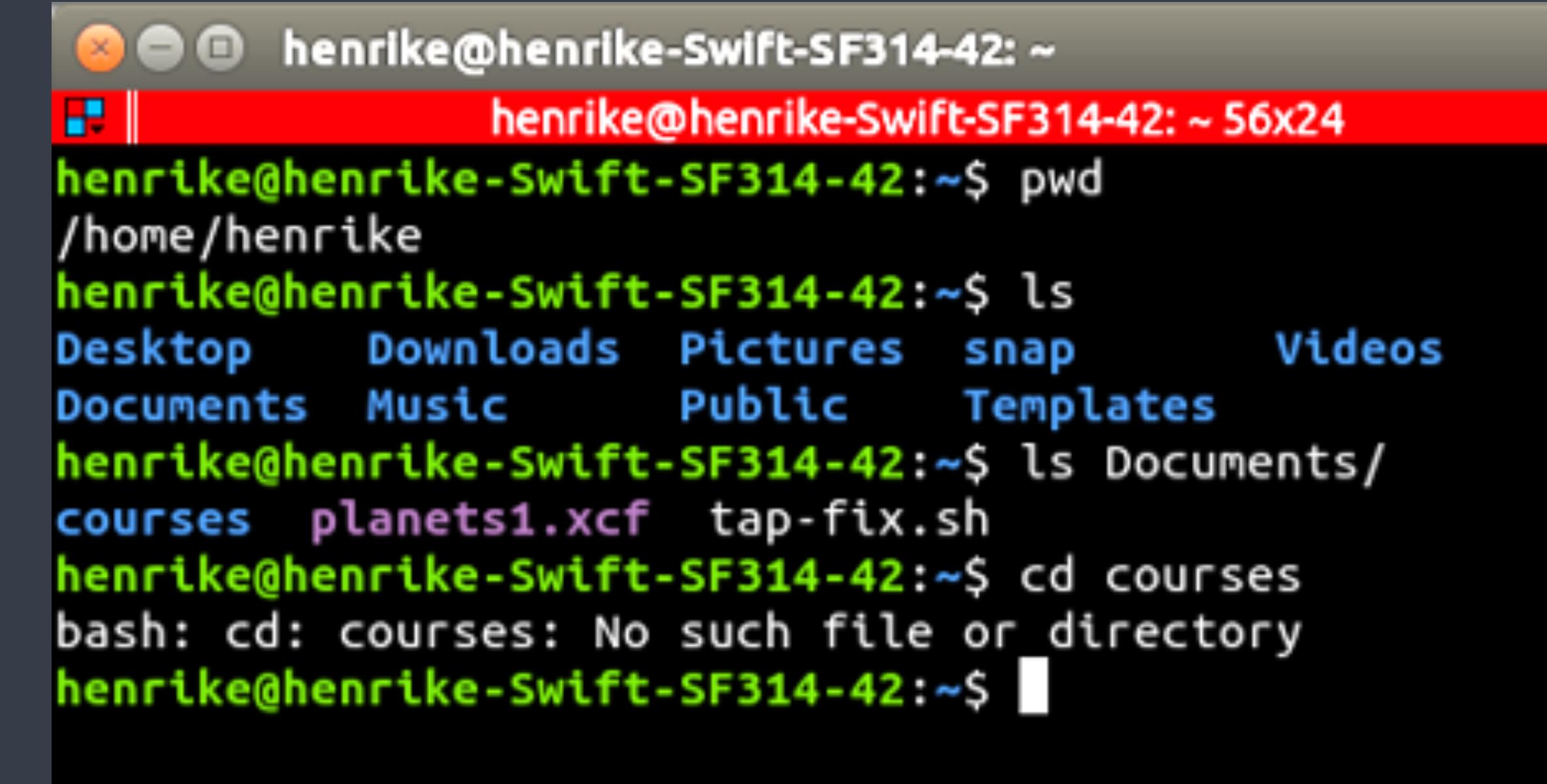
Read the error message

3

Try to correct your error

4

If lost, google or read the help page



```
henrike@henrike-Swift-SF314-42: ~
henrike@henrike-Swift-SF314-42: ~ 56x24
henrike@henrike-Swift-SF314-42:~$ pwd
/home/henrike
henrike@henrike-Swift-SF314-42:~$ ls
Desktop    Downloads  Pictures  snap      Videos
Documents  Music     Public    Templates
henrike@henrike-Swift-SF314-42:~$ ls Documents/
courses   planets1.xcf  tap-fix.sh
henrike@henrike-Swift-SF314-42:~$ cd courses
bash: cd: courses: No such file or directory
henrike@henrike-Swift-SF314-42:~$
```

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**

```
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 ..
```

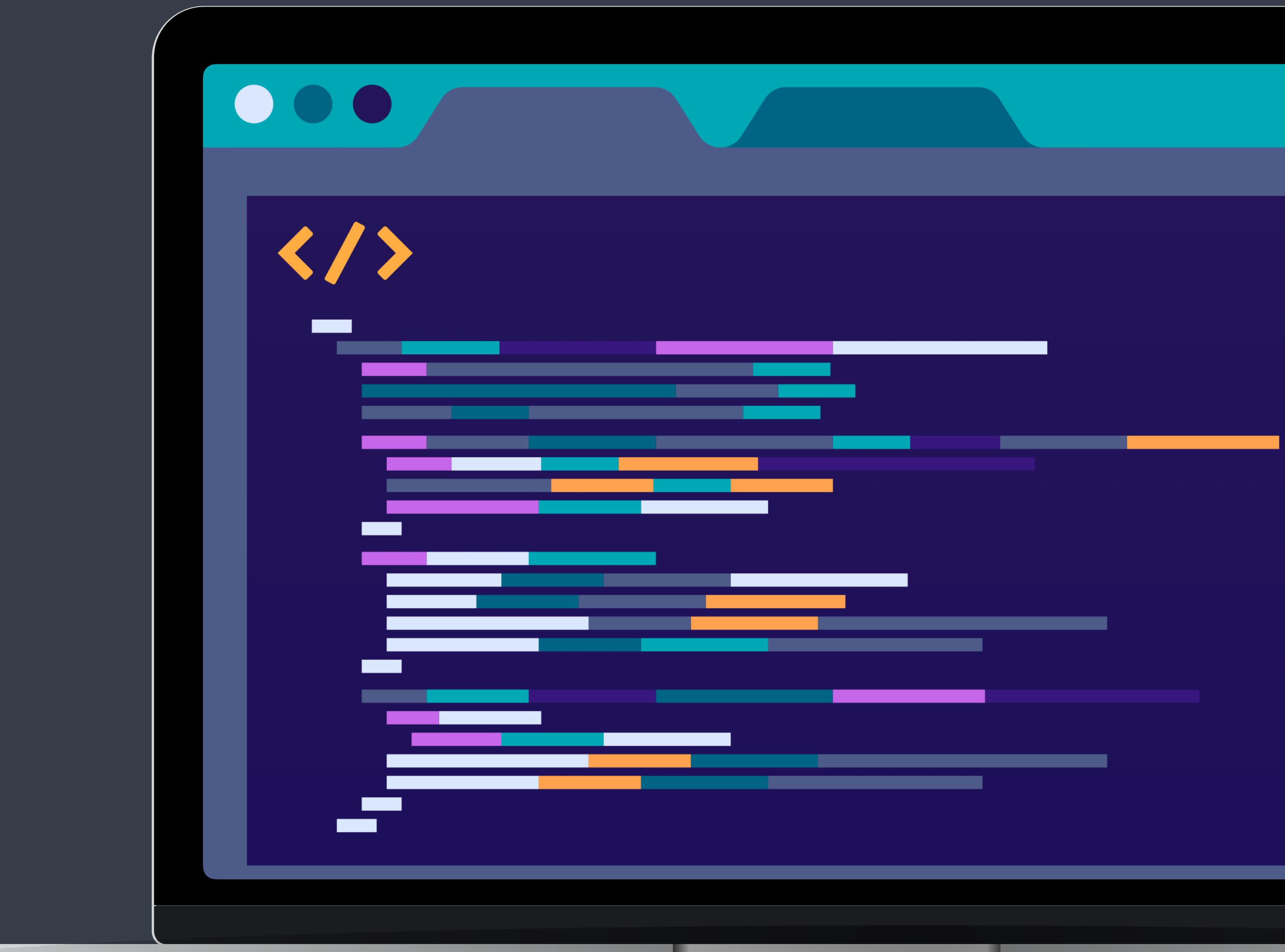
ge ls(1) line 1/297 7% (press h for help or q to quit)

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

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.



DO'S and DON'TS



Consistent and Logical Directory Structure and Naming



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).

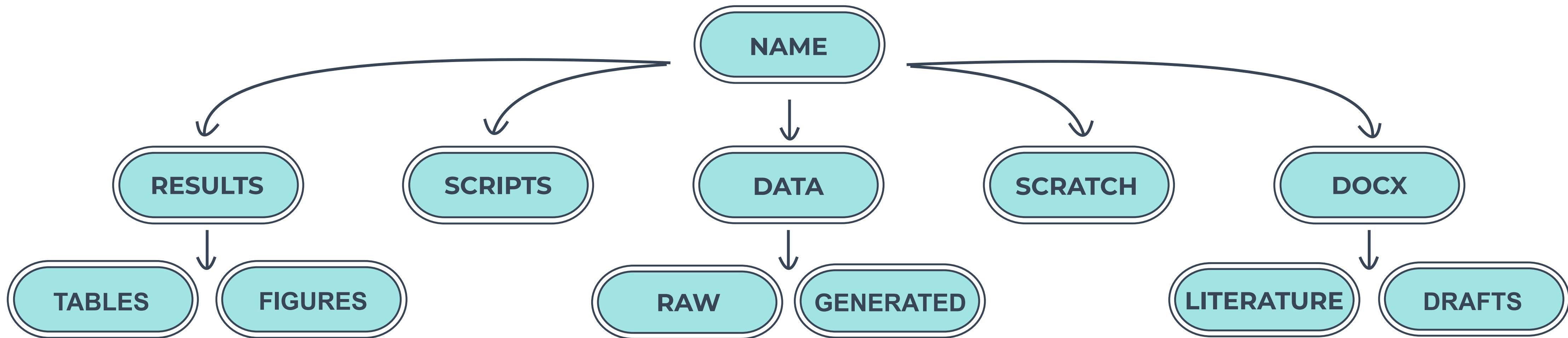
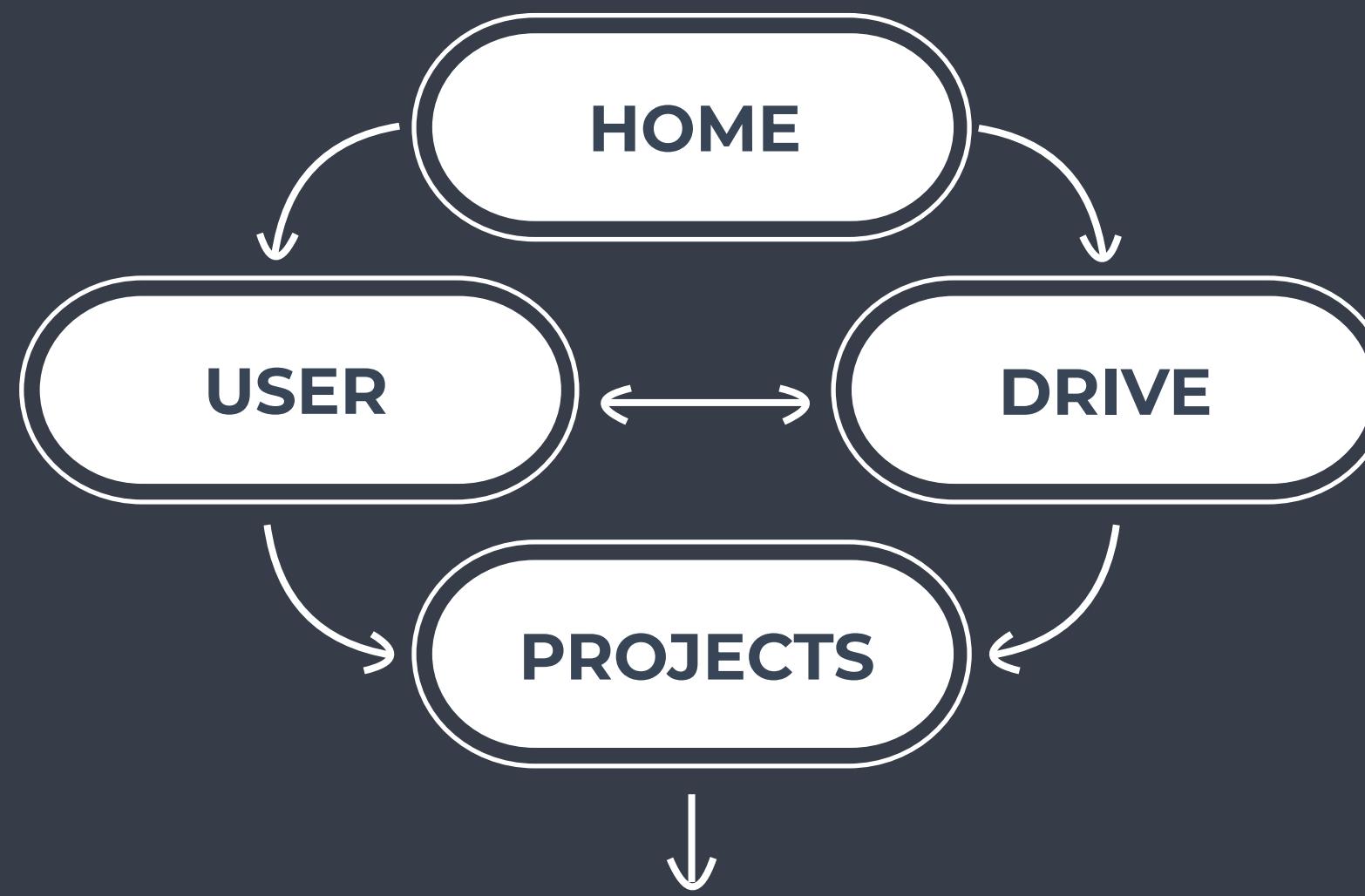


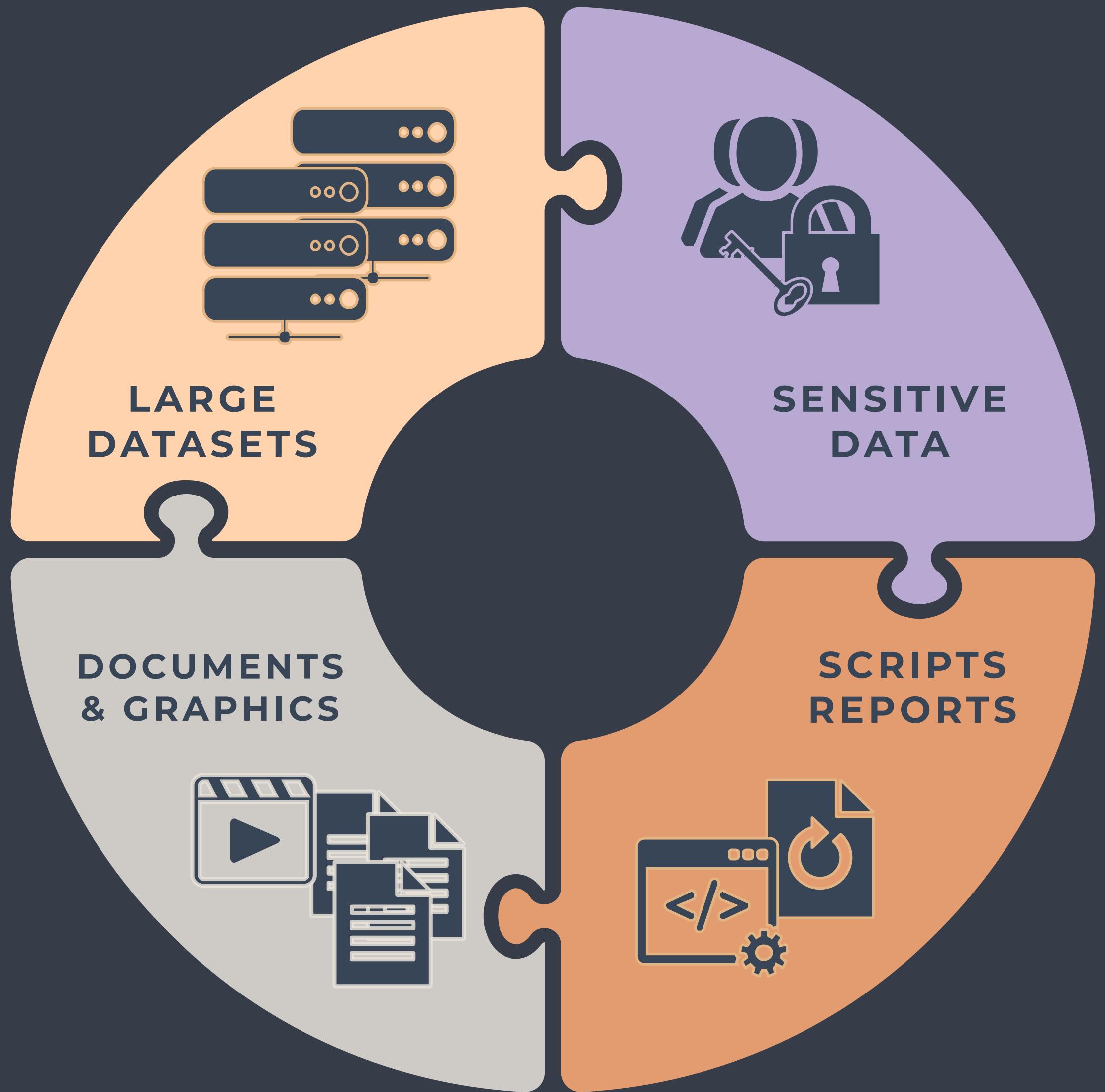
Don't Store Large Data/Files on Your Local Computer



Don't Store Sensitive Datasets on Your Computer

A SUGGESTION FOR STRUCTURE





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
- SCRIPTS, MARKDOWNS:
 - git/GitHub**
(KU drives)
- DOCX, EXCEL, POWERPOINT:
 - Google Drive
 - Dropbox
 - (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>

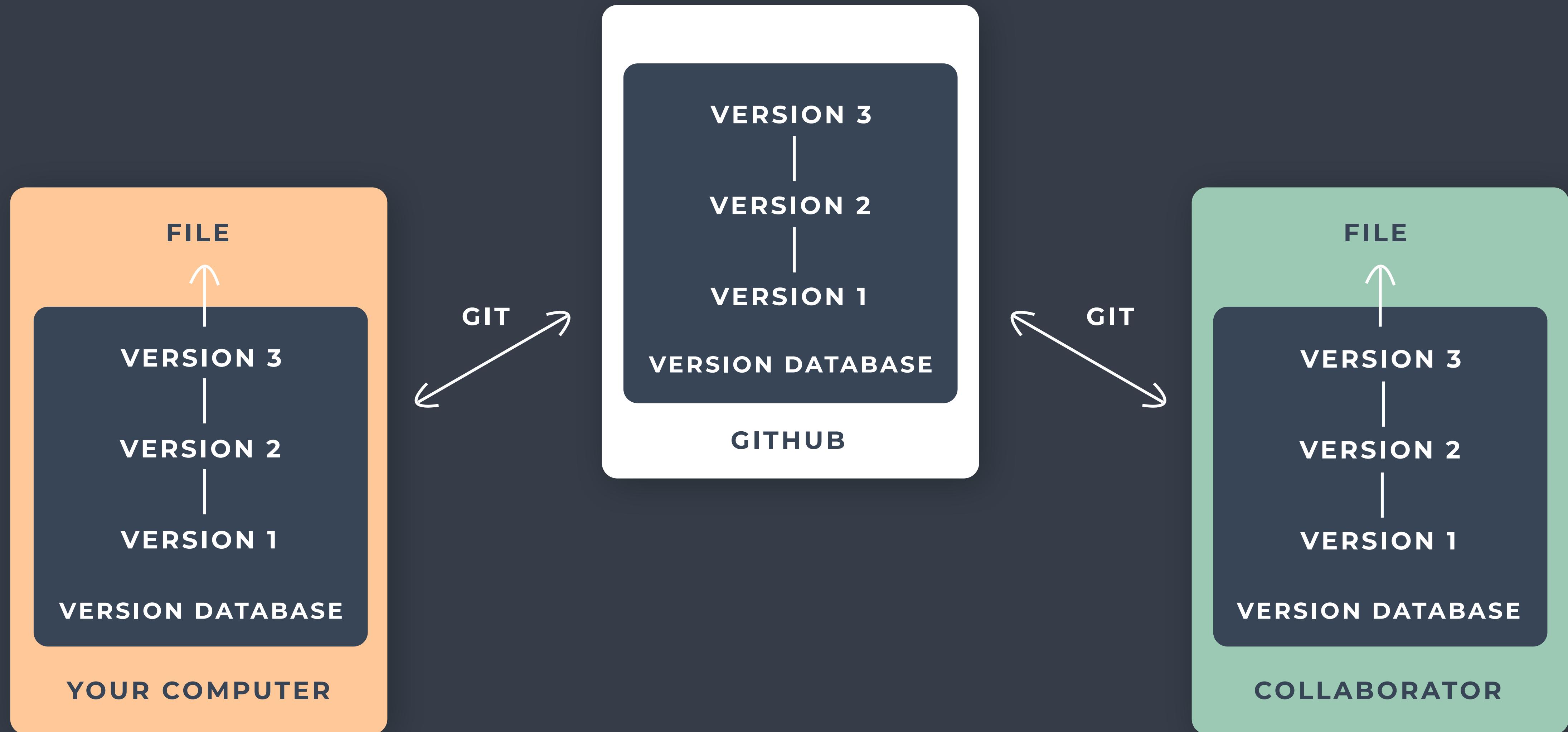


GitHub is a web-based hosting service for version control that uses git

Other services exist.

<https://github.com>

WHAT IS A VERSION CONTROL SYSTEM?

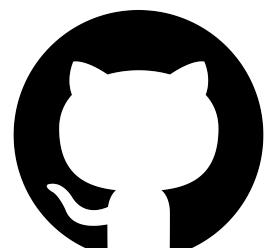


GIT WORKS FROM THE COMMAND LINE

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)

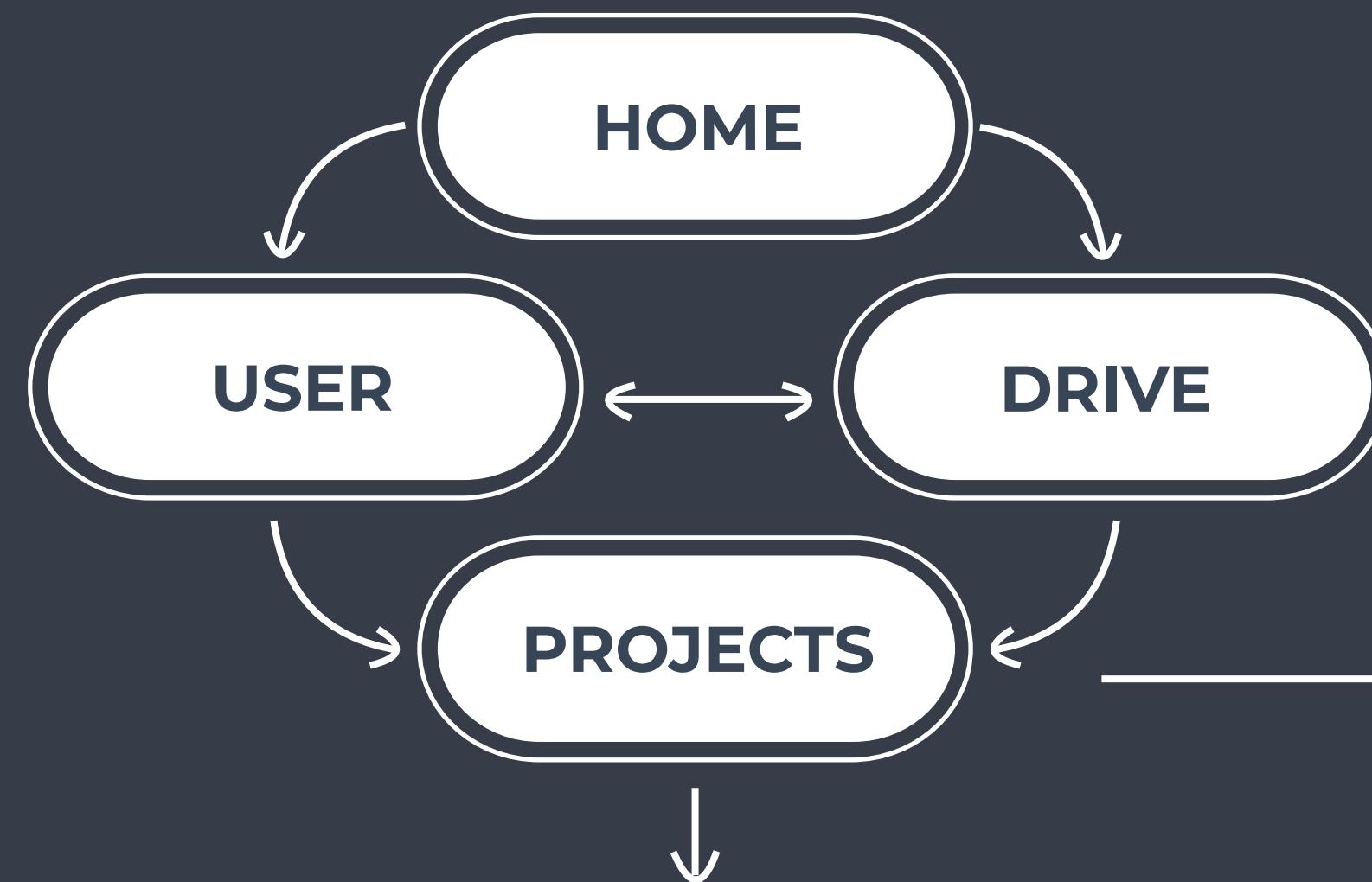
git add (add new files or changes)

git commit -m (commit the changes)

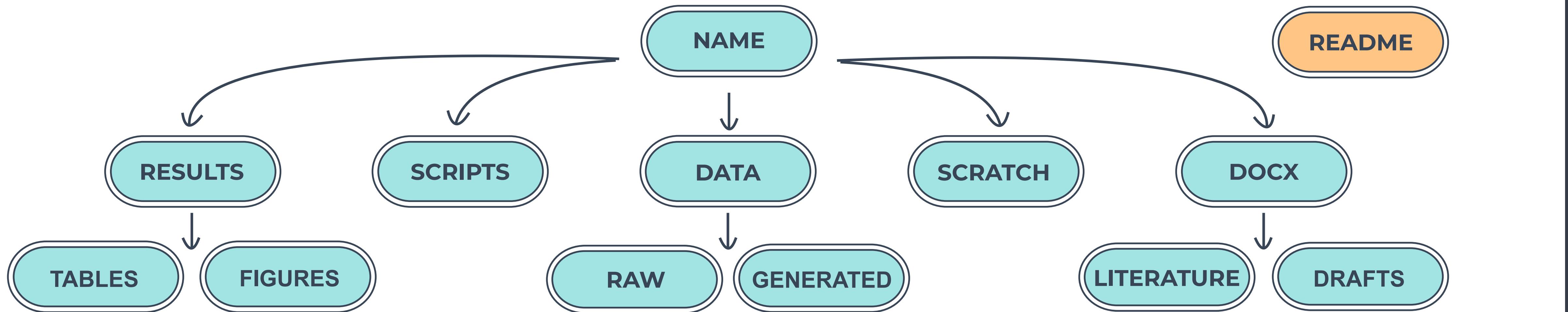
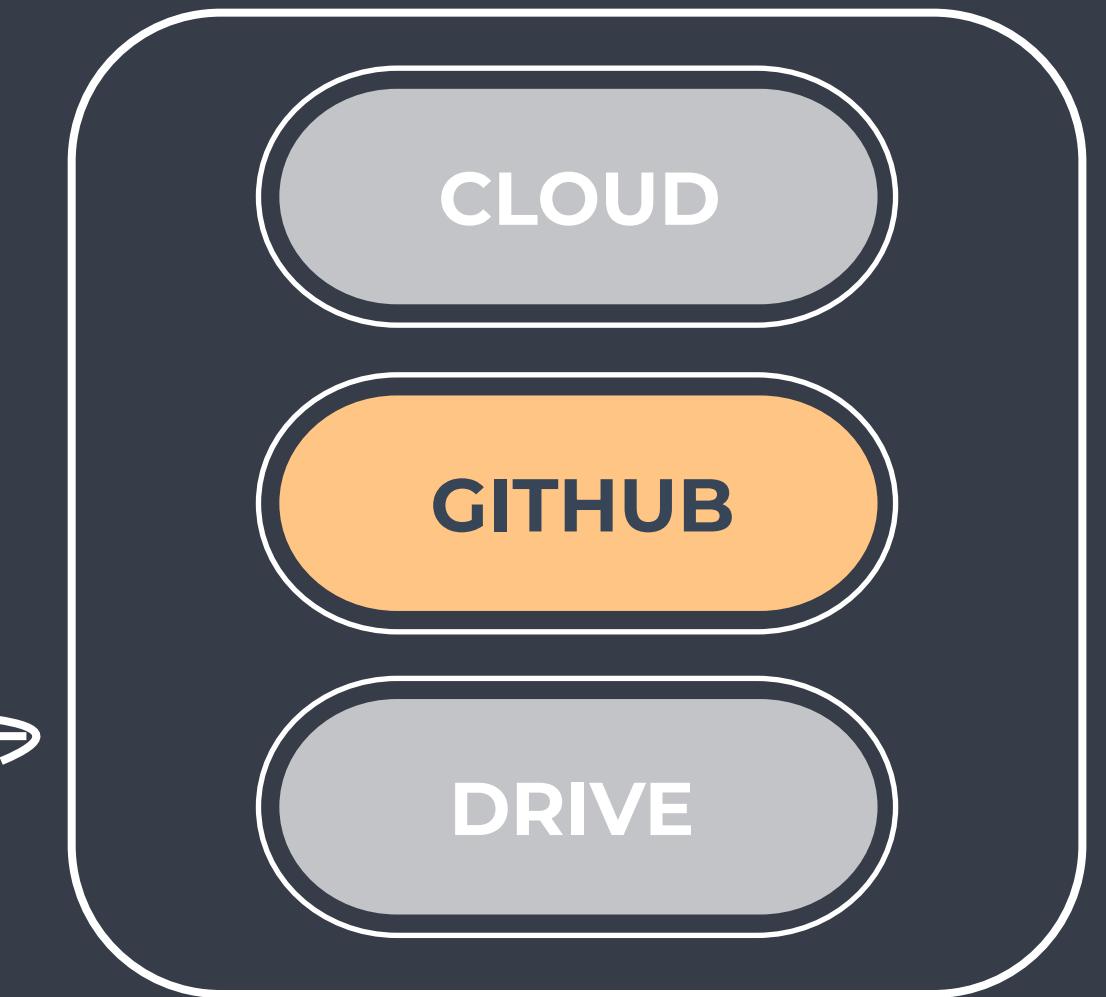
git push (push the changes to GitHub)

A SUGGESTION FOR STRUCTURE

LOCAL COMPUTER

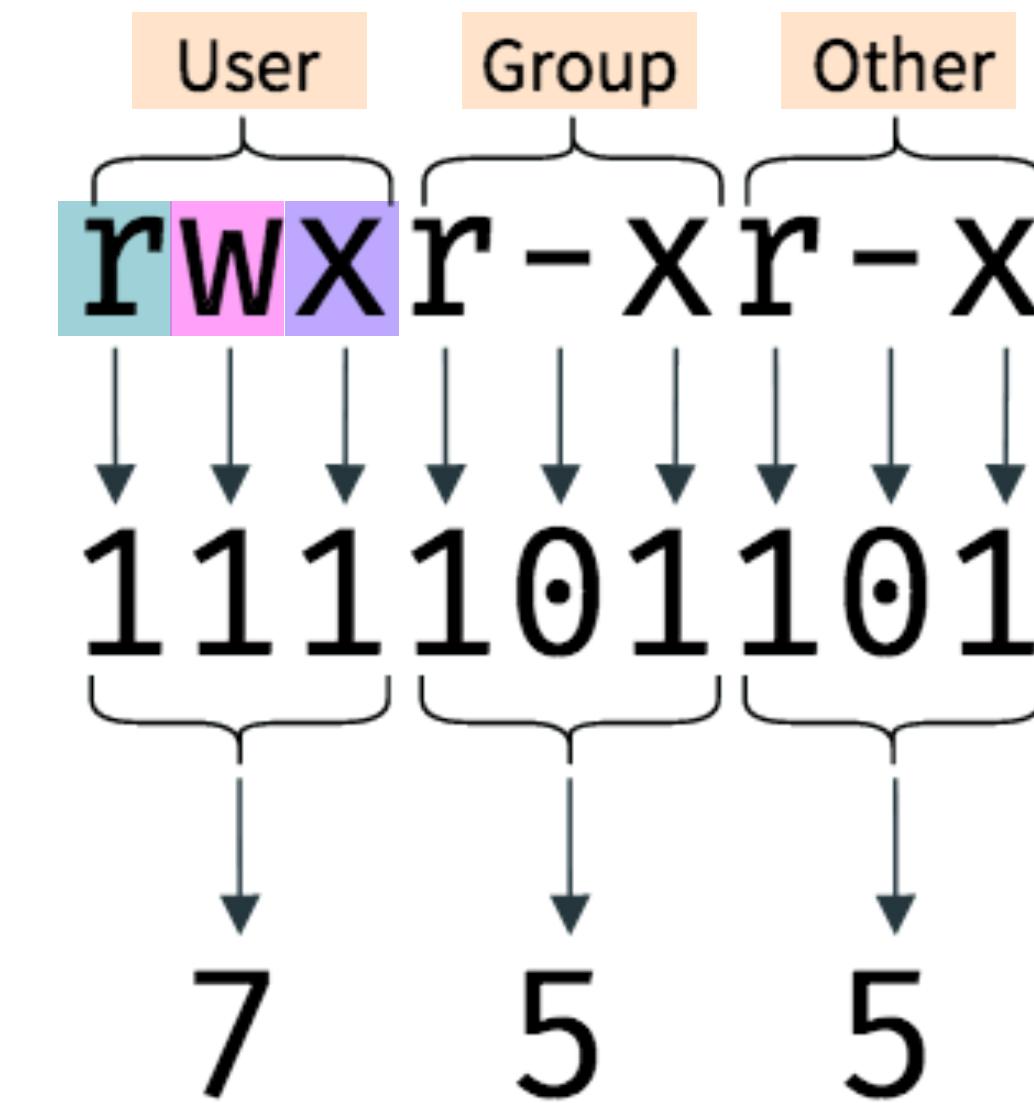


EXTERNAL BACKUP



USERS, GROUPS & PERMISSIONS

- Files & directories have **permission settings**
- Permissions denote who can **read**, **write** (edit) and **execute** a file/dir
- Permissions can be changed **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



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

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

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

SPECIAL CHARACTERS

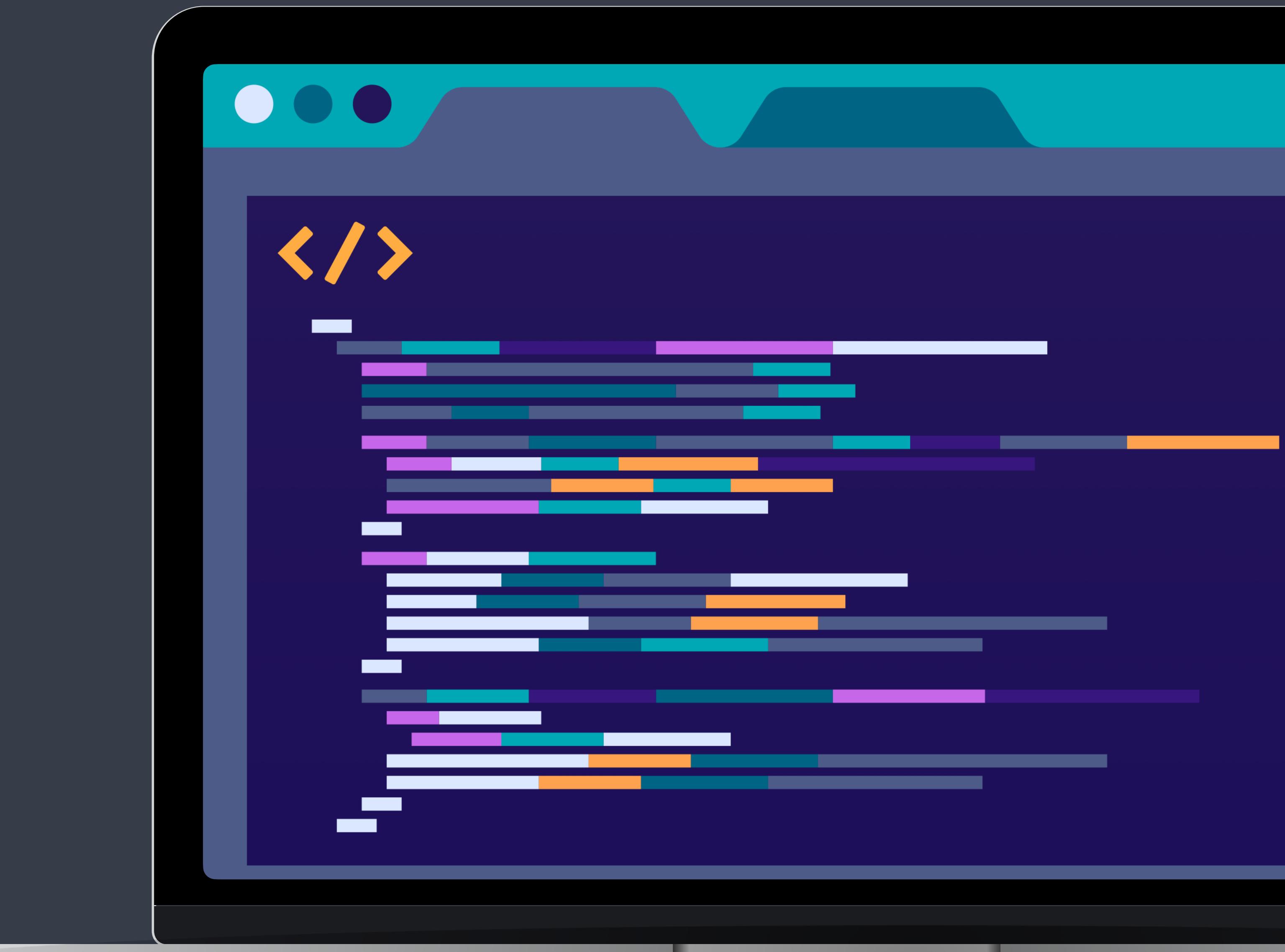
```
touch [name] # make a file  
mkdir [name] # make a dir  
rm (-r) [name] # remove a file or dir  
  
cp [name] # copy a file/dir  
mv [name] [path] # move file/dir
```

FILE/DIR BASICS

```
ls -lh * # all sizes  
ls -lh [name] # file/dir size  
du -sh # total size  
  
chmod +rwx [name] # add a permission  
chmod -rwx [name] # remove a permission
```

SIZE & 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. Interactive viewing.
 - 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
.

```
less myfile.txt > myfile_new.txt
```

**head
&
tail**

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

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

EDITING FILES

- **Nano** [file] :

- Simple and easy to use
- Limited functionality
- Exit with **Ctrl + x (Y + enter)**

- **VIM** [file] :

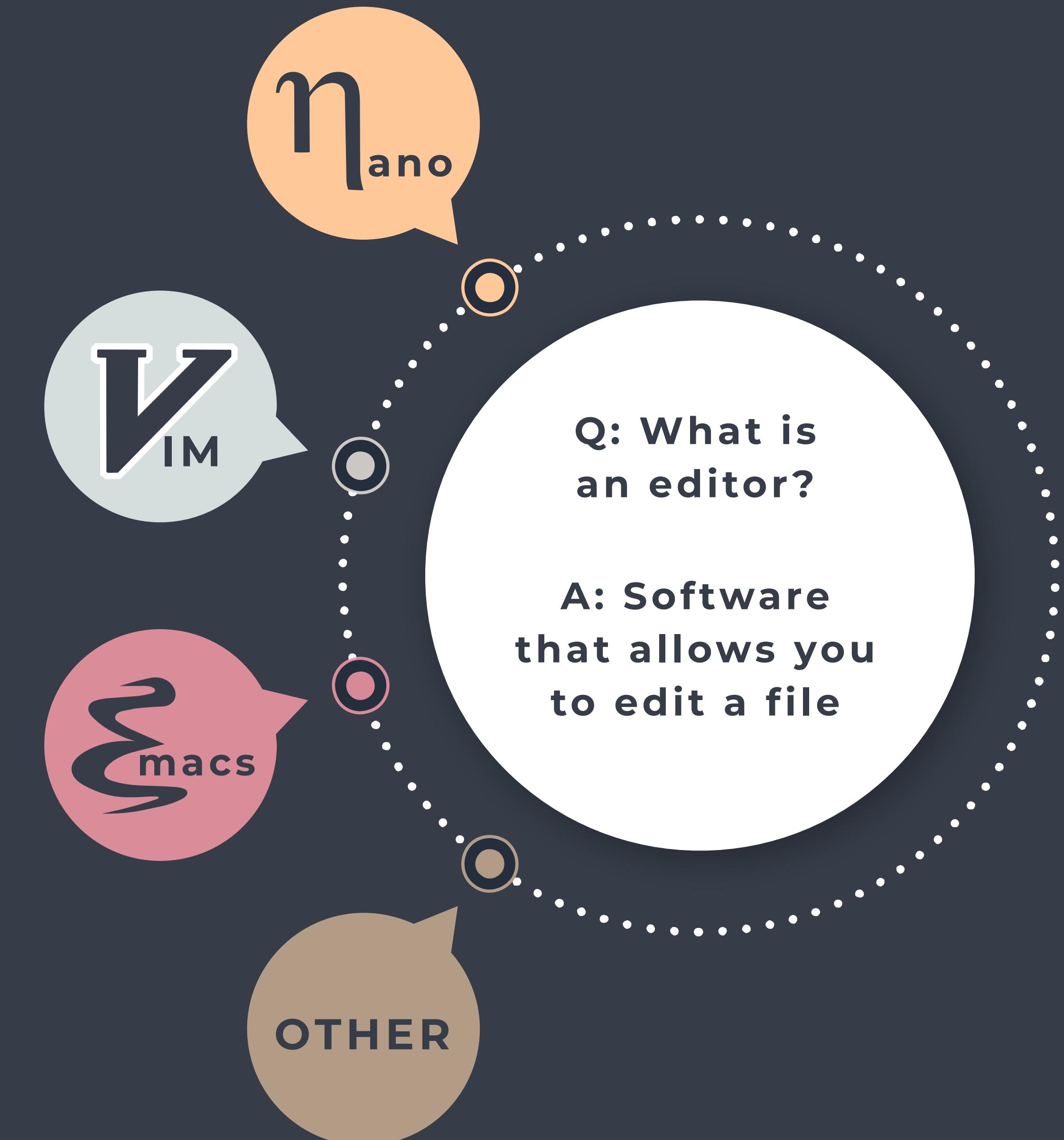
- Many functionalities, i.e. complex
- Keyboard shortcuts
- Exit with **:q (:w or :q!)**

- **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

**Standard gzip (GNU zip) compression.
May contain multiple files & directories**

.TAR

**An archive of multiple files put
together inside a single file.**

.TAR.GZ

**A compressed archive of multiple files
put together inside a single file.**

`gunzip -k [file.gz]`

`gzip -dk [file.gz]`

`unzip [file.zip]`

`tar -xf [file.tar.gz]`

`tar -tf [file.tar]`



CHEAT SHEET 2

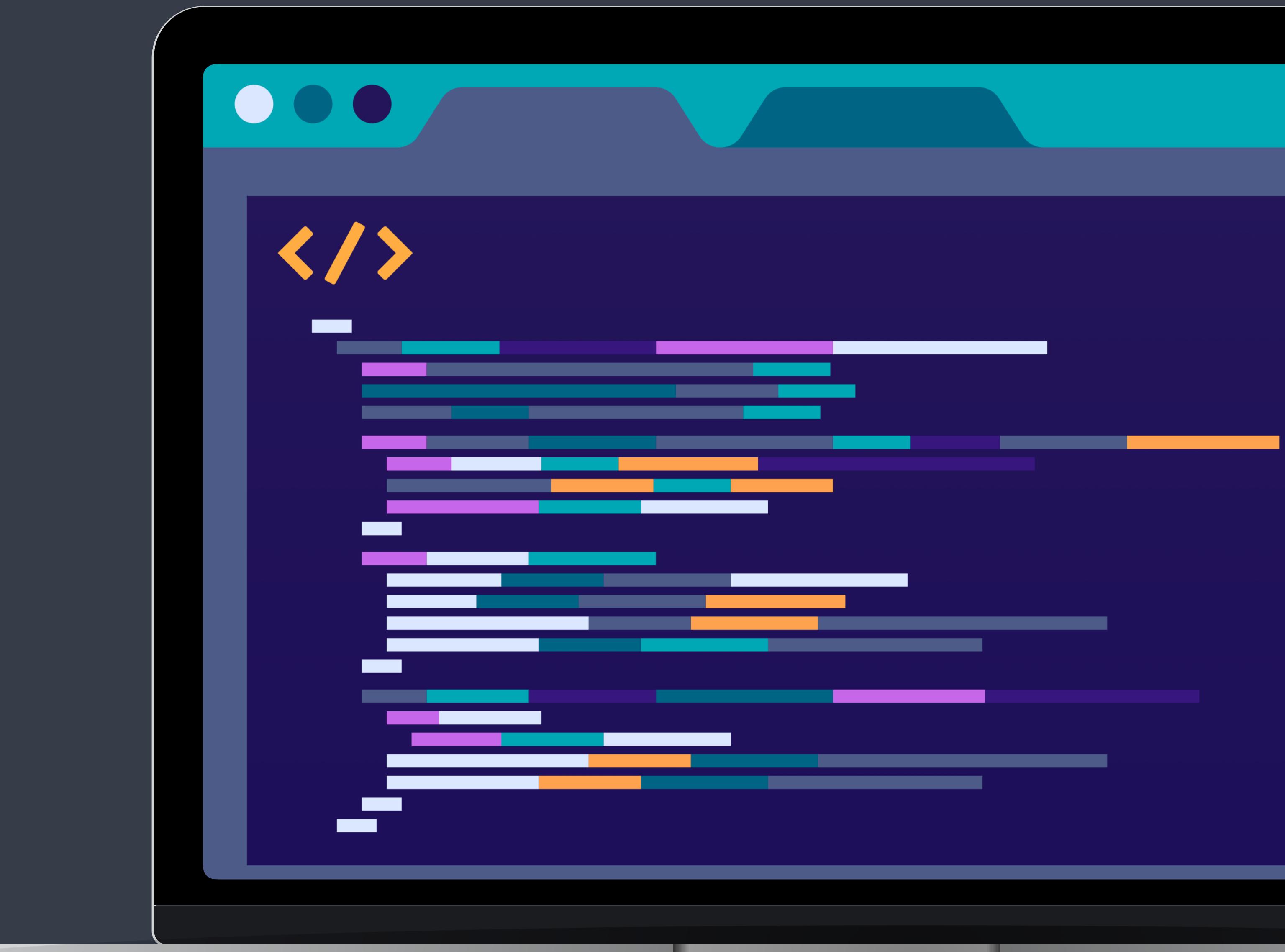
View Files

```
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 [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

sort -[f] [file]:
Sort a file/column
Number, character, mixed

Sort

uniq -[f] [file]:
Unique entries only

grep

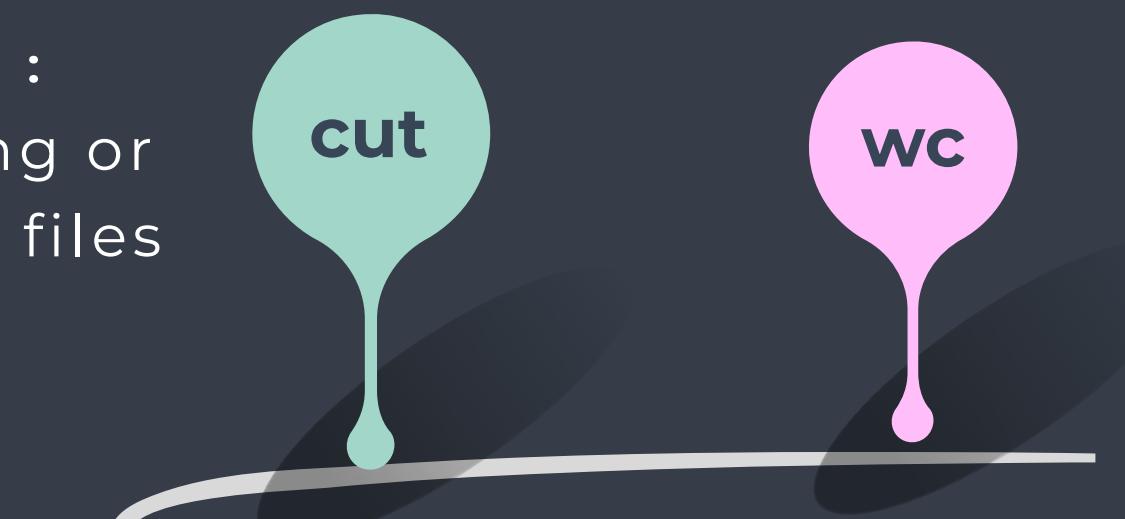
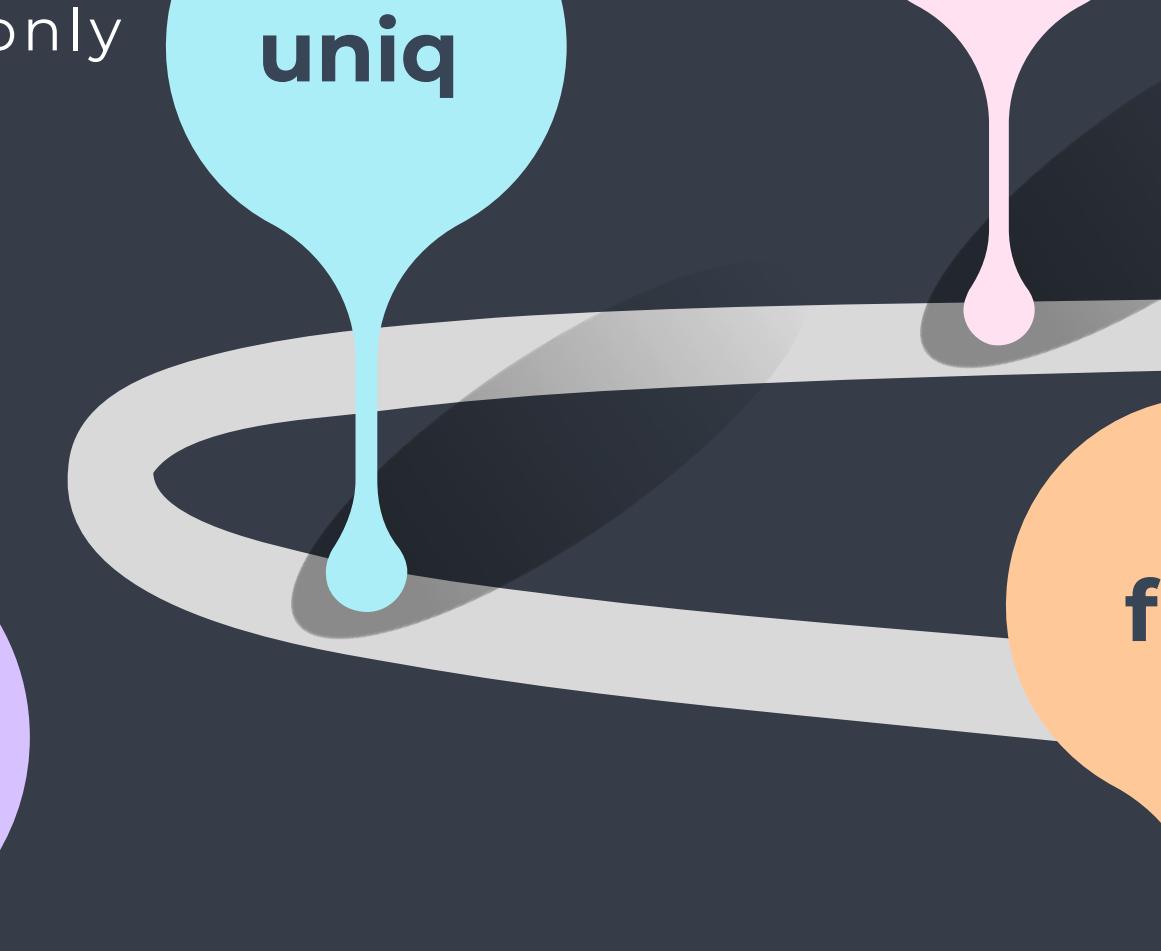
grep -[f] ['pattern'] [file]:
Search file for pattern



awk '{pattern}' [file]:
Search, replace, extract, manipulate
awk is tool & a command



paste -[f] [files]:
Merge corresponding or
subsequent lines of files

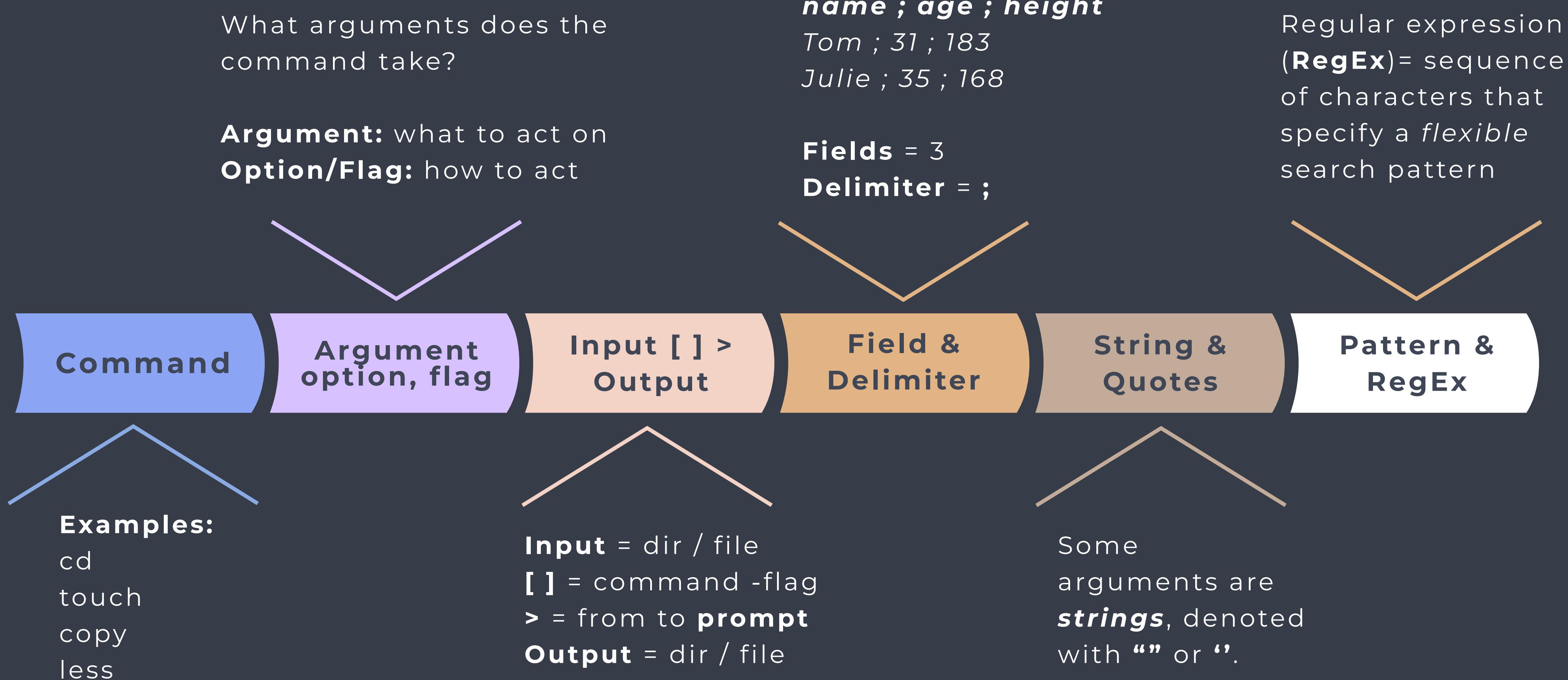


wc -[f] [file]:
Count lines & words

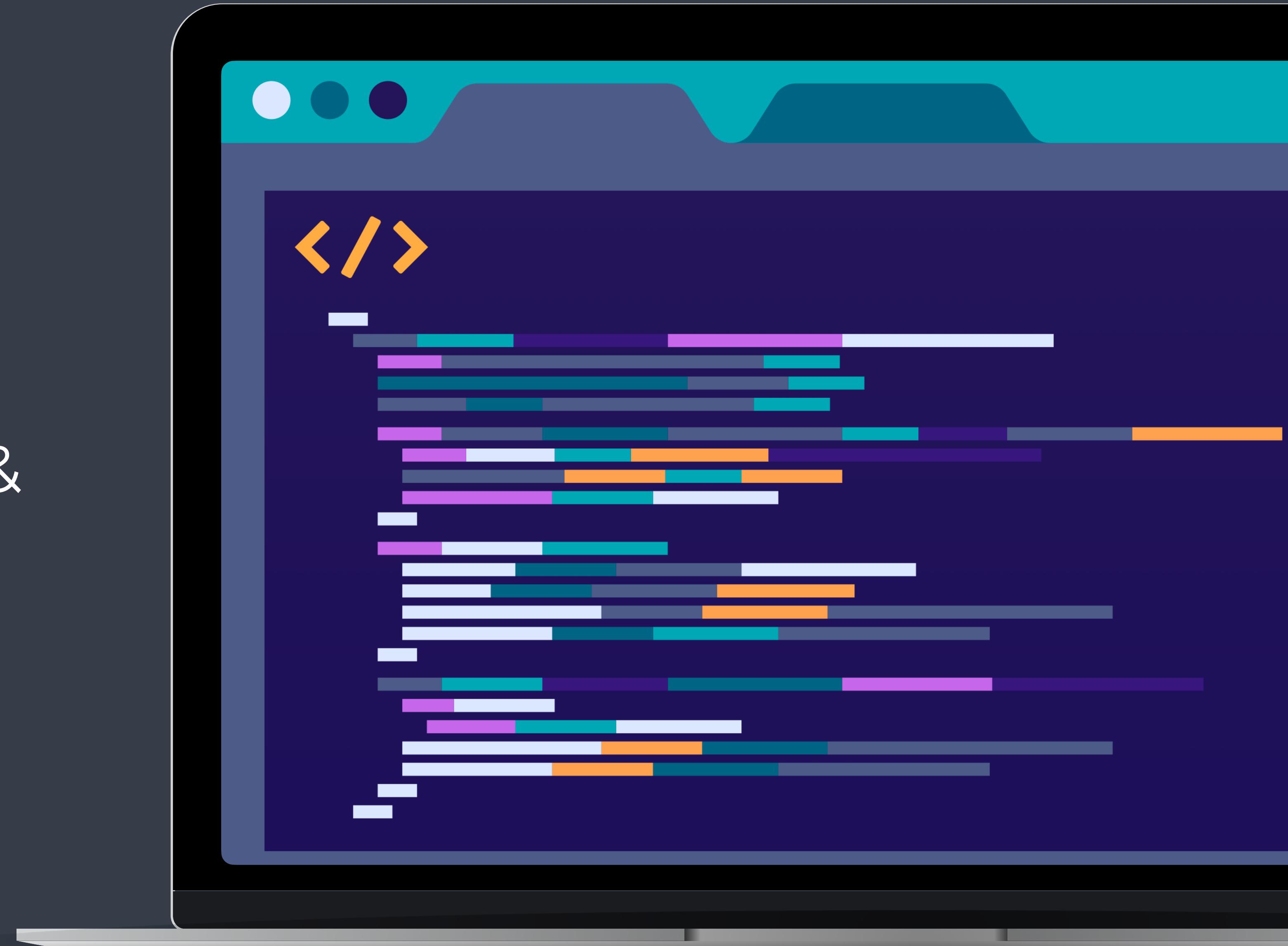
cut -[f] [file]:
Select field of each line

sed -[f] 'command' [file]:
Find and replace,
insertion or deletion

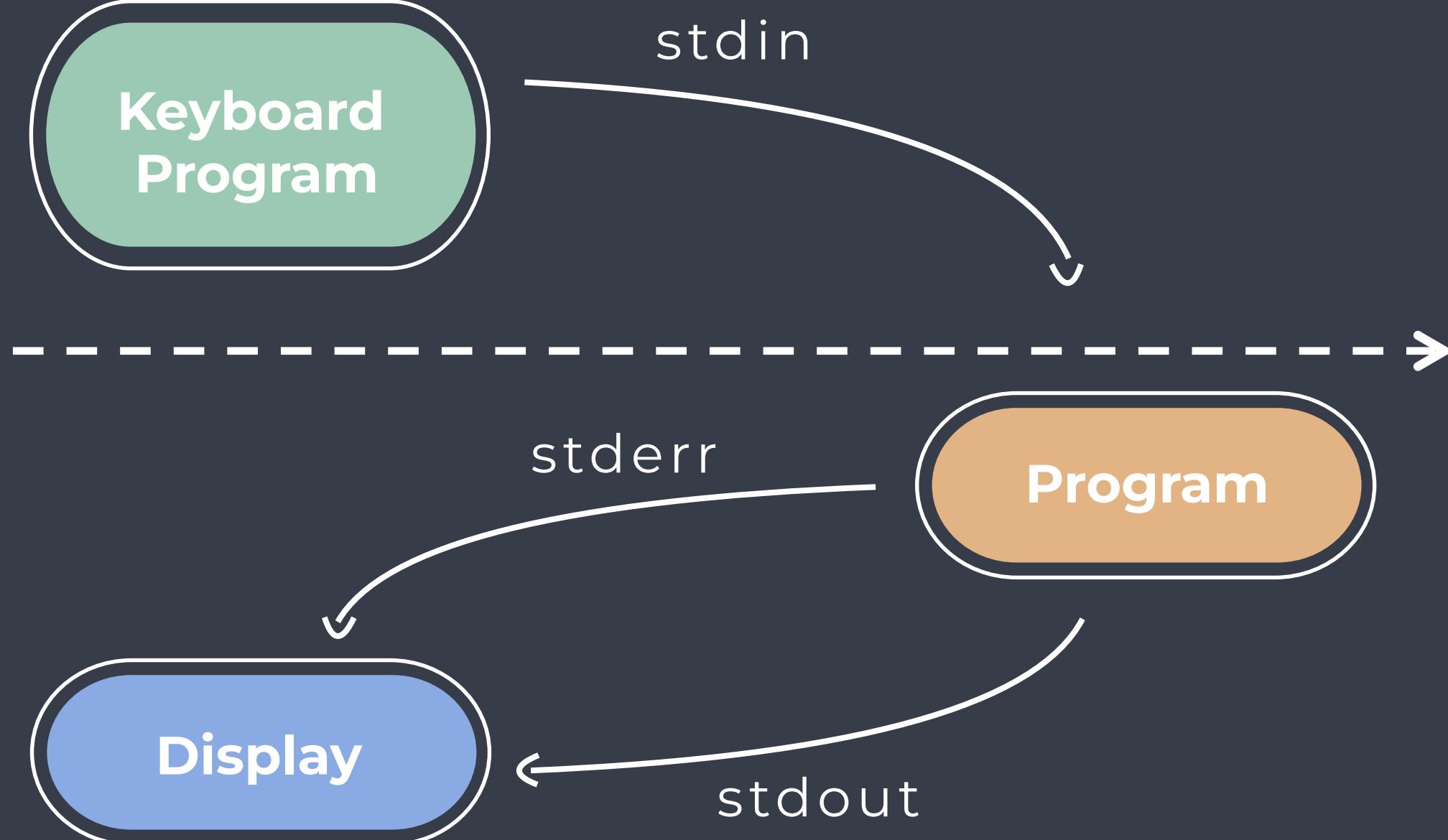
A LITTLE TERMINOLOGY



5. REDIRECTION & PIPES

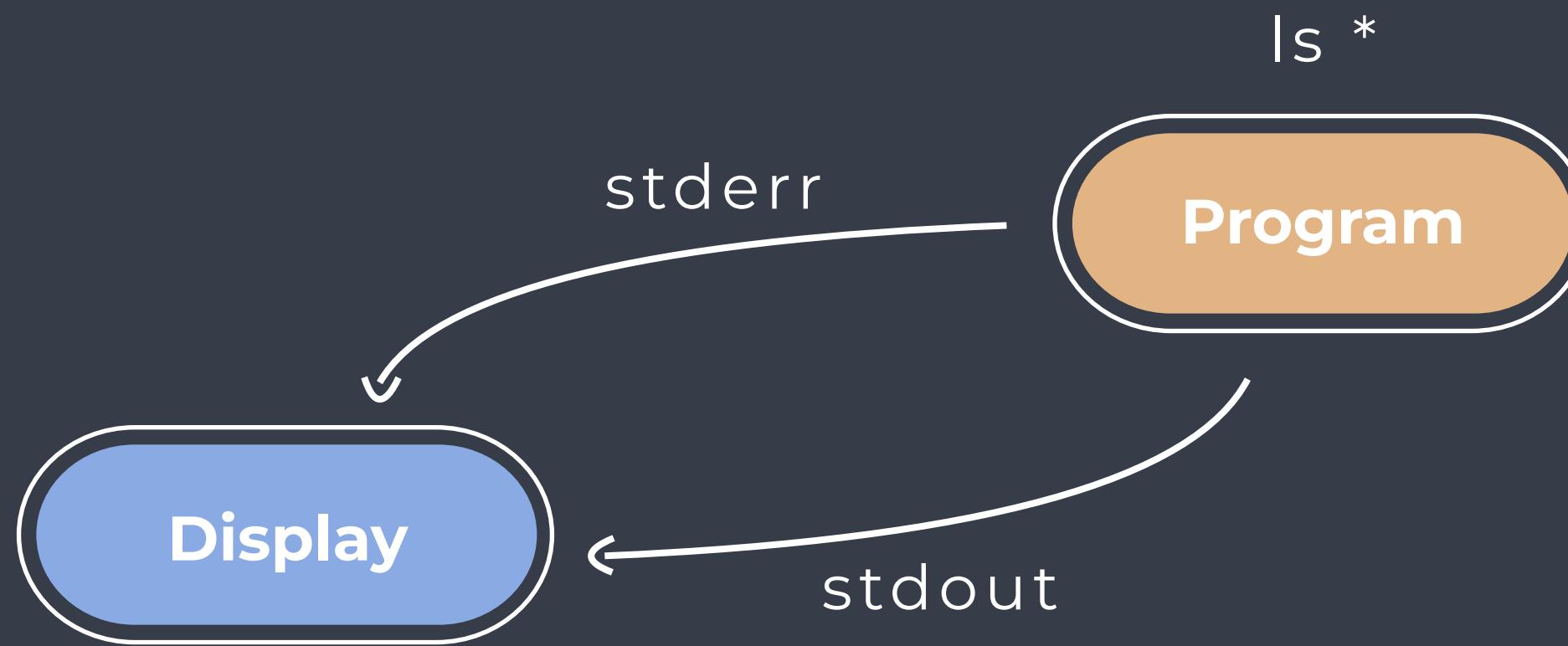


A TALE OF THREE STREAMS



- 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



Your terminal

```
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
```

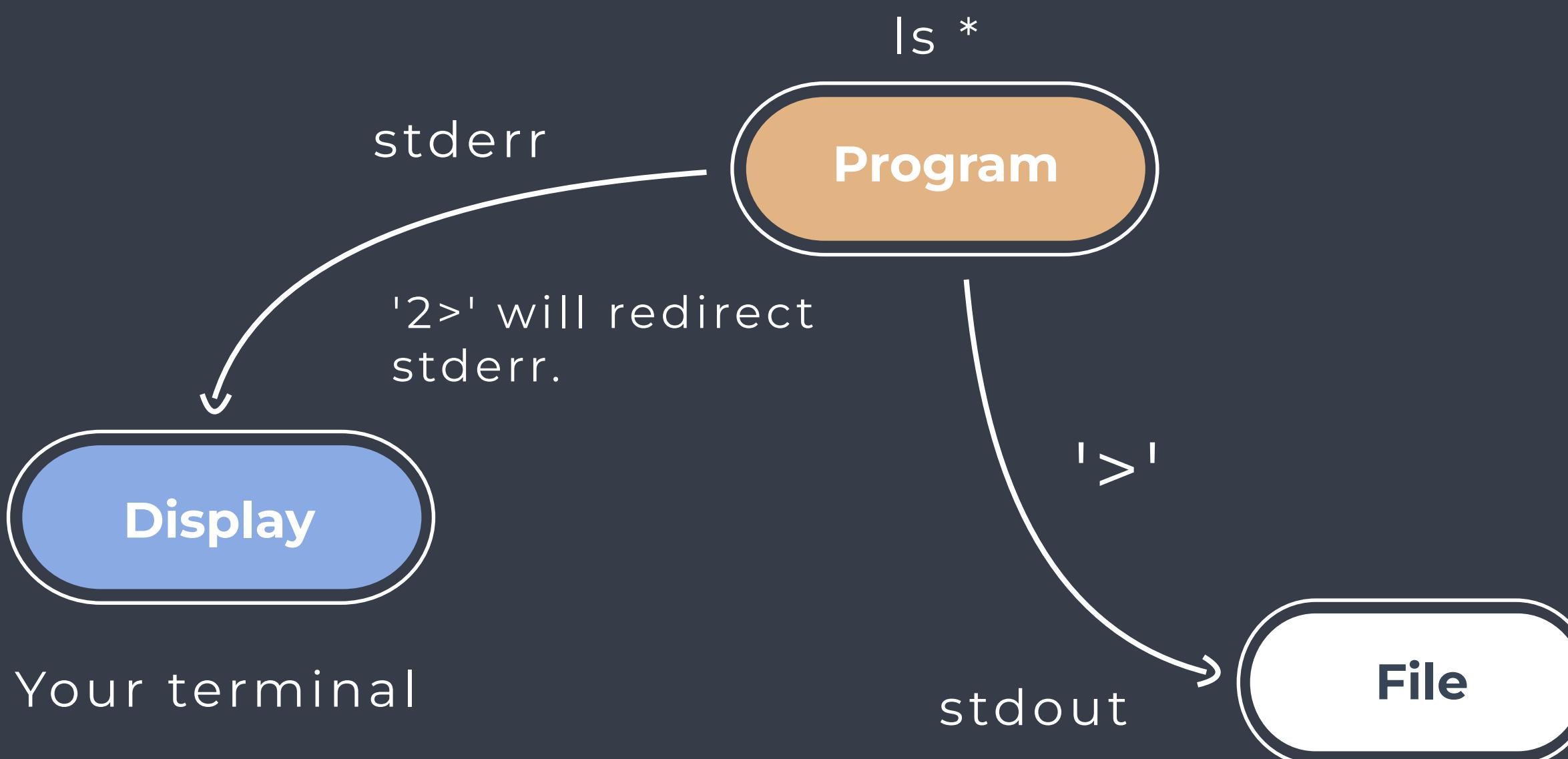
- **stdout** (standard out) - the output of commands moves along this stream

- **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
```

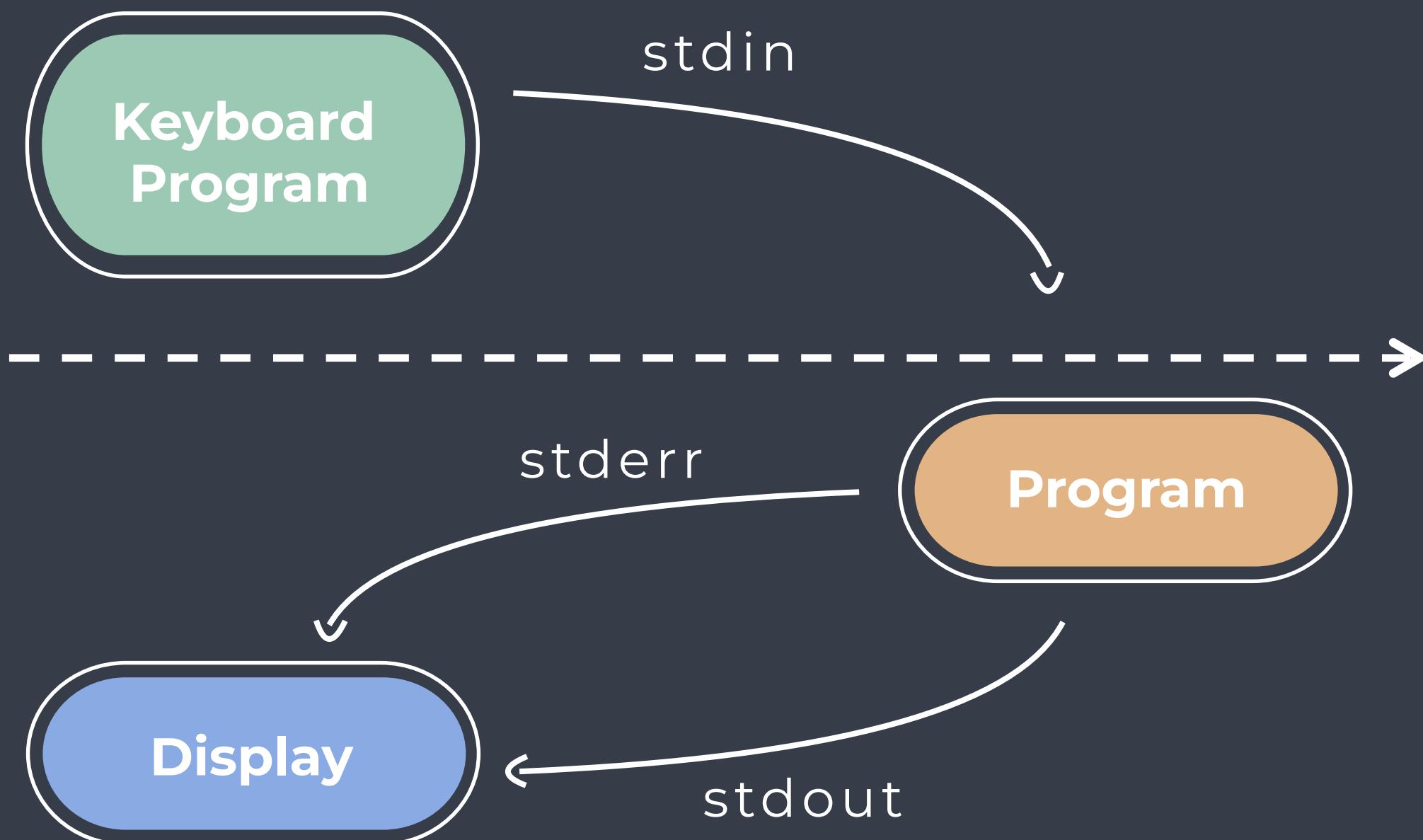
Redirect both into different files:

```
$ 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



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

sort

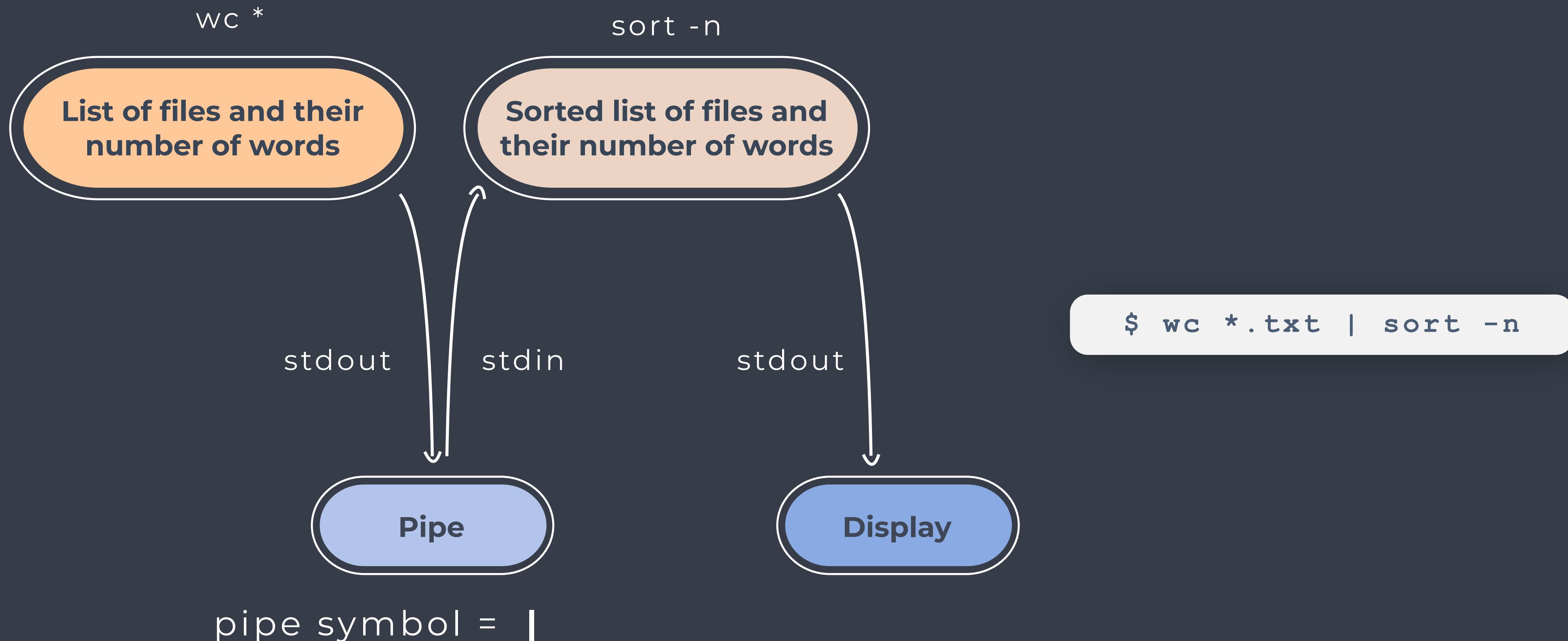
```
$ wc *.txt > file_lengths
```



```
$ sort -n file_lengths
```

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

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

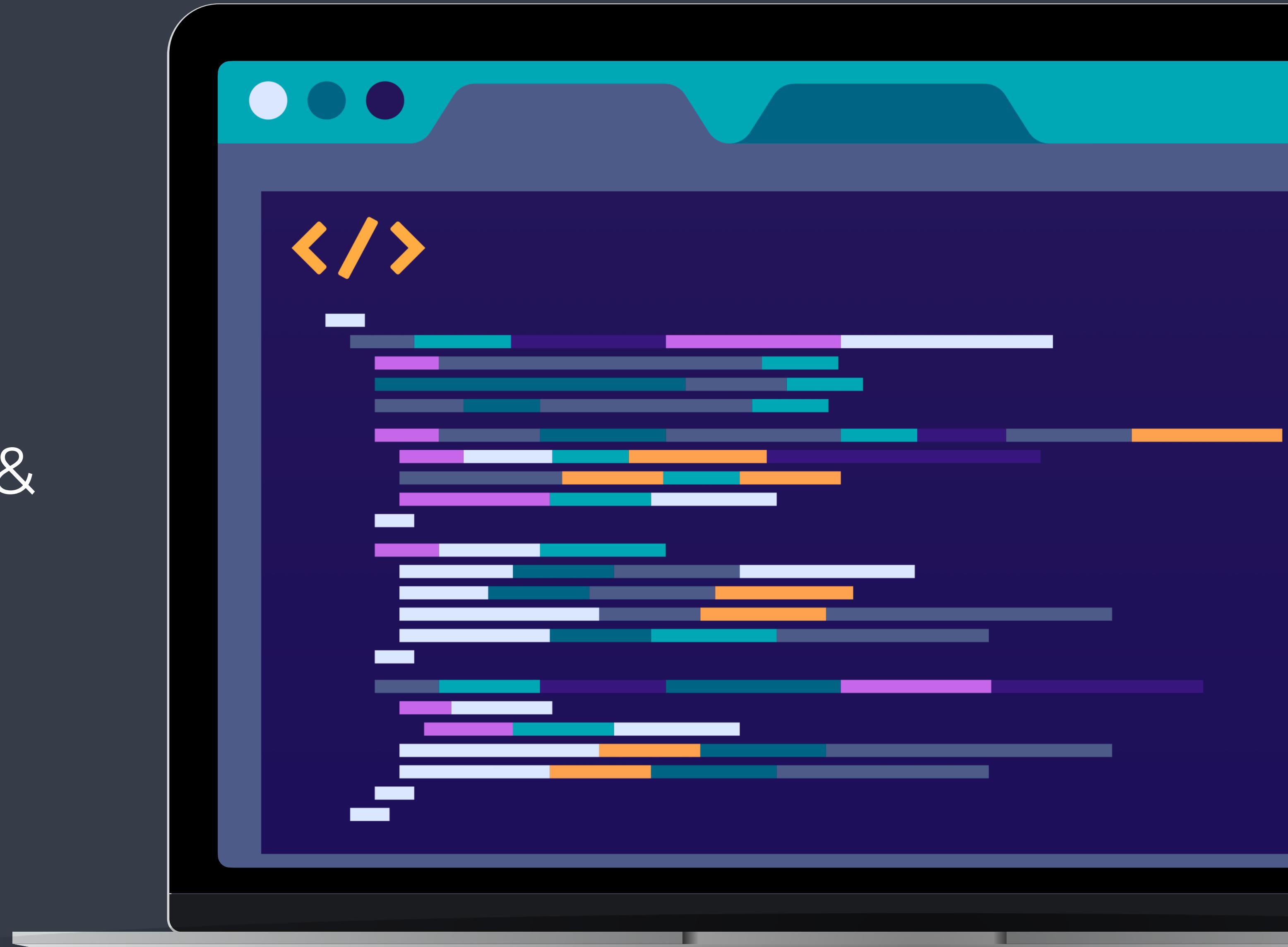
View Files

```
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

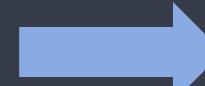
```
tar -[f] [file] # .tar files  
gzip -[f] [file] # decompress (.tar).gz files  
unzip -[f] [file] # decompress .zip files  
zless -[f][file] # view .gz file w/o decompression  
  
others: zcat, zmore, gzcat
```

6. 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
```



```
x - my_script.sh (~/Documents/Heads_center_management/courses)
Open + ~/Documents/Heads_center_management/courses/Just-... Save
my_script.sh
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 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
```

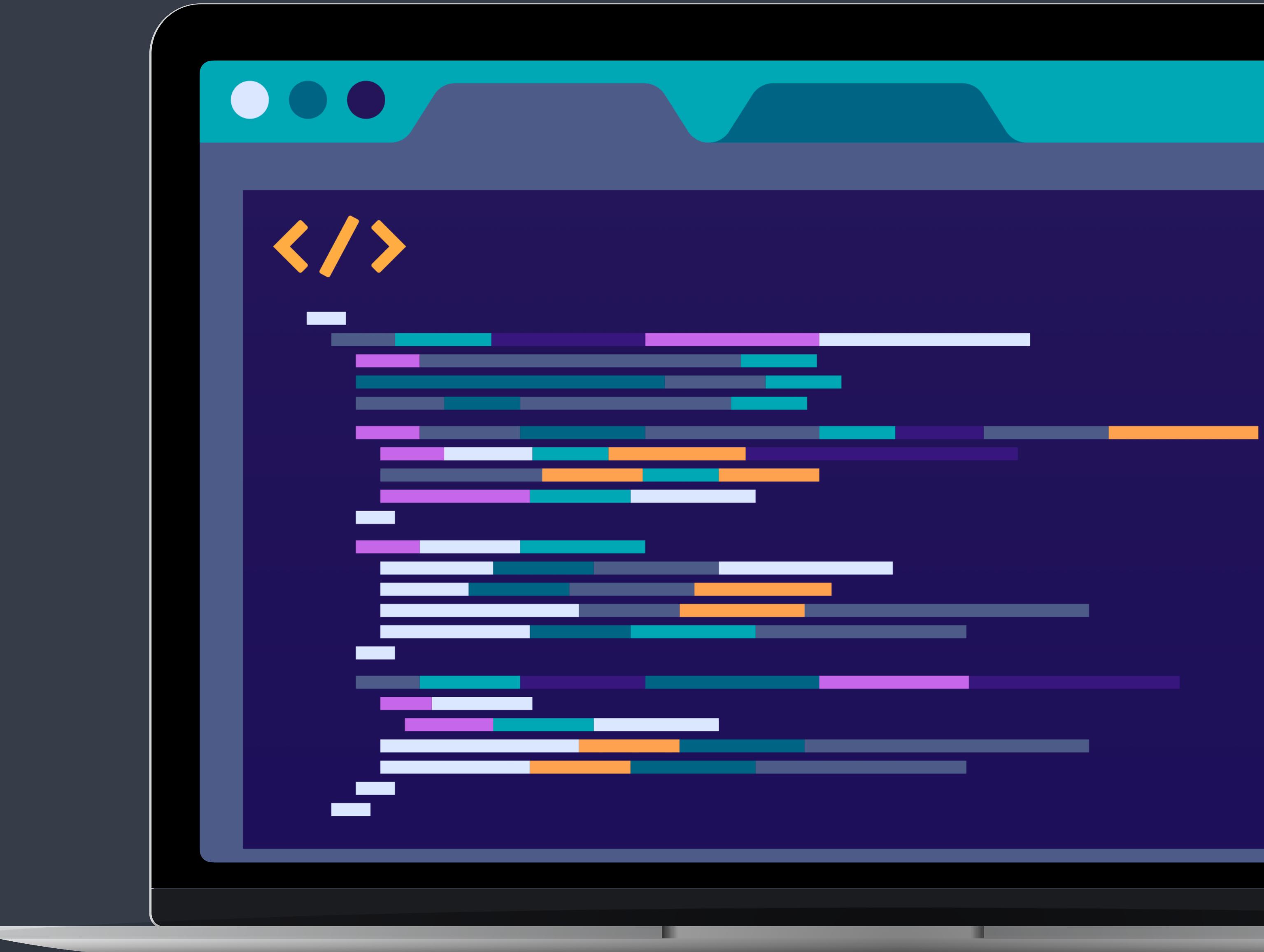
```
$ bash sort_this.sh my_textfile.txt
```

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

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

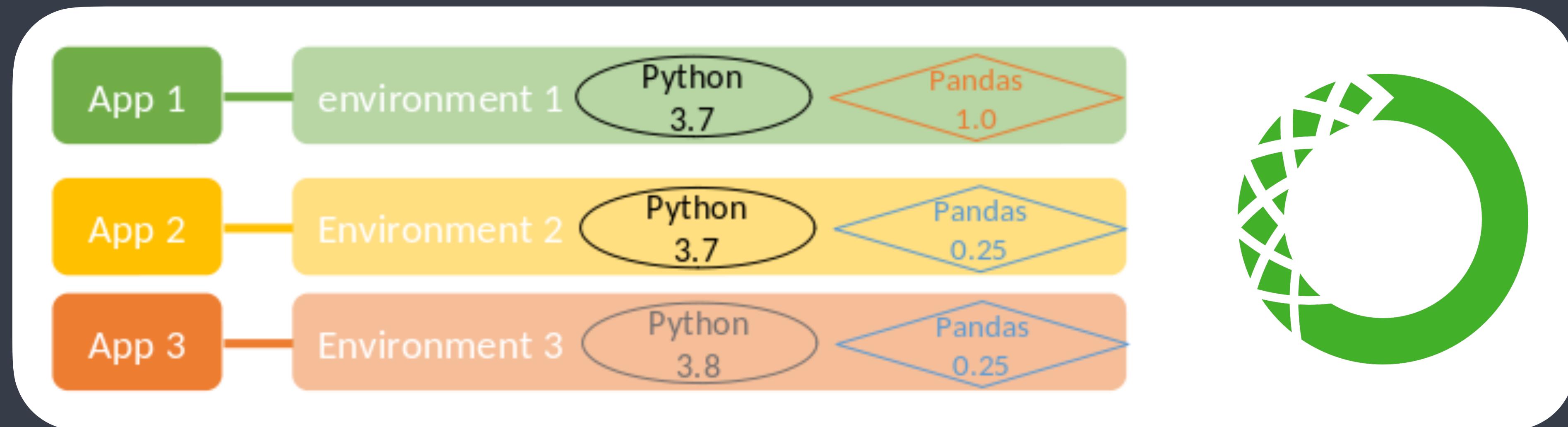
```
$ bash cut_col patients.txt 3
```

7. SOFTWARE INSTALLATION UPKEEP & MORE



CONDA

- **Conda** is an open source package -and 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.



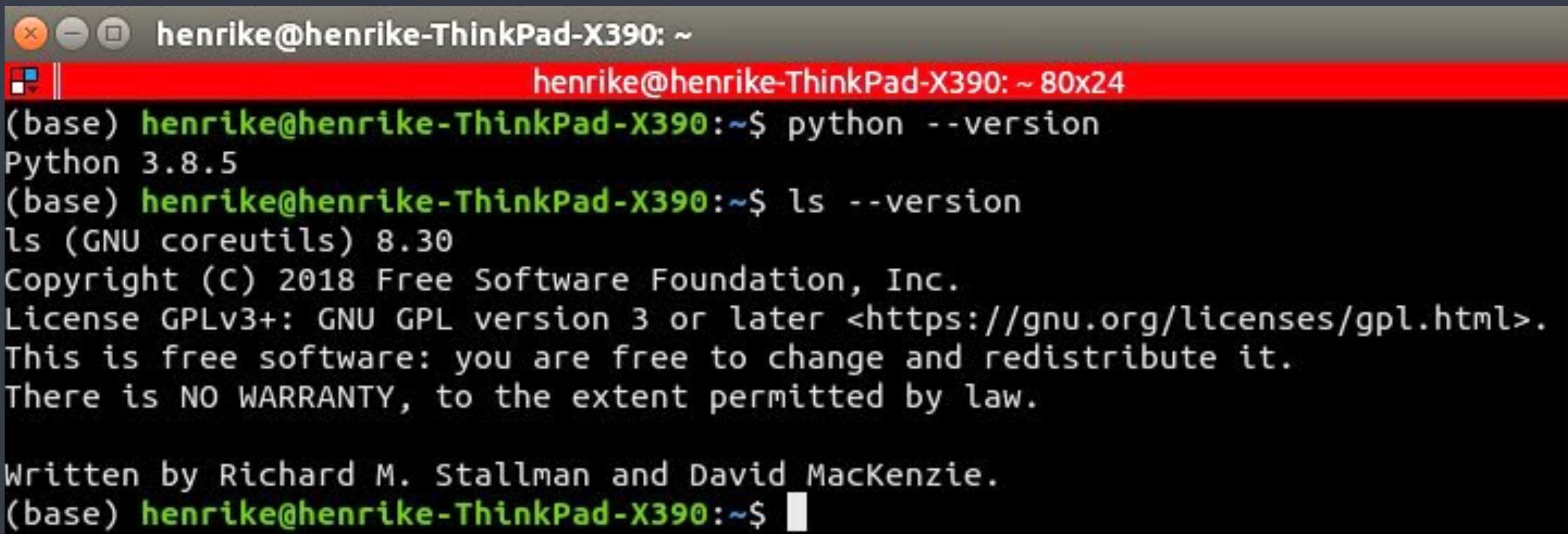
HOME BREW

APT-GET

SOFTWARE VERSION

You can generally check the version of installed software with:

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



The screenshot shows a terminal window with the following session:

```
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

Generally, you update your software with the same tool you used to install it:

- Installed with **conda**:
(updates only the currently active environment)

```
conda update -all
```

- Installed with homebrew:

```
brew upgrade python3
```

- Installed with **apt-get**:

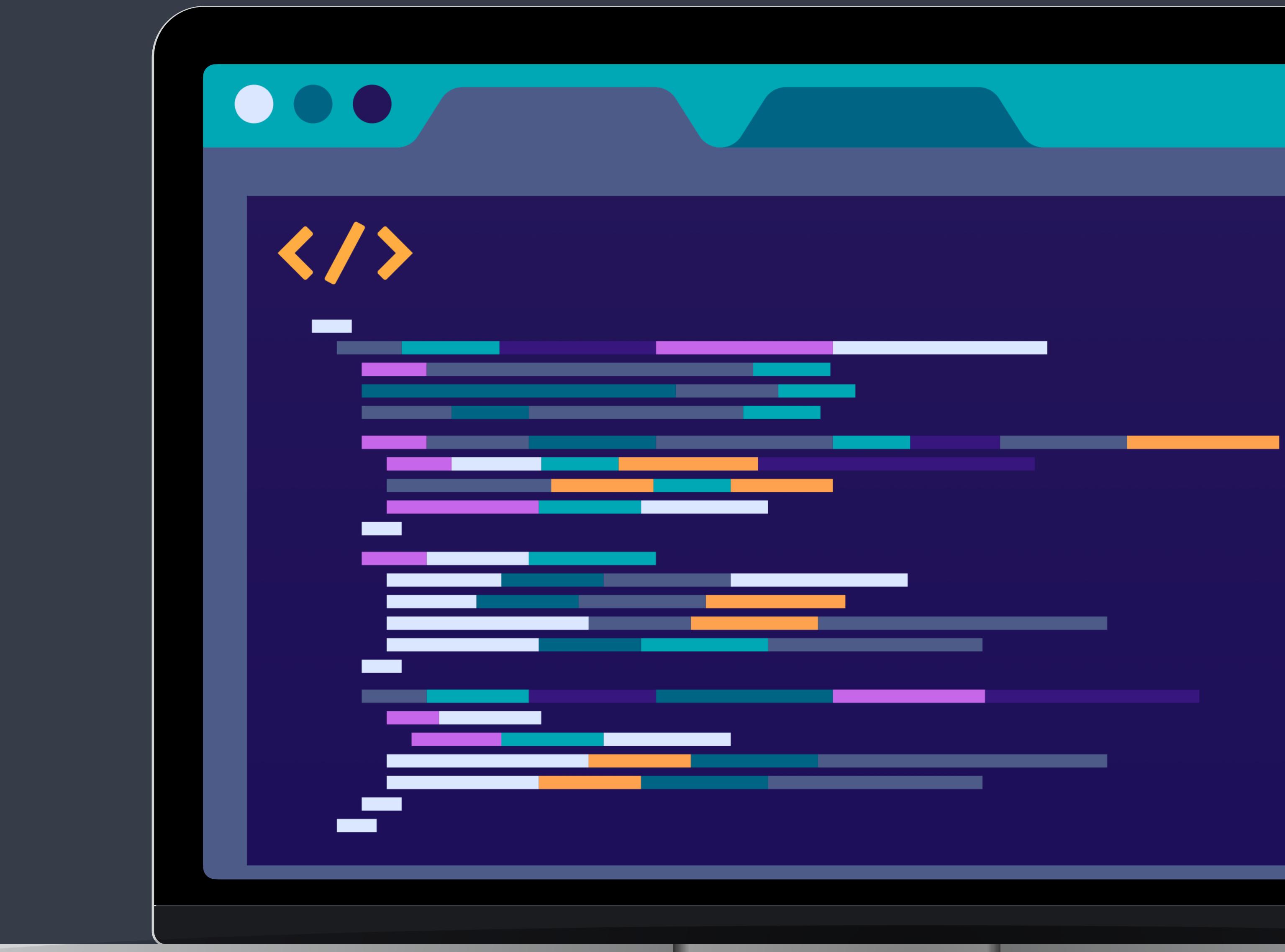
```
apt-get update && apt-get dist-upgrade
```

- Installed with **Windows App store**:

Update via Windows App store

CONFIGURATION FILES

8. WORKFLOW LANGUAGE & COMPUTE POWER



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
```

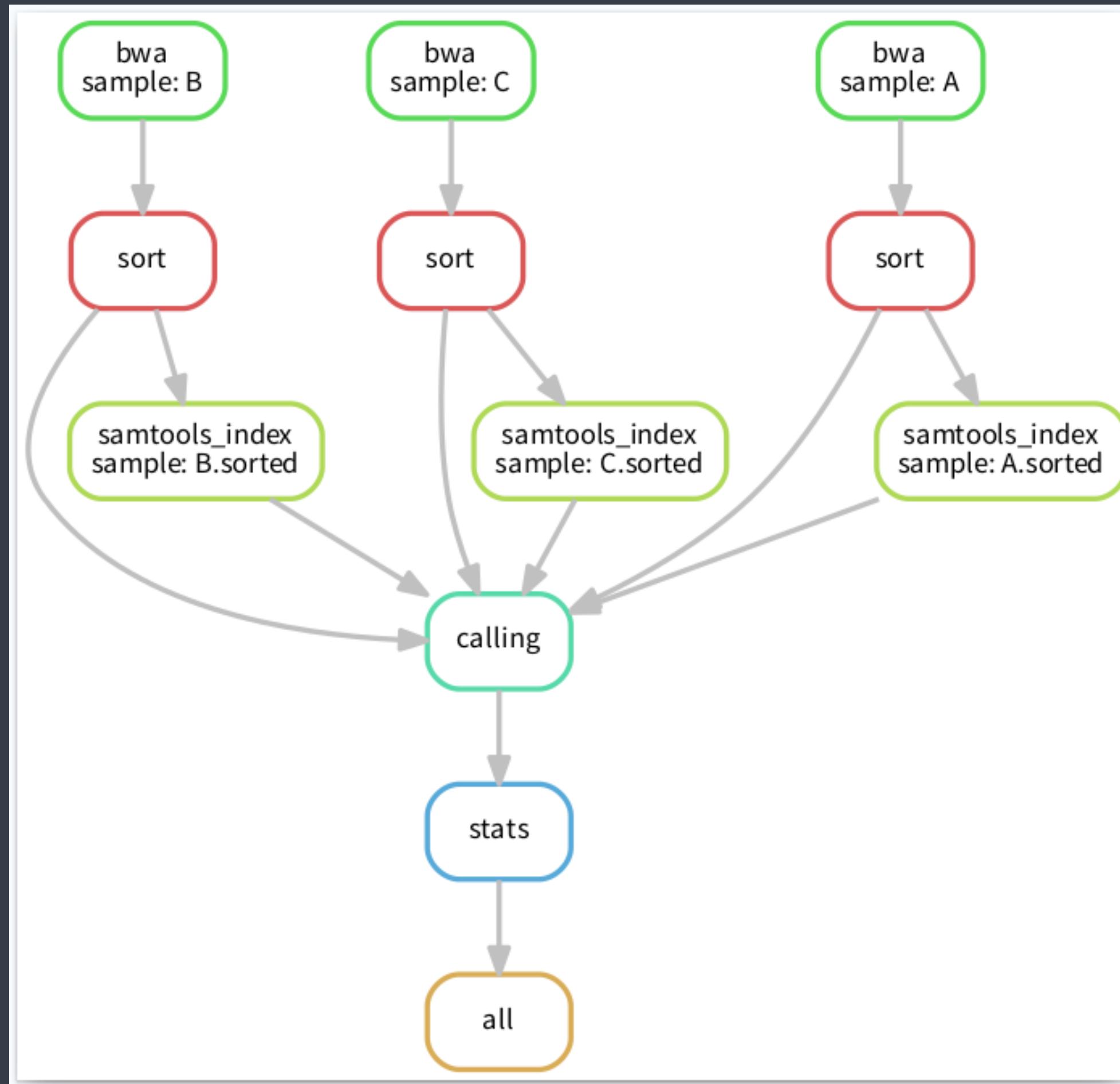
- Loops allow you to repeat the same action several times, once for each file for example.

- Many bash commands can be run on several files, i.e.

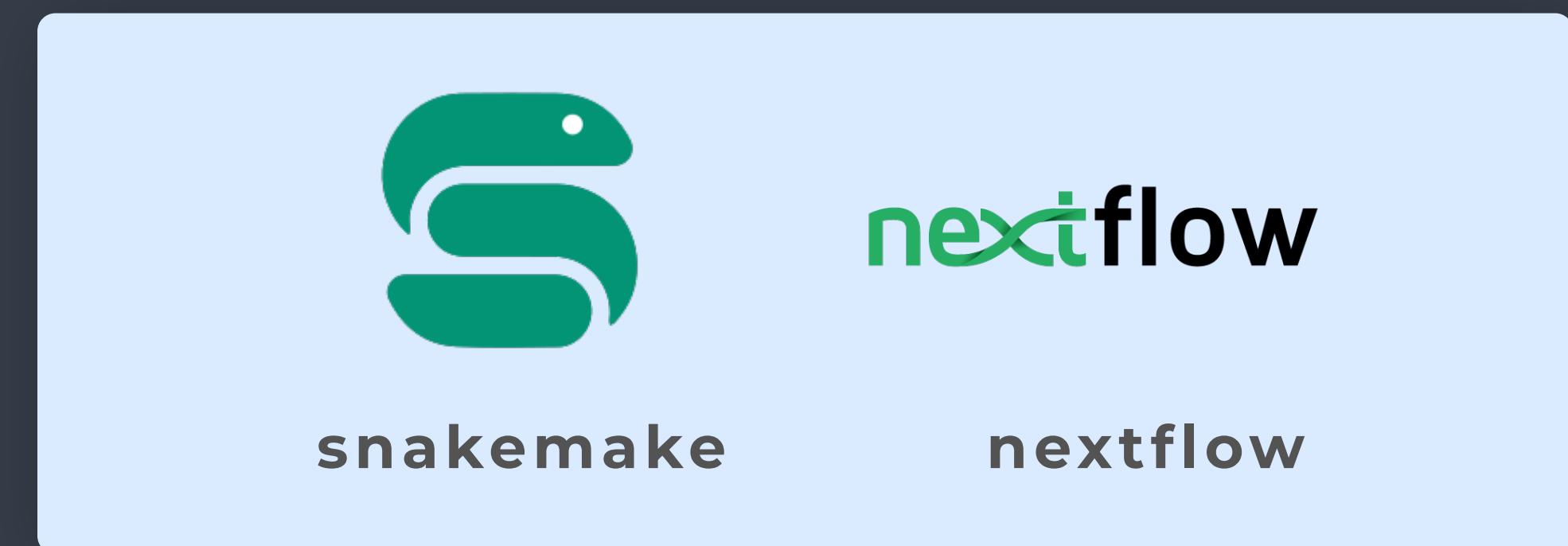
```
$ wc *.txt
```

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

WORKFLOW LANGUAGES



- **Workflow languages:** Tools to create pipelines which are more reproducible & robust than bash scripts.
- Workflow languages allow you to:
 - Control the flow of your pipeline
 - Enable you to restart at various defined points
 - Ensure previous commands have executed without error
 - Logging & Reporting of your job



HIGH PERFORMANCE COMPUTING (HPC)
