Introduction to Open & Reproducible Science (IORDS)

Michael Dayan, Data Scientist Manager

Methods & Data facility
Human Neuroscience Platform
Foundation Campus Biotech Geneva

Virtual machine info

To get an IP, please fill the form at:
`https://tinyurl.com/IORDS2021-IP-linux2`

START RDP CLIENT (as instructed in email / Slack):

- *Remote Desktop Connection* on Windows
- *Remote Desktop App* on Mac OS
- *Remmina* on Linux distributions (e.g. Ubuntu)

PLEASE CONNECT TO THE VM
→ Login: brainhacker
→ Password: brainhack!

Connect to Slack and download the exercise slides

ANY PROBLEM? Please raise your hand or ask questions on Slack: channel **`#linux`**

Connecting your:	WIFI SSID	WIFI Password
Laptop (no phones)	NIDS_course	reproduciblescience
Phone	CAMPUS_VISITORS	welcomecampus

On site support for coding exercises:

Maël 

Louis 

HNP METHODS & DATA

LECTURE OBJECTIVES

Linux lectures objectives:

- Be comfortable with Linux & the command line
- Understand the notion of kernel, shell and command line
- Understand the linux file system structure
- Know how to navigate the file system
- Know how to create, delete and interact with (text) files
- Understand file permissions and know how to modify them

Linux Part 1

- Learn how to write shell scripts
  - Variables
  - For loop

Linux Part 2

  - More control flow statements (if-else statements, etc.)
  - Define and use user-provided arguments

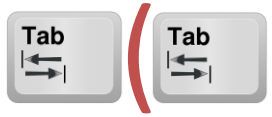
Linux Part 3 ?

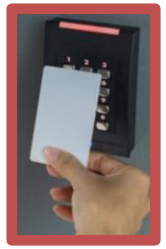
RECAP OF LINUX PART 1

back in history

forward in history

`history`

AUTOCOMPLETION MAGIC 



`whoami`

user: brainhacker

`pwd` (root)

path: / home /brainhacker /desk

> *command arg1 arg2 …*

	owner (user)	group	other
	brain hacker	neuro	
	rw-	r--	---

```
man cowsay (or tldr cowsay)
cowsay hello
cowsay -e ^^ hello
fortune | cowsay -e ^^
fortune | cowsay -e ^^ > desk/wise_cow.txt
chgrp neuro /home/brainhacker/desk/wise_cow.txt
chmod o+r /home/brainhacker/desk/wise_cow.txt
```

HNP METHODS & DATA

“>” will overwrite the file if it exists
“>>” will append to the file, i.e. add to the end of the file

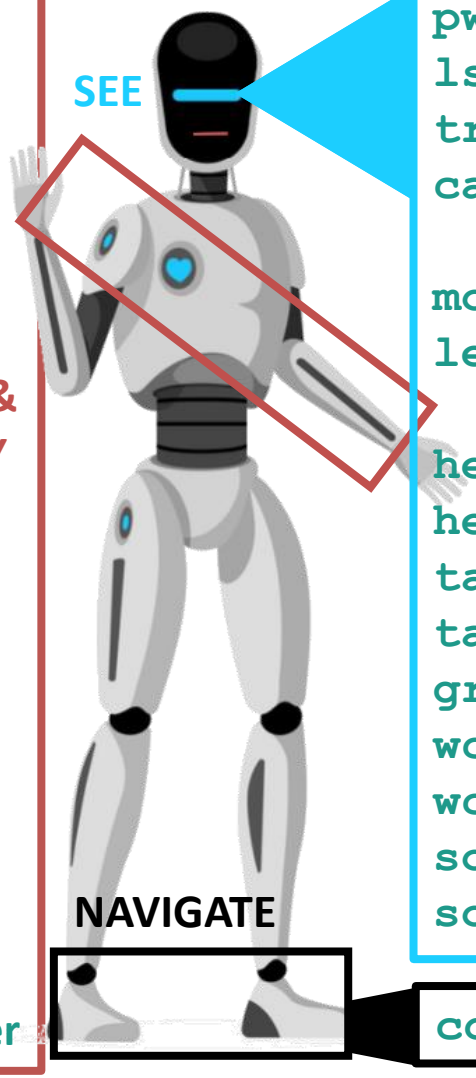
campus biotech

Fondation Campus Biotech Geneva

RECAP OF LINUX PART 1

```
mkdir DIRPATH       # make new directory
                    # (parent dir must exist)
mkdir -p PATH       # make new directory
                    # and create parents
touch FILEPATH      # create empty file
rmdir DIRPATH       # remove empty dir
rm FILEPATH         # remove file
# copy file to FILEPATH2, overwriting
cp FILEPATH1 FILEPATH2
# copy file to DIRPATH, (overwriting possible)
cp FILEPATH1 DIRPATH
# copy recursively DIRPATH1 to DIRPATH2
cp -R DIRPATH1 DIRPATH2
# move (i.e. rename) a file
mv FILEPATH1 FILEPATH2
echo MESSAGE        # print MESSAGE
fortune             # show random message
chmod [ugo][+-][rwx]  # change r, w, x permissions
                      # for user (owner), group, other
```

CREATE & DESTROY




```
pwd                 # print working directory
ls DIR              # list (files and directories) in DIR
tree -L DEPTH DIR   # tree of length DEPTH of DIR
cat FILE            # print FILE to standard output
                    # (can concatenate files)
more FILE           # view FILE (press SPACE for more)
less FILE           # similar but can also go backward
                    # (PAGE UP and PAGE DOWN keys)
head FILE           # view first 10 lines of FILE
head -n X FILE      # view first X lines
tail FILE           # view last 10 lines
tail -n X FILE      # view last X lines
grep STRING FILE    # find STRING in FILE
wc FILE             # count of FILE chars, words, lines
wc -l FILES         # count of lines in all FILES
sort FILE           # sort FILE by alphabetical order
sort -n FILE        # sort FILE by numerical value
```

```
cd DIR              # change directory
```

HNP METHODS & DATA

WILDCARDS: *, ?, [char1char2], [char1-char2], {seq1,seq2}
Example: `ls desk/[Pp]aper[0-9]*.{txt,pdf}`

campus biotech

Fondation Campus Biotech Geneva

WRITING SHELL SCRIPTS – BASICS & EDITOR

Choose a text editor

You like Word ?

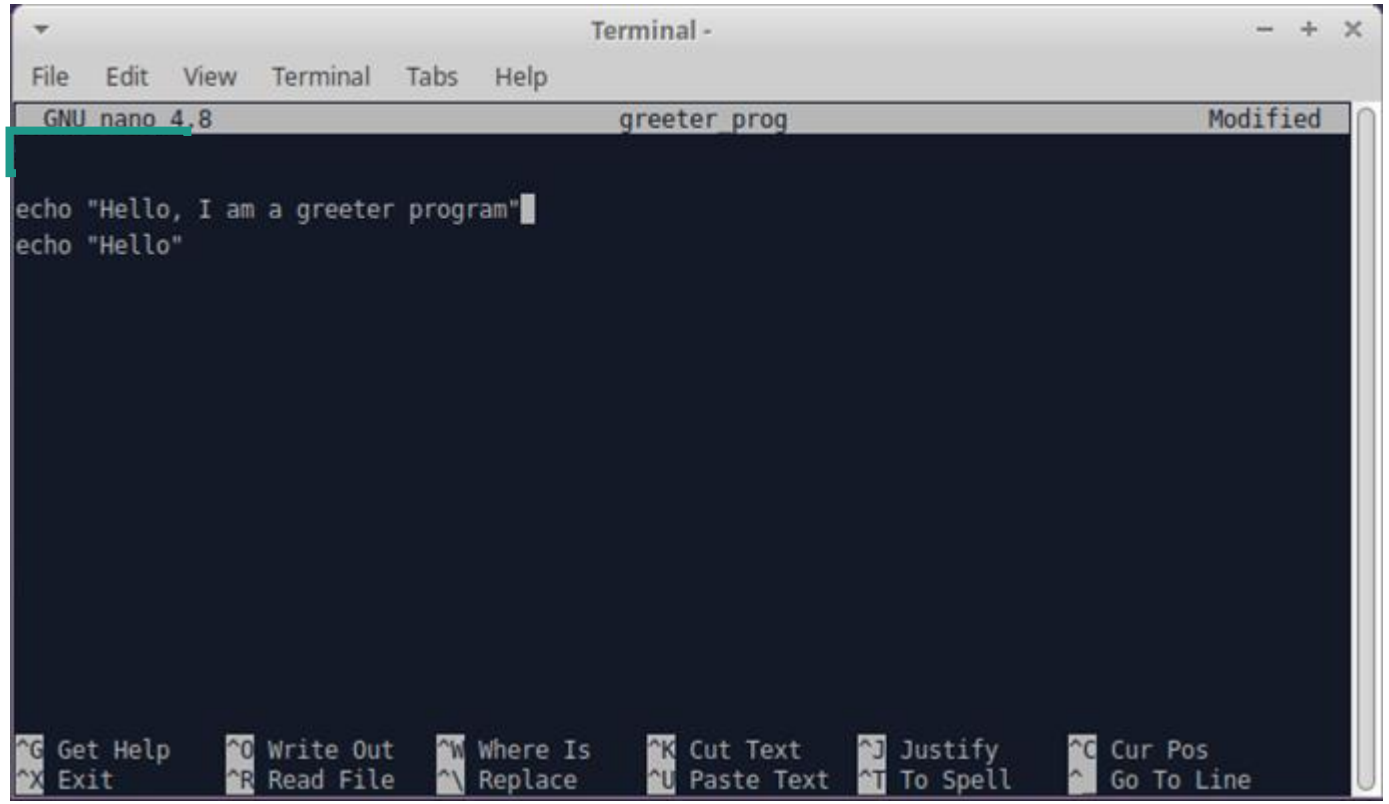
- You will probably not like `nano`

`nano greeter_prog`

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

shebang

absolute path to interpreter



aguhwwgggghhh uggguh huurh raaaaaahhgh aarrragghuuhw [1]

0100110011101010111101
1111101010010111000110
1101011101110000111010

[1] https://www.wookietranslator.com/

WRITING SHELL SCRIPTS – BASICS & EDITOR

Choose a code editor

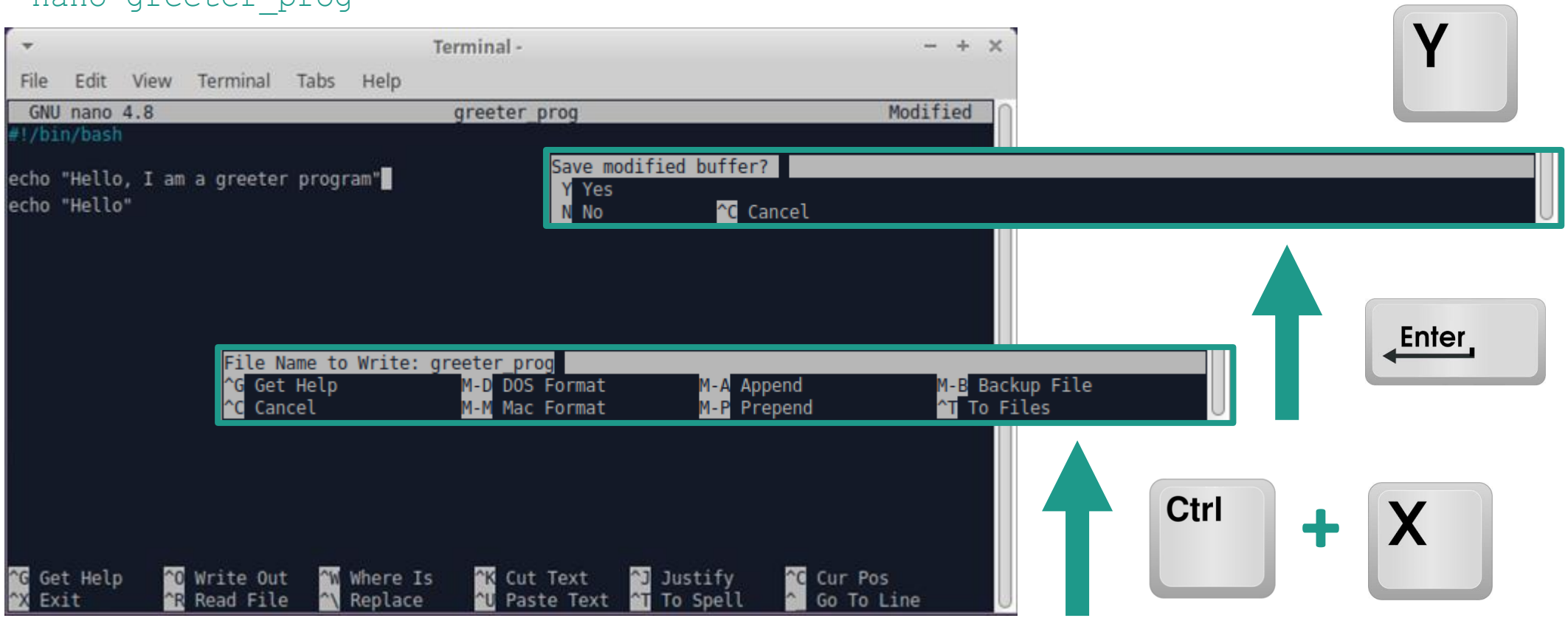
You like Word ?

- You will probably not like `nano`

```
nano greeter_prog
```

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

shebang — absolute path to interpreter



WRITING SHELL SCRIPTS – BASICS & EDITOR

Choose a code editor

You like Word ?

- You will probably not like `nano`

```
nano greeter_prog
```

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

shebang

absolute path to interpreter

Execute the code

- Make sure the program file is executable (i.e. that executable permissions are set)
- Run the program (either indicate the path to the file, or if the shell knows how to find it simply type the program name)

More on that at the next lecture

TASKS

Make sure you are in your home directory (TIP: you can use `pwd`, `ls`, `cd`)

Create new dir `hello_proj` (with `mkdir`) and go inside it (with `cd`)

Use `nano` to create a bash program which prints a greeting message with a first name, e.g. "Hi Andrew" (TIP: cf above for how to start `nano`)

Save the file with the name `hello_v1` (TIP: cf previous slide)

Make the file executable and run it (TIP: add user execute permission, then type the path to the file to run it)

(Optional) Use `nano` to edit your program so that it prints a `fortune` story after the greeting

(Optional) Use `cowsay` to have a more entertaining output

WRITING SHELL SCRIPTS – BASICS & EDITOR

Choose a code editor

You like Word ?

- You will probably not like `nano`

  `nano greeter_prog`

- But you may like Visual Studio Code (VS Code)
  1. Open your web browser at `W.X.Y.Z:8080`
  2. Then File → New File

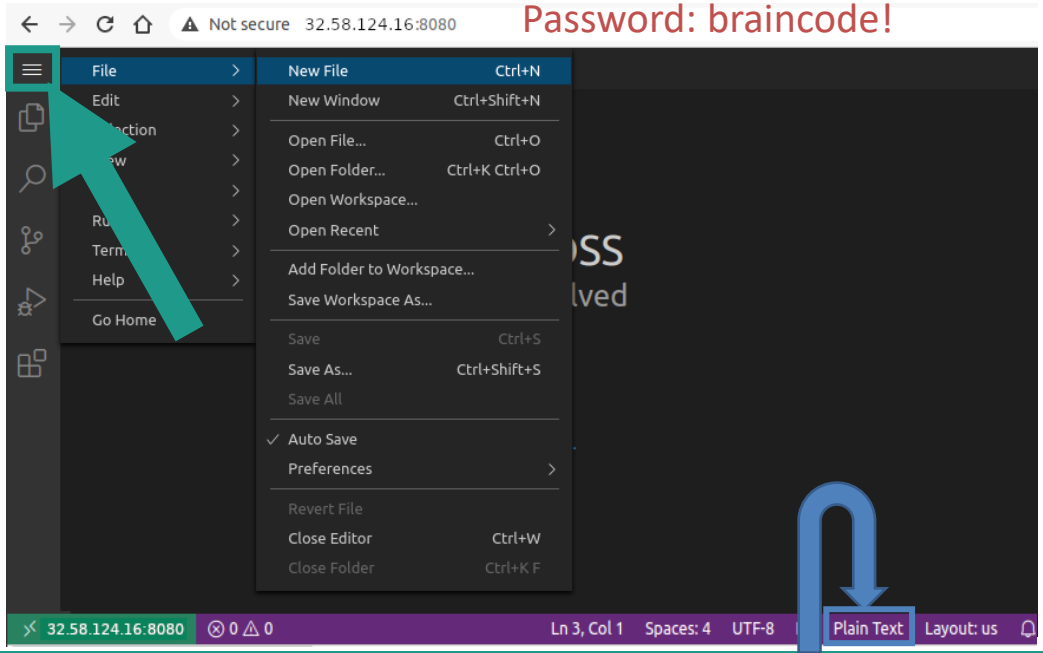
The IP you received by mail
Password: braincode!

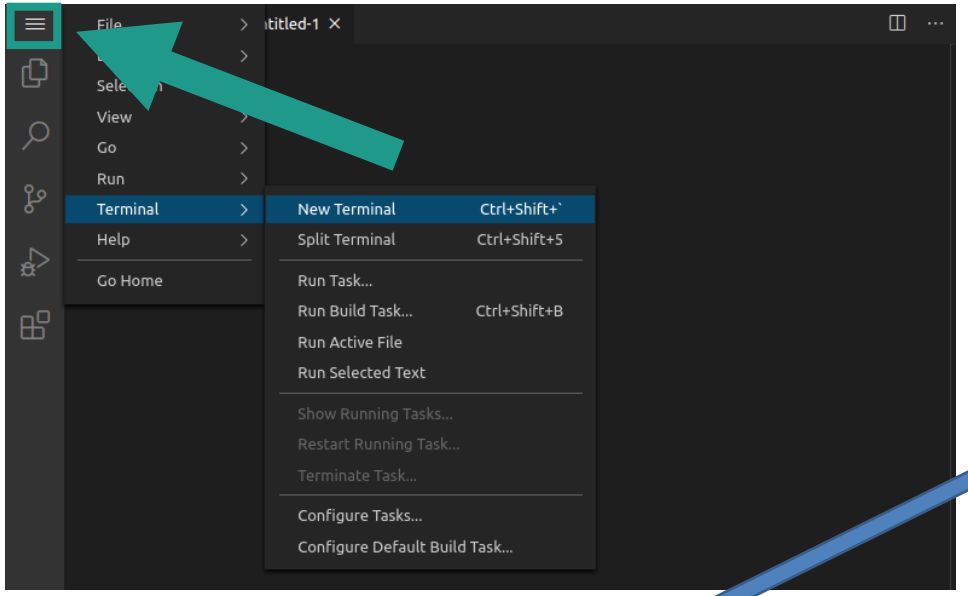
TASKS

Use VS Code to create the same program with a different first name

Save your program in `hello_proj` dir as `hello_v2` (File → Save As)
Note: click " . . " to go "up" the filesystem tree (i.e. to the parent dir)

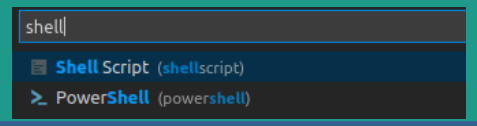
Make the file executable and run it using the terminal in VS Code





Will automatically color the text in a meaningful way in the text editor

3. Click on "Plain Text" and search for "Shell Script" in the search box popping up
4. Start "New Terminal"



HNP METHODS & DATA

campus biotech

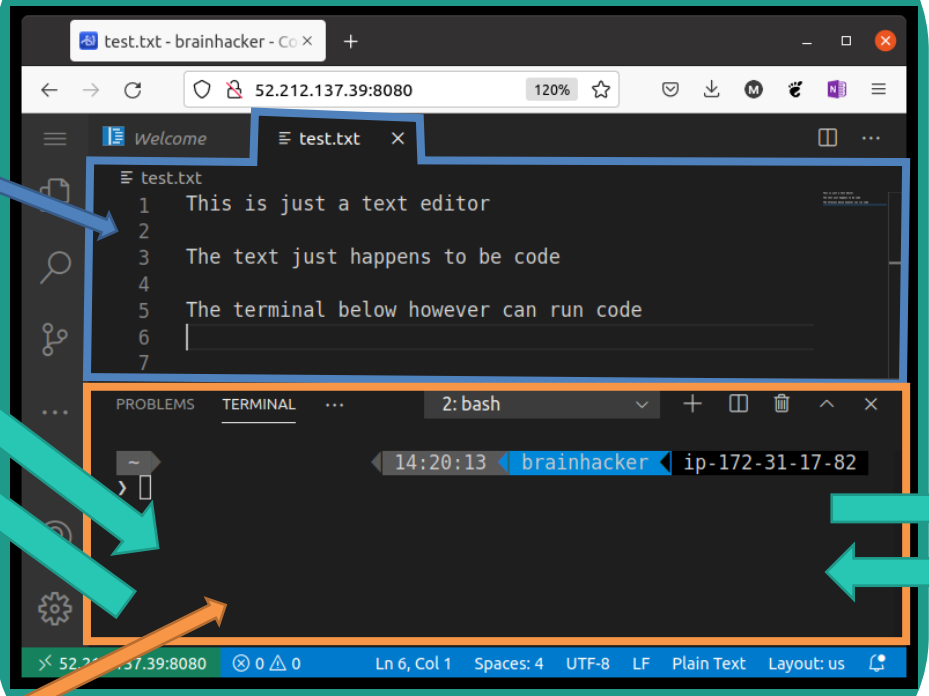
Fondation Campus Biotech Geneva

TERMINAL ON REMOTE VM VIA BROWSER

Text editor (text happens to be code)

Terminal for convenience (could have used the usual one)

USER SPACE



Windows 10 Mac OS OS

Remote Virtual Machine (VM)

BASH SHELL

LINUX USER SPACE

hello_v2

OS

Ubuntu Linux

LINUX KERNEL

WINDOWS 10, MACOS or LINUX KERNEL

HNP METHODS & DATA

campus biotech

Fondation Campus Biotech Geneva

BASIC PRINCIPLES WHEN CODING

- **Remember that a program interpreter is dumb**:
  a single mistyped character and your command or script may generate an error
  - use auto-completion as much as you can (if it doesn't auto-complete nor propose choices you may have mistyped something)
  - try copy-pasting when you can (with your mouse select in the terminal what to copy, then press middle-mouse button to paste)
  - reuse previous commands by navigating your history (up and down arrows) [also CTRL+R with fzf]

AUTOCOMPLETION MAGIC

Tab (Tab)

↑ *back in history*

↓ *forward in history*

- **Solving programming errors is common at all levels**:
  when facing an error
  - don't panic, and breathe
  - read carefully the error and think (may be simple to solve)
  - copy paste the error in a search engine (solution most likely on Stack Overflow)
    - Still need to think for yourself if the solution really applies to your case, or it is just a generic error

HNP METHODS & DATA

Use shellcheck on your script (e.g. `shellcheck hello_v3`)
Copy solution to previous task to get back on track:
`cp solutions/hello_proj/hello_v2 hello_proj/`

campus biotech

Fondation Campus Biotech Geneva

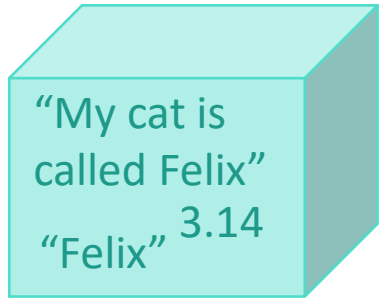
WRITING SHELL SCRIPTS - VARIABLES

Variables allow for generalization and reusability

- A variable is a reference to an object. The object can be:
  - A sequence of characters, i.e. a "string":
    - `my_var="Felix"`
    - `my_var="My cat is called Felix"`
    - `my_var="/home/felix/manipulating_humans.txt"`
  - A number (integer, float)
    - `my_var=3.14`
  - A list of objects (separator is a space → quotes are important):
    - `my_vars=("felix" "/home/felix/file.txt" 3.14)`
- In most languages to get the content of a variable – its value – you just write its name. In bash, this is too ambiguous:
  - Is `my_cat` a variable having a value (e.g. `"Felix"` ?) or simply the string `"my_cat"` ?
  - Use the `$` sign to get the variable value (print with `echo`):
    - `my_cat="Felix"`
    - `echo $my_cat`
- In bash, enclosing the variable in curly braces `{}` is safer
  - `echo $my_cat_is_black`
  - `echo ${my_cat}_is_black`

NOTES

1. A variable name starts with a character and then use characters and/or numbers but NO space and NO special characters (/, -, *, etc.) but '_' is fine
2. It is preferable to use quotes "" for strings (e.g. to avoid issues with spaces)
3. "=" is a bit misleading as it is an assignment ("<-" in R), now we must use something else to check for equality



TASKS

Edit your previous script as `hello_v3` to:

- **create a variable named `first_name` and assign it the value you chose previously (i.e. `first_name=…`)**
- **replace the name you previously wrote explicitly with the content of the variable (i.e. write `${first_name}` where appropriate)**
- **run your script**

WRITING SHELL SCRIPTS - VARIABLES

When the variable is a list of objects: arrays

- It is common to work with a list of things (files, subject names, etc.) and arrays are typically used to represent these data:
  - `people=("Luke" "Leila" "R2D2")`
  - `status=("human" "human" "robot")`
  - `my_files=(/home/brainhacker/inflammation_data/*/*.csv)`

This is called filename expansion or "globbing"

- You can access elements of the array with an index (integer):
  - `echo "${people[0]}"` (use quotes to avoid globbing)
  - `echo "${people[1]}"`
- To get all elements as an array, use `[@]`:
  - `more_people=("${people[@]}" "another_one")`
- To get all elements as a single string, use `[*]`:
  - `echo "${people[*]}"`
- To count the total number of elements, use `#` on the array ([@]):
  - `echo "I have ${#my_files[@]} files"`

Coding style

- Properly writing code is important for code reusability
  - Use meaningful variable names (not `a, b, c`)
  - Document the code by writing comments
    - In Bash comments start with a pound (#) sign

```
#!/bin/bash
#
# This program greets a list of people
#

# List of Star Wars crew
sw_crew=("Luke" "Han Solo" "R2D2")
…
```

TASKS

Edit your previous script as `hello_v4` to:

- **define an array of names instead of a single name**
- **print to the screen all the names in your array instead of the single name you previously had (use `[*]`)**
- **add comments to your program (use #)**
- **run your script**

(Optional) Add a line printing the number of elements in your array

WRITING SHELL SCRIPTS – CONTROL FLOWS

A script is a list of instructions, like a cooking recipe

```
#!/bin/bash
#
# This program makes pancakes

favorite_alcohol = <user_defined>
gluten_intolerant = <user_defined>

take bowl
```

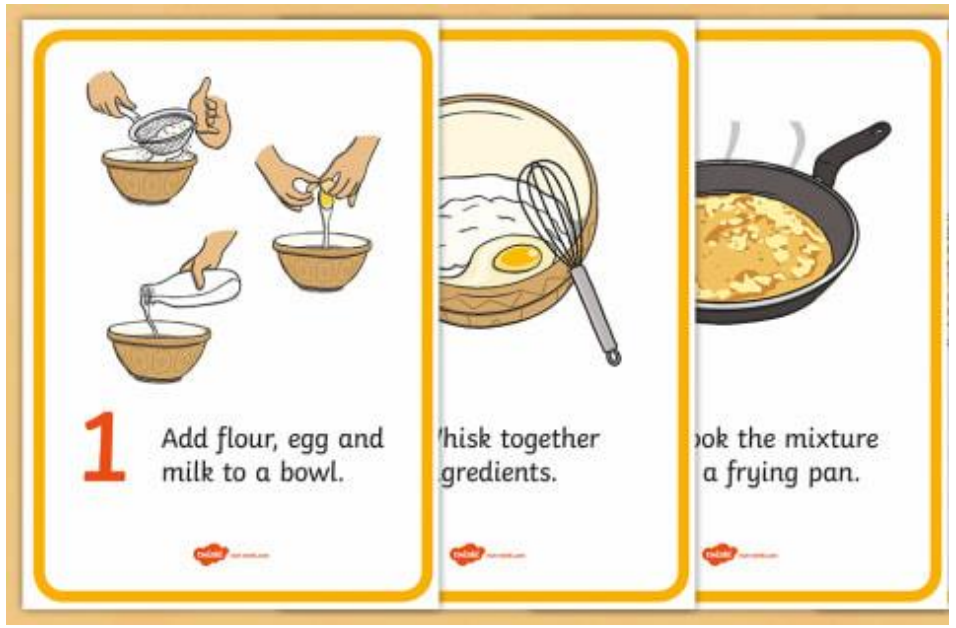
```
break and add egg
break and add egg
break and add egg
break and add egg
break and add egg
```

Laborious repetitive task

case-dependent instructions ?

```
case favorite_alcohol:
  rum: add 5 CL rum
  beer: add half beer glass
```

```
while bowl is not empty:
    pour one ladle on pan
    while not golden:
        cook
    serve in plate
```




```
if gluten_intolerant :
  add wheat flour
else:
  add rice flour
```

if <condition> else ?

```
while the mix is not smooth:
  add milk
```

while <condition> ?

- FOR LOOP
- IF-ELSE STATEMENT
- WHILE LOOP
- CASE / SWITCH STATEMENT

WRITING SHELL SCRIPTS – FOR LOOP

For loops are essential for repetitive tasks

- Loop on objects in an array

```
#!/bin/bash
for item in ARRAY_ITEMS; do
   command 1
   command 2
   ..
done
```

e.g. `${sw_crew[@]}`

- Loop on an integer index ranging from 0 to N

Initialization — Continuation condition — Update after each iteration

```
for (( I=0; I<=N; I+=1 )); do
   command 1
   command 2
done
```

TASKS

Edit your previous script to print to the screen each name in turn using loop on objects, save it as `hello_v5` (TIP: use `[@]`)

Edit your previous script as `hello_v6`, using loop on index to:

- **define an array `status` having only "robot" and "human" items, to describe the status of each person in the names array e.g. `status=("human" "robot" "human")`**
- **print to the screen each name AND status in turn (TIP: use a single for loop and the same index for both arrays)**

Example

```
crew_size=${#sw_crew[@]}
for (( I=0; I<=crew_size; I+=1 )); do
  member=${crew_size[I]}
  echo "Member: ${member}"
done
```

Example shortened

```
for (( I=0; I<=${#sw_crew[@]}; I+=1 )); do
  echo "Member: ${crew_size[I]}"
done
```

TIP

Use `shellcheck` to check for problems in your script:

e.g. `shellcheck hello_v5`

WRITING SHELL SCRIPTS – IF ELSE STATEMENT

If and if-else statements to check a condition

➢ If statement

```
#!/bin/bash
if [[ X BIN_OP Y ]]; then
    commands
fi
```

condition which is true or false

BIN_OP is a binary operator (comparison between two elements)

```
if [[ X BIN_OP Y ]]; then
    set of commands
else
    other set of commands
fi
```

optional

```
if [[ X BIN_OP Y ]]; then
    set of commands
elif [[ X BIN_OP Y ]]; then
    other set of commands
fi
```

Notes

1. You can have as many `elif` as desired
2. An optional "`else`" can be added at the end

➢ Binary operators

If X and Y are strings

BIN_OP	Condition is true if
==	strings are equal, or X matches regular expression Y
!=	strings are different, or X does not match regular expression Y

If X and Y are numbers

BIN_OP	Nature of test
-gt	>
-ge	≥
-lt	<
-le	≤
-eq	=
-ne	≠

Example 1

```
if [[ $status == "robot" ]]; then
    …
elif [[ $status == "human" ]]; then
    …
fi
```

Example 2

```
if [[ $crew_size -gt 2 ]]; then
```

WRITING SHELL SCRIPTS – IF ELSE STATEMENT

If and if-else statements allow to check a condition

- If statement

```
#!/bin/bash
if [[ X BIN_OP Y ]]; then
    commands
fi
```

condition which is true or false

BIN_OP is a binary operator (comparison between two elements)

- Binary operators

If X and Y are strings

BIN_OP	Nature of test
==	strings are equal, or X matches regular expression Y
!=	strings are different, or X does not match regular expression Y

If X and Y are numbers

BIN_OP	Nature of test
-gt	>
-ge	≥
-lt	<
-le	≤
-eq	=
-ne	≠

Example 1

```
if [[ $category == "robot" ]]; then
```

Example 2

```
if [[ $crew_size -gt 2 ]]; then
```

TASKS

Using an if or if/else statement, create a new script `hello_v7` by keeping the previous status array defined in `hello_v6` and editing the loop to print out the subject's name AND her humanoid status in this way:

```
Luke is a humanoid
R2D2 is not a humanoid
```

TIP: apply the if/else condition to the variable you defined for status (cf example 1 on previous slide)

WRITING SHELL SCRIPTS – IF ELSE STATEMENT

If and if-else statements allow to check a condition

- If statement with a unary operator (look at a single variable)

```
if [[ UNI_OP X ]]; then
    commands
fi
```

UNI_OP	Condition is true if
-f	X exists and is a file
-d	X exists and is a directory
-z	X is an empty string

- Use ! to check for negation (e.g. not an existing file)

Example 1

```
my_file=/home/brainhacker/book.txt
if [[ -f ${my_file} ]]; then
  echo "${my_file} is an existing file"
fi
```

Example 2

```
output_dir=/home/brainhacker/outputs
if [[ ! -d ${output_dir} ]]; then
  echo "${output_dir} does not exist"
fi
```

- Command expansion (useful bash feature)

```
my_output=$(command)
```

Example: `n_lines=$(cat ${my_file} | wc -l)`

TASK

Using an `if else` statement, keep the previous status array defined in `hello_v6`, and edit the loop to print out the subject's name and her humanoid status such as "Luke is humanoid", "`R2D2` is not humanoid", etc. Save as `hello_v7`.
TIP: apply the if/else condition to the variable you defined for status

HOMEWORK

Create a new file called `desk_search` (and make it executable as described in "Using the code editor")

Create an array with the content of the directory `/home/brainhacker/desk` (using * wildcard), and for each item print if it is a file or a directory (TIP: use a unary operator)

(Optional) Use the `basename` command and the command expansion feature to also print each filename

WRITING SHELL SCRIPTS – WHILE LOOP

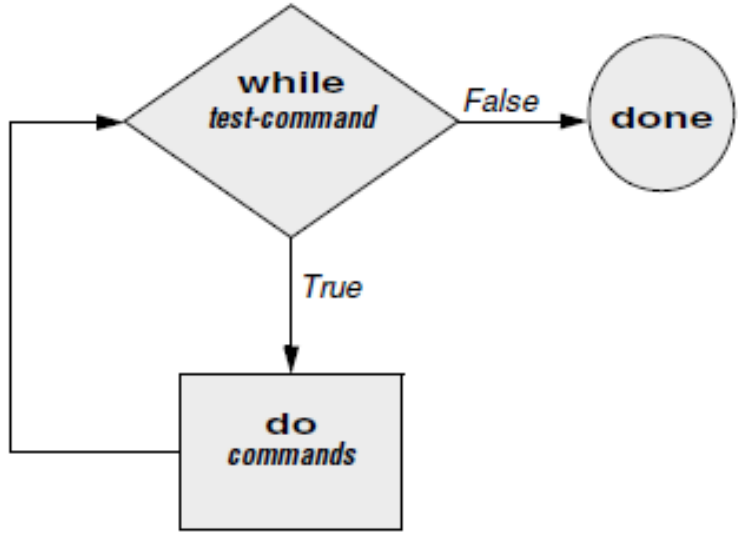
While loop (repeat until condition is no longer met)

```
#!/bin/bash
while [[ CONDITION ]]; do
    commands
done
```

condition checked at each iteration (can be true or false)

Commands run while the condition is true

→ **Make sure the commands eventually change the condition** otherwise it is an infinite loop!



Example

```
sw_crew=("Luke" "Han Solo" "R2D2")
crew_size=${#sw_crew[@]}
K=0
while [[ $K -lt $crew_size ]]; do
  echo "Crew member: ${sw_crew[K]}"
  K=$(( K + 1 ))
done
```

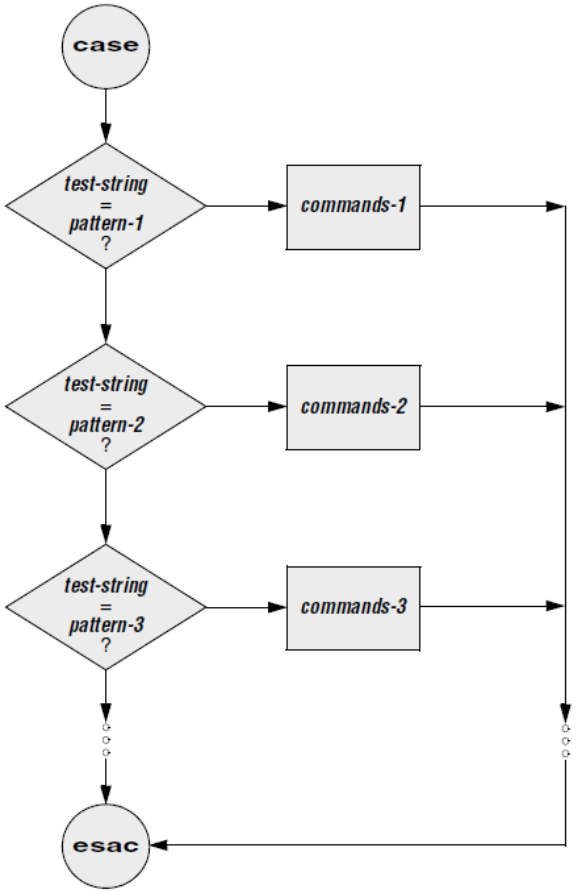
KEEP
CALM
AND
PRESS
CTRL C

WRITING SHELL SCRIPTS – CASE/SWITCH STATEMENTS

Case / switch statement (to match various options)

```
#!/bin/bash

case $VAR in
  pattern_X )
    commands
    ;;
  pattern_Y )
    commands
    ;;
  pattern_Z )
    commands
    ;;
  *)
    commands
    ;;
esac
```



Example (adapted from [1])

```
echo -n "Do you agree with this? [yes or no]: "
read yes_or_no # yes_or_no now has the answer of the user

case $yes_or_no in
  [yY][Ee][Ss] )
    echo "You agreed"
    ;;
  [nN][Oo] )
    echo "You did not agree"
    ;;
  *)
    echo "Invalid input. Please answer yes or no."
    ;;
esac
```

WRITING SHELL SCRIPTS – SPECIAL VARIABLES / FUNCTION

Special variables within the script

Variable	description
$0	The command the script was called with
$1	The first argument of the script
$2	The second argument of the script
…	
$#	The number of arguments

Function within a script

```
my_function() {
  commands using or not $1, $2, etc.
}
```

Example 1

```
impersonal_greeter() {
  echo "Hi!"
}
```

```
impersonal_greeter → Hi!
impersonal_greeter "Bob" → Hi!
```

Example 2

```
personal_greeter() {
  echo "Hi $1!"
}
```

```
personal_greeter → Hi !
personal_greeter "Bob" → Hi Bob!
```

Special variable after running the script

Variable	description
$?	The exit status code (0: good, ≠0: problem)

Examples:

- 127: command not found
- 126: command not executable

TASKS

Edit `hello_v7` so that it also prints out the command it was called with, as well the first and second arguments

Save your script as `hello_v8`, run it with two arguments
`Example: ./hello_v8 random_arg1 random_arg2`

After running your script, print the exit status code

Define a function `usage` which prints out a description of the script (use several `echo` commands). Call it at the end.

(Optional) Use an `if` statement to call `usage` only if the script is called without arguments

COURSE SUPPORT

SLACK (iords2021.slack.com)

- Course main channel: `#general`
- Topic channels: `#linux`, `#git`, `#python`, `#full-example`, `#machine-learning`

→ Check regularly for course info (esp. pinned items)

→ Do not hesitate to ask questions (please reply "in thread")

1-to-1 OFFICE HOURS for course questions:

- Week of October 4th: 20-min slots Tuesday and Wednesday (not on Friday exceptionally)

→ Book a time slot here: https://tinyurl.com/IORDS-office-hours

→ Do not hesitate to ask any kind of question, this is a beginner course !

EMAIL: methods@fcbg.ch ← Please whitelist!


Thank You!
Michael Dayan: methods@fcbg.ch
H N P
METHODS & DATA
campus biotech
Fondation Campus Biotech Geneva