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-linux3`

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)

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

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`**

On site support (including coding):


Maël


Louis


Nathan

Remote support (including coding):


Serafeim

HNP METHODS & DATA

LECTURE OBJECTIVES

Linux lectures objectives:

Linux Part 1

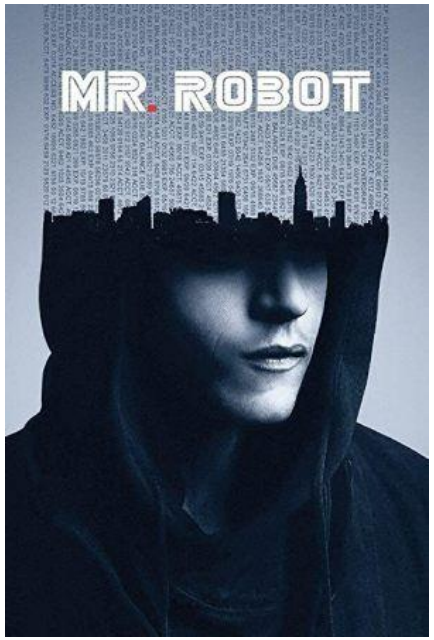
- Be comfortable with Linux & the command line
- Understand the notion of kernel, shell and command line
- Understand the linux file system structure
- Know how to navigate the file system
- Know how to create, delete and interact with (text) files
- Understand file permissions and know how to modify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

Linux Part 2

- Learn how to write shell scripts
  - Variables
  - For loop
  - More control flow statements (if-else statements, etc.)

Linux Part 3

  - Define and use user-provided arguments
- Manipulate string variables
- Understand the basics of SSH and how to use this protocol

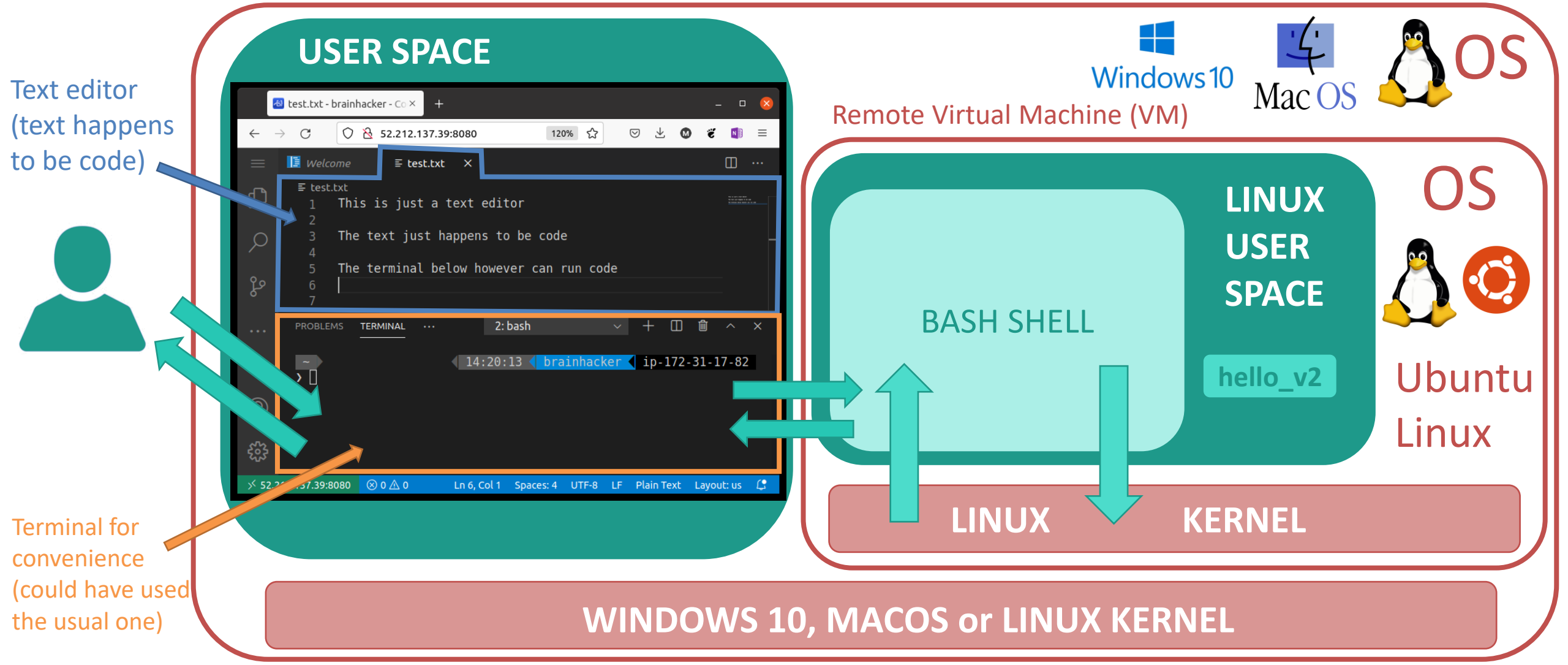


HNP METHODS & DATA

→ Write your own script usable by other users to automate specific operations

(+ better understand series like Mr Robot)

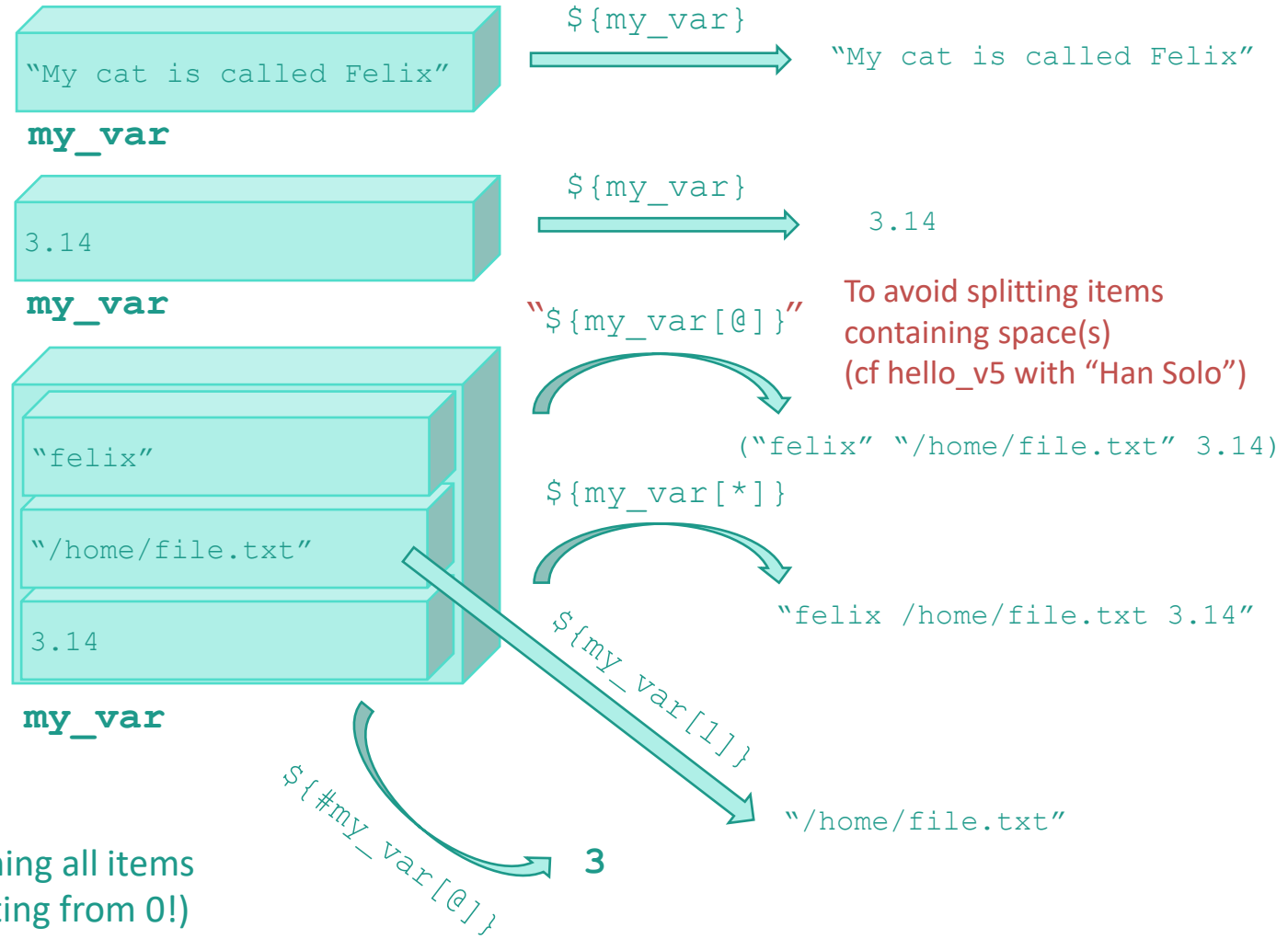
RECAP FROM LINUX 2



RECAP FROM LINUX 2

shebang `#!/bin/bash` absolute path to interpreter

- A variable is a reference to an object which could be:
  - A string:
    - `my_var="My cat is called Felix"`
  - A number (integer, float)
    - `my_var=3.14`
  - A list of objects, called an array (separator is a space → quotes are important):
    - `my_var=("felix" "/home/file.txt" 3.14)`
  - The output of a command (string, number, …) (command expansion)
    - `my_output=$(command)`
- The content of a variable is obtained using `${…}`
  - For a simple variable (i.e. not an array):
    - `echo ${my_var}`
  - For an array variable:
    - `echo ${my_var[@]}` to return the whole array
    - `echo ${my_var[*]}` to return a single string joining all items
    - `echo ${my_var[K]}` to return the K^{th} item (starting from 0!)
    - `echo ${#my_var[@]}` to return the total number of items



WRITING SHELL SCRIPTS – CONTROL FLOWS

FOR LOOP

- Loop directly on the items of the array

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

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

- Loop on an integer index K from 0 to N

Initialization: `K=0` — Continuation condition: `K<=N` — Update after each iteration: `K+=1`

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

IF-ELSE STATEMENT

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

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	≠

```
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

WHILE LOOP

```
while [[ COND ]]; do
    commands
done
```

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 given to the script
$2	The second argument given to the script
…	
$#	The number of arguments

Example 1

A script called `greeter`

```
#!/bin/bash
echo "Hi!"
```

What happens when running it

```
> greeter
Hi!
> greeter "Bob"
Hi!  (because no special variables
      are used in the script)
```

Example 2

A script called `friendly_greeter`

```
#!/bin/bash
echo "Hi $1!"
```

What happens when running it

```
> friendly_greeter
Hi!  (because $1 is empty)
> friendly_greeter "Bob"
Hi Bob!
```

Special variable after running the script

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

Examples: 127 for command not found, 126 for command not executable

Part of your script can be isolated in its own function:

- that you can call at any point in the script
- to which you can provide arguments

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

Example 1

```
#!/bin/bash
greetings(){
  echo -n "Good morning "
}
crew=("R2D2" "Leila" "C3PO")
for member in "${crew[@]}"; do
  greetings
  echo ${member}
done
```

Example 2

```
#!/bin/bash
greetings(){
  echo "Good morning $1"
}
crew=("R2D2" "Leila" "C3PO")
for member in "${crew[@]}"; do
  greetings ${member}
done
```

WRITING SHELL SCRIPTS – SPECIAL VARIABLES / FUNCTION

TASKS

Create a dir `hello_proj` in your home dir, and copy there the file `hello_v7` from the `~/solutions/hello_proj` dir

Open `~/hello_proj/hello_v7` in VS Code (<IP>:8080 in internet browser, password: `braincode!`) and edit it as `hello_v8` so that now it also prints out the command it was called with, as well the first and second arguments.
For example, the output of:

```
/home/brainhacker/helloproj/hello_v8 "hello" "bonjour"
```

Should be:

```
/home/brainhacker/helloproj/hello_v8
hello
bonjour
```

TIP: look at special variables on the previous slide

Note: don't forget to use `shellcheck` on your script (e.g. `shellcheck hello_v8`)

Test your script by executing it with two arguments

After running your script, print the exit status code (TIP: look at special variables on the previous slide)

Inside your script, define a function `usage` which prints out a description of the script (use several `echo` commands). Call this function within your script at the very end.
TIP: see example 1 on previous slide on how to define, and then call a function

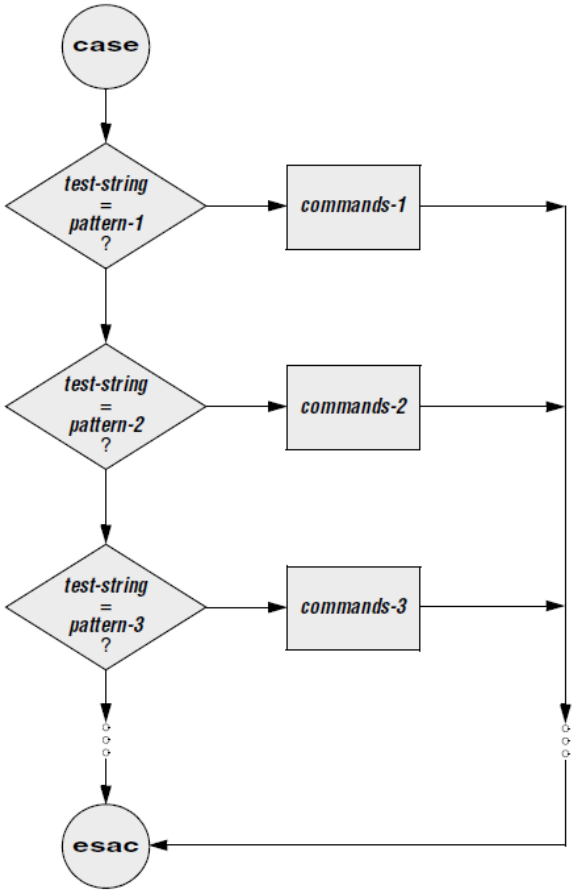
(Optional) Use an `if` statement to call `usage` only if the script is called without arguments

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
```

`case` spelled backward

[1] https://www.thegeekstuff.com/2010/07/bash-case-statement/

WRITING SHELL SCRIPTS – PARSING ARGUMENTS WITH GETOPTS

Parsing arguments with getopts

```
print_cow="no"
out_file=

SCRIPT=$(basename $0)

while getopts ":co:" arg; do
  case ${arg} in
    # Use cowsay to format output
    c )
      print_cow="yes"
      ;;
    # Indicate output file where to save the result
    o )
      out_file=${OPTARG}
      ;;
    # Option entered by user does not exist
    \? )
      echo "${SCRIPT}: Invalid option -${OPTARG} ignored"
      ;;
    # Option entered by user is missing the option parameter
    : )
      echo "${SCRIPT}: must supply a parameter to -${OPTARG}"
      ;;
  esac
done
```

Valid options: `-c -o`

"`:`" after `o` means the option `-o` takes a parameter (as in `-o <o parameter>`)

name of variable which will get the option value in the while loop (e.g. `c` then `o`, etc.)

allow for custom message when invalid option entered

content of the variable having the value of the option parameter (e.g. "`myfile.txt`")

case when option entered by user is missing the option parameter (e.g. –o without path to the file)

```
usage() {
  echo "Usage: $1"
  echo " -o out_file    path to the file where to save the output"
  echo "[-c]            use cow formatting"
}
if [[ $# -eq 0 ]]; then
  usage ${SCRIPT}
  exit 1
fi
```

```
# Required argument
if [[ -z "${out_file}" ]]; then
  echo "Required output file option missing."
  usage ${SCRIPT}
  exit 1
fi
```

WRITING SHELL SCRIPTS – PARSING ARGUMENTS WITH GETOPTS

TASK

Edit your script as `hello_v9` to add a `getopts` parser so that:

- the user can choose to have all outputs printed with `cowsay`
  - TIP 1: look at previous slide for how to define a cowsay format variable
  - TIP 2: before printing the output, now use if/else statements to check the of that variable, and use cowsay or not accordingly
- the user has to indicate a file where the output will be saved
  - TIP: cf previous slide
- the output is saved to that file (use > to write to a file, >> to append)
  - TIP: instead of printing the output to the screen use the file redirection operators (i.e. > or >>)
- the script will not run if the file provided by the user exists
  - TIP: look at example of required argument on the previous slide, and modify it to use the appropriate unary operator to check for file existence (cf slide on if statement + unary operator)
- add a usage message when the script is called without arguments
  - TIP: look at usage function example on previous slide

TIP

Start from the getopts_skeleton script (`~/getopts_example`) or copy code from that script to your own file.

Implement one feature at a time, then test it (the first feature of adding the cowsay format will require quite some work before testing it)

Use `shellcheck` to check for problems in your script: e.g. `shellcheck hello_v9`

Finish this task as HOMEWORK

Files + instructions to install VS Code + general support on Slack Channel

WRITING SHELL SCRIPTS – WHY CANNOT MY PROGRAM BE FOUND

The interpreter search for commands in `$PATH`

- Even if your program is in the current directory, the shell is not going to find it if your current directory is not in your `$PATH`
- Print the content of your `$PATH` (note: semicolon ":" is a separator between dirs)

```
echo ${PATH}
```

- To know the path to a program which can be found by the shell (i.e. in your `$PATH`) use `which`
- You can add a directory to `PATH` by modifying your `.bashrc`, a file read at startup by the shell which loads "environment variables"

```
HELLO_PROJ_DIR=${HOME}/hello_proj
export PATH=${PATH}:${HELLO_PROJ_DIR}
```

- Your current environment can be refreshed at any time from new content in your `.bashrc` with:

```
source ~/.bashrc
```

Use the full path `/home/brainhacker/.bashrc` if you cannot type `~`

- Your can print all your environment variables with `printenv`

TASKS

Find where is fortune by typing `which fortune`

Check if the shell can find one of your program (e.g. `hello_v8`) using `which`

Edit your `.bashrc` to add `${HOME}/hello_proj` to your `$PATH`. Then refresh your environment.

Check again if the shell can find your program

WRITING SHELL SCRIPTS – STRING MANIPULATION

The shell is useful to move around and create files

Often need access to specific parts of the original path (parent directory, filename, extension, etc.) and adapt it

➢ Removing part of a string

STR="i.like.dots.txt"

- `${STR#PATTERN}` **removes** the shortest *PATTERN* match from the start

Example: `echo ${STR#*.}` **i.like.dots.txt** → `like.dots.txt`

- `${STR##PATTERN}` **removes** the longest *PATTERN* match from the start

Example: `echo ${STR##*.}` **i.like.dots.txt** → `txt`

- `${STR%PATTERN}` **removes** the shortest *PATTERN* match from the end

Example: `echo ${STR%.*}` **i.like.dots.txt** → `i.like.dots`

- `${STR%%PATTERN}` **removes** the longest *PATTERN* match from the end

Example: `echo ${STR%%.*}` **i.like.dots.txt** → `i`

More examples

MY_PATH="/path/to/file.txt"

```
file_basename=${MY_PATH##*/}      /path/to/file.txt → file.txt
parent_path=${MY_PATH%/*}          /path/to/file.txt → /path/to
parent_name=${parent_path##*/}     /path/to          → to
```

➢ Replace substring by another

STR="i.like.dots.txt"

- `${STR//PATTERN1/PATTERN2}` **replaces** *PATTERN1 with PATTERN2*

Example: `echo ${STR//dots/dogs}`
i.like.dots.txt → `i.like.dogs.txt`

Useful to remove spaces from filenames

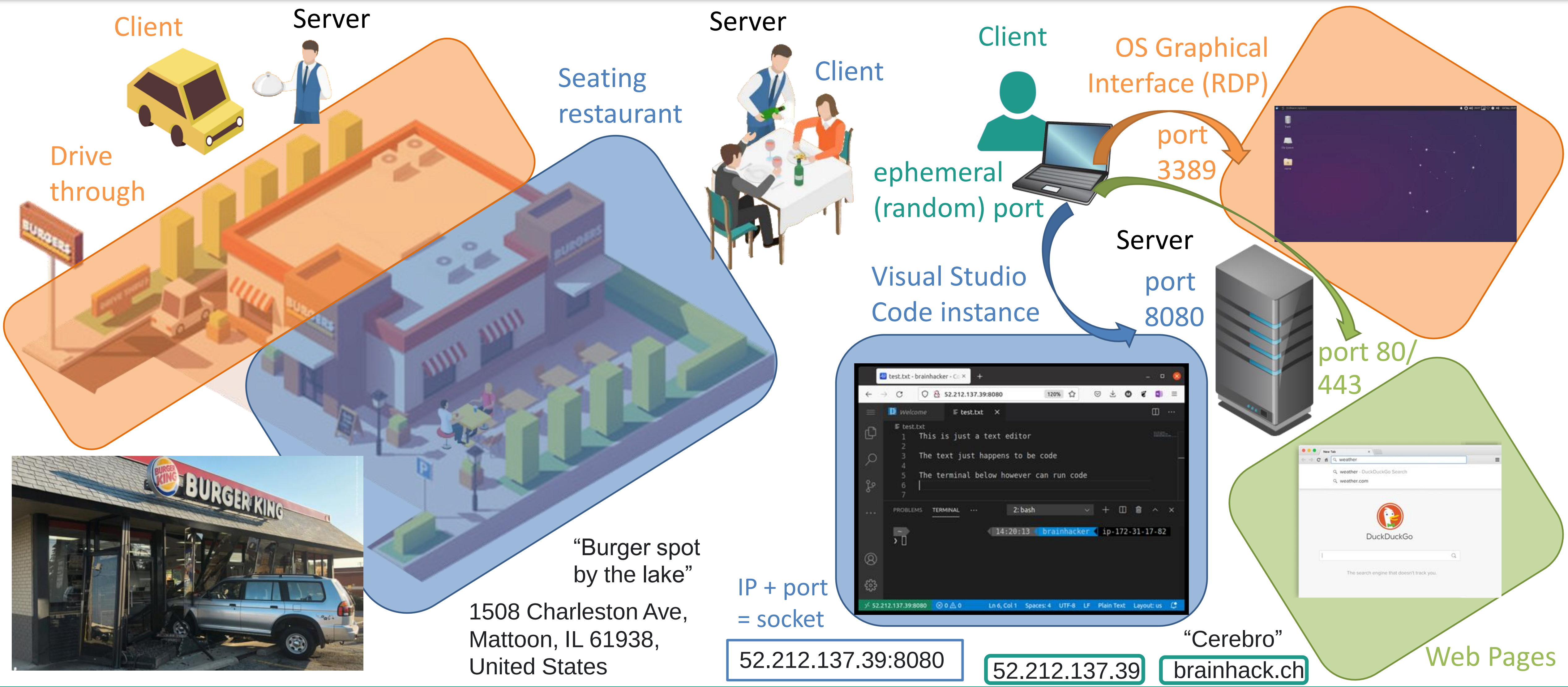
```
WIN_FILE="win10 crash.log"
mv "${WIN_FILE}" ${WIN_FILE// /_}
win10 crash.log → win10_crash.log
```

TIP: use echo with your command using string manipulation to check it before issuing it

```
echo "mv ${WIN_FILE} ${WIN_FILE//*/_}"
mv win10 crash.log
```

OOPS

BASIC NETWORKING: IP ADDRESS AND PORTS



RDP Client (e.g. Remote Desktop App) ⇔RDP Server
VS Code Client (e.g. Chrome, Firefox App) ⇔VS Code Server
Web Client (e.g. Chrome, Firefox App) ⇔Web Server

HNP METHODS & DATA

campus biotech

Fondation Campus Biotech Geneva

BASIC NETWORKING: IP ADDRESS AND PORTS

- Test if a server is “alive” with ping

  `ping <server IP or domain name>`

  Note: some servers block `ping`, so it may not be possible to tell if such servers are alive or not

- Check if a port is open with nmap

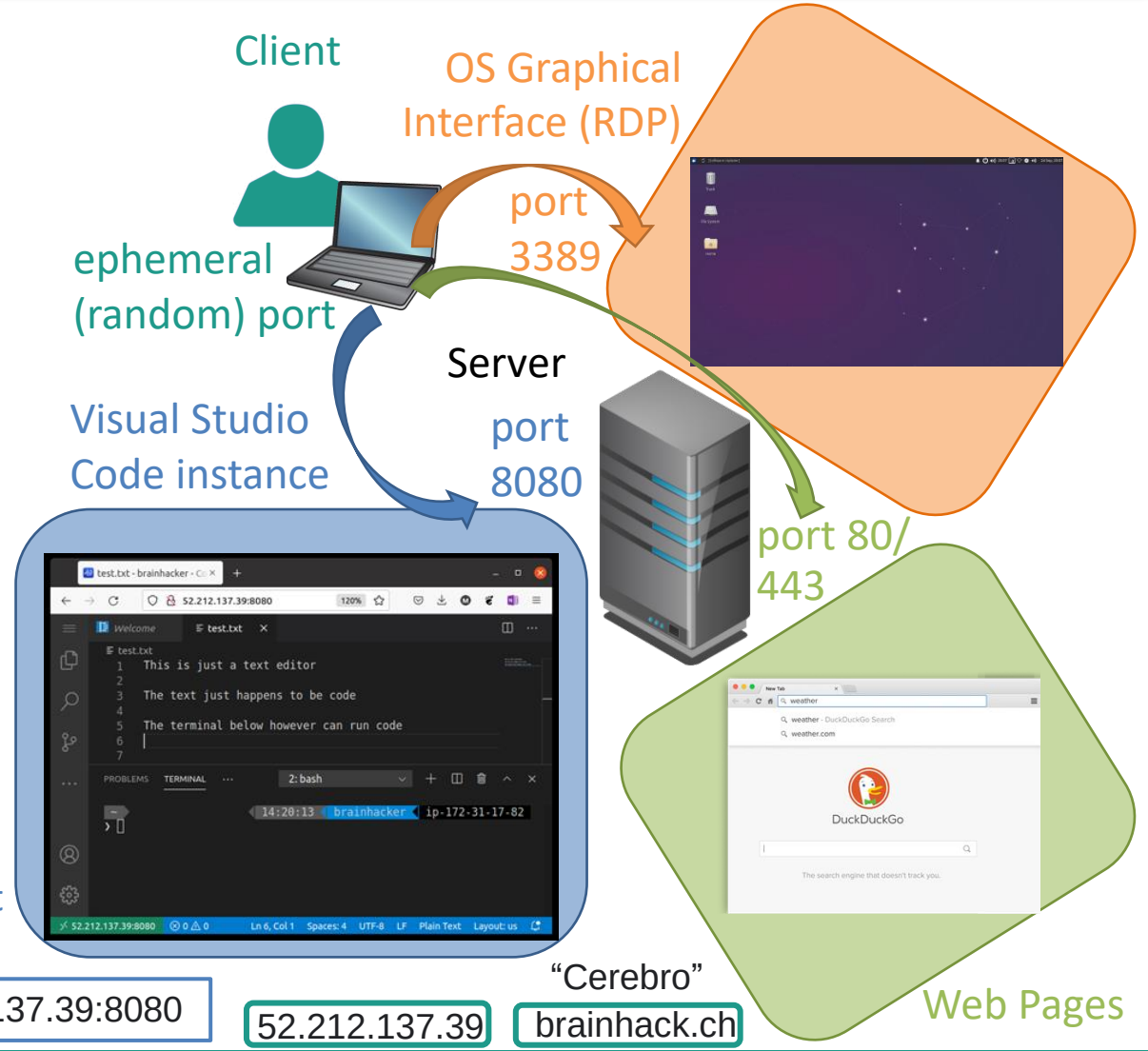
  `nmap -p <port number> <server IP or domain name>`

  - Status “open” means the port is open
  - Status “filtered” means it is blocked

TASKS

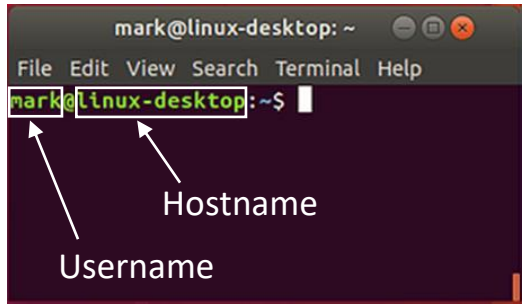
Check if the server hosting `brainhack.ch` is alive with `ping`. Bonus: what is the server IP address ?

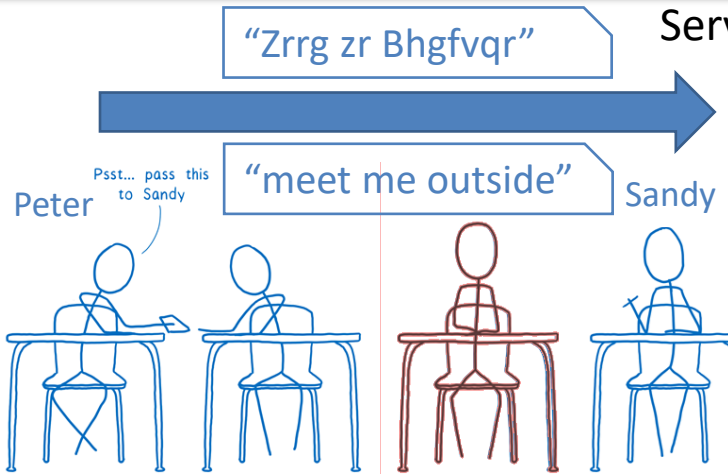
Check if the server hosting `brainhack.ch` has the port 443 open. What about port 3131 ?



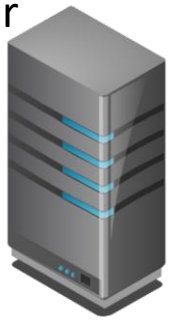
BASIC NETWORKING: SSH (Secure Shell)

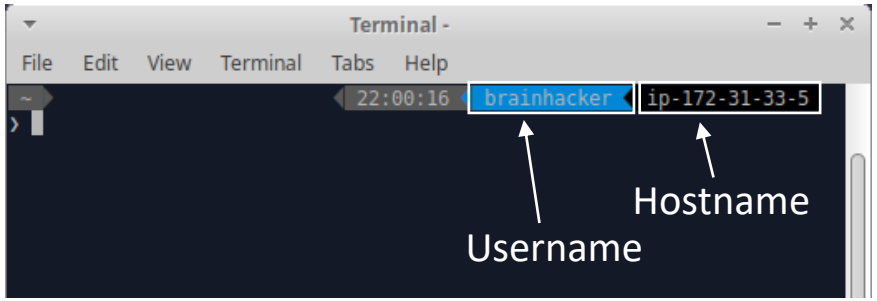
Client



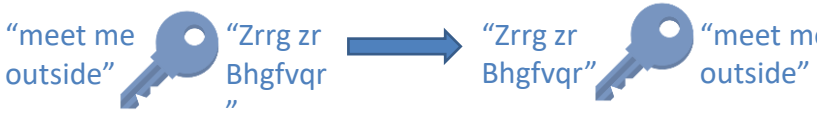


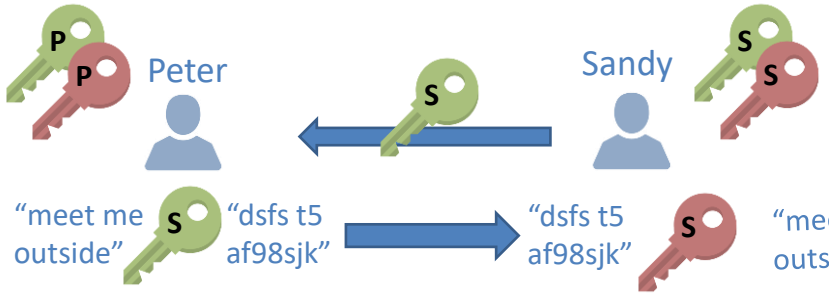
Server





Peter — Symmetrical Encryption (same key) — Sandy





Man-in-the-middle attack

Asymmetrical Encryption (public + private keys)

Public key to encrypt (to lock) — Private key to decrypt (to unlock)

P and S serve as IDs: a message encrypted with a public key will only be decrypted by the key owner

Steps:

1) Client + server setting up encryption
   - Server shows its ID to prove it is the right target
   - Temporary public + private keys to create a session symmetrical encryption key

Server ID

2) Client authentifiying itself to server
   a) Password (discouraged)
   OR
   b) Public ID key (must be in the authorized list on the server)

Client ID

If by public ID key, the server will use that key to encrypt a challenge message to be sent to the client. The client needs to prove its identity by decrypting the challenge message.

3) Communications within encrypted "tunnel" with symmetrical encryption key from step #1 can now take place.

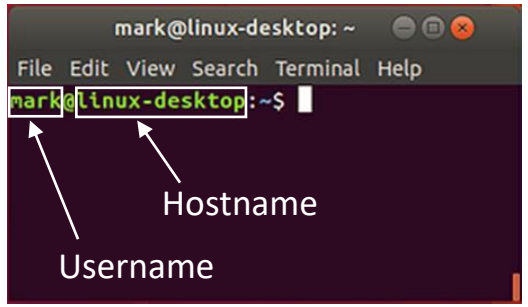
HNP METHODS & DATA

campus biotech

Fondation Campus Biotech Geneva

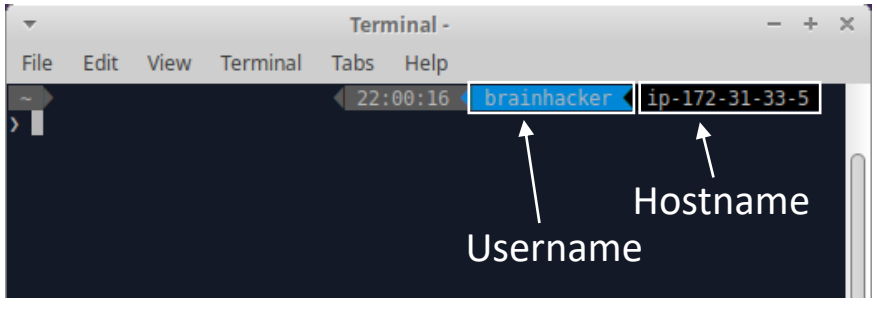
BASIC NETWORKING: SSH (Secure Shell) IN PRACTICE

User (SSH client)



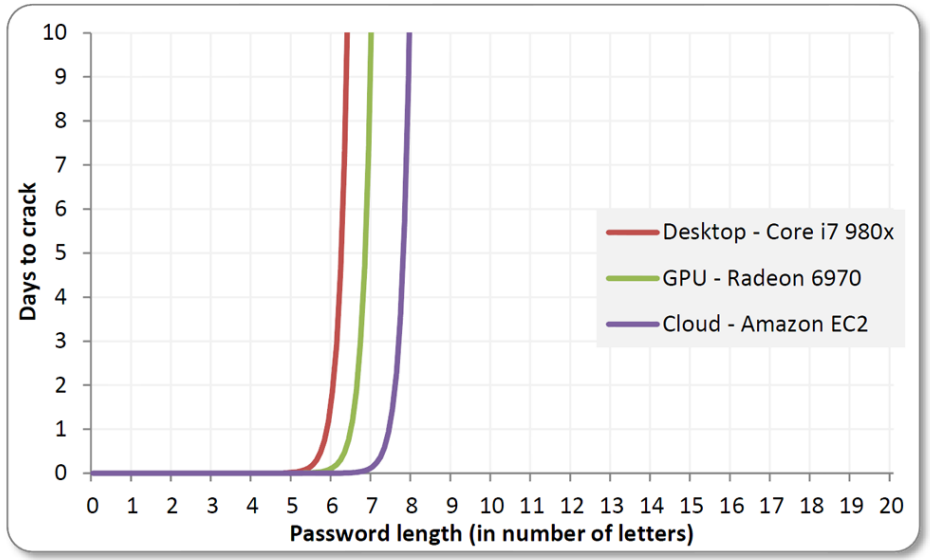
Encrypted SSH tunnel

SSH Server (port 22)



- Information required
  - IP address of the server
  - Name of the user to log into on the server (e.g. `brainhacker`)
  - Password for that user the first time, and client public key afterwards

1. Create a SSH key pair if you do not have one
   a) Check if it exists by listing the files in `~/.ssh`
      (the public key is typically called `id_rsa.pub`)
   b) If it doesn't exist, create an RSA key 4096-byte long with `ssh-keygen`
      `ssh-keygen -t rsa -b 4096` (keep pressing Enter key to select all default options (no passphrase)
   c) This will create two files:
      - `~/.ssh/id_rsa` this is your private key, **never share !**
      - `~/.ssh/id_rsa.pub` this is your public key, **feel free to share**
   d) (Optional) Check your key fingerprint: `ssh-keygen -l`



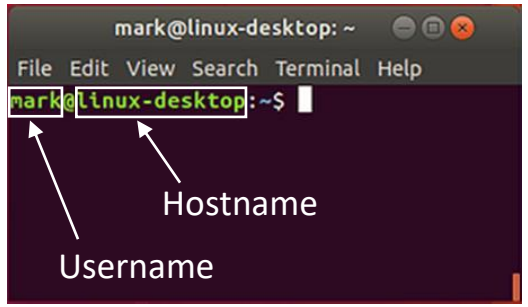
HNP METHODS & DATA

campus biotech

Fondation Campus Biotech Geneva

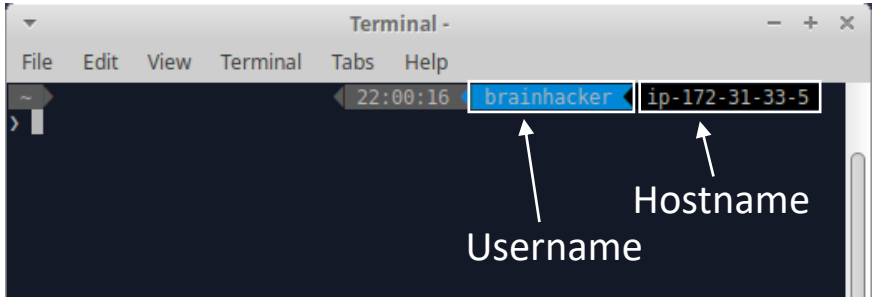
BASIC NETWORKING: SSH (Secure Shell) IN PRACTICE

User (SSH client)



Encrypted SSH tunnel

SSH Server (port 22)



- Information required
  - IP address of the server
  - Name of the user to log into on the server (e.g. `brainhacker`)
  - Password for that user the first time, and client public key afterwards

1. Create a SSH key pair if you do not have one
   a) Check if it exists by listing the files in `~/.ssh`
      (the public key is typically called `id_rsa.pub`)
   b) If it doesn't exist, create an RSA key 4096-byte long with `ssh-keygen`
      `ssh-keygen -t rsa -b 4096`
      (keep pressing Enter key to select all default options (no passphrase)
   c) This will create two files:
      - `~/.ssh/id_rsa` this is your private key, **never share !**
      - `~/.ssh/id_rsa.pub` this is your public key, **feel free to share**
   d) (Optional) Check your key fingerprint: `ssh-keygen -l`

2. Add your ID to the list of authorized ID for a given user on the server
   a) Copy your public key to the server list (need identifying with that user password)
      `ssh-copy-id username@remote_host`

   Note: this command copy the content of your public key `~/.ssh/id_rsa.pub` to the list of authorized keys on the host server at `/home/username/.ssh/authorized_keys`
   b) Check the server fingerprint if you know it, and enter the user password as prompted
   c) Now test you can SSH to the server without having to type any password
      `ssh username@remote_host`

BASIC NETWORKING: SSH (Secure Shell) IN PRACTICE

TASKS

Check your current username and computer name (your current machine is the client computer)
TIP: use `whoami` and `hostname`

Ask anyone for an IP (it will be considered a server). Check if you can SSH to that server as `brainprober` (password: `brainprobe!`) without typing a password
TIP: you will have to type the password

Once you SSH'd, check your server username and computer name. It should be different from step 1.

Exit from the server back to your own machine (the client) by running the `exit` command. Now check your current user and computer names again.

Create an RSA key pair of length 4096 bytes

Display the content of the client public key file
TIP: the file should be `~/.ssh/id_rsa.pub`

Keep your current terminal to have the printed content of the client public key handy

Open a new terminal

From this new terminal use `ssh-copy-id` to copy your public key to the list of authorized keys for the user `brainprober` on the server you got an IP of

Check you can now ssh to that server without password

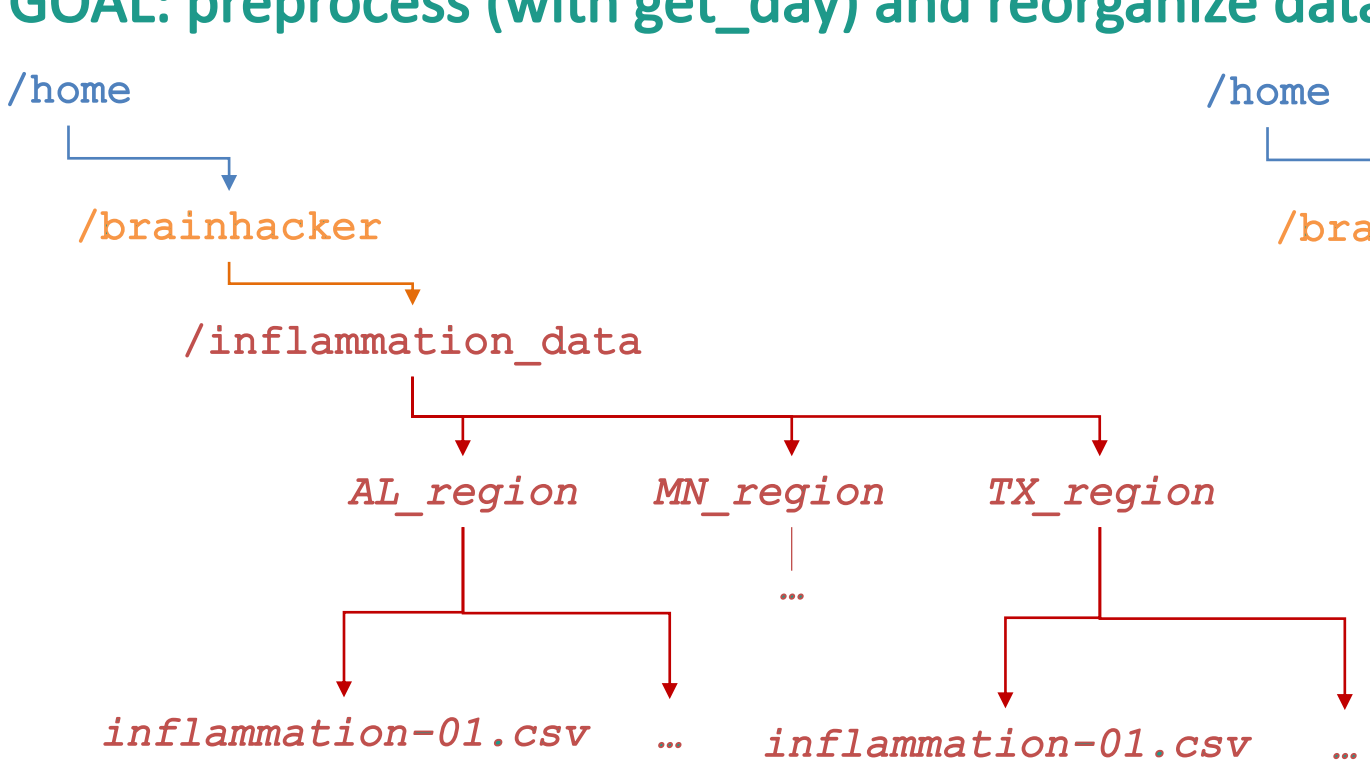
Now compare the content of the `.ssh/authorized_keys` file in the `brainprober` home directory on the server (new terminal) to your client machine public key (old terminal)

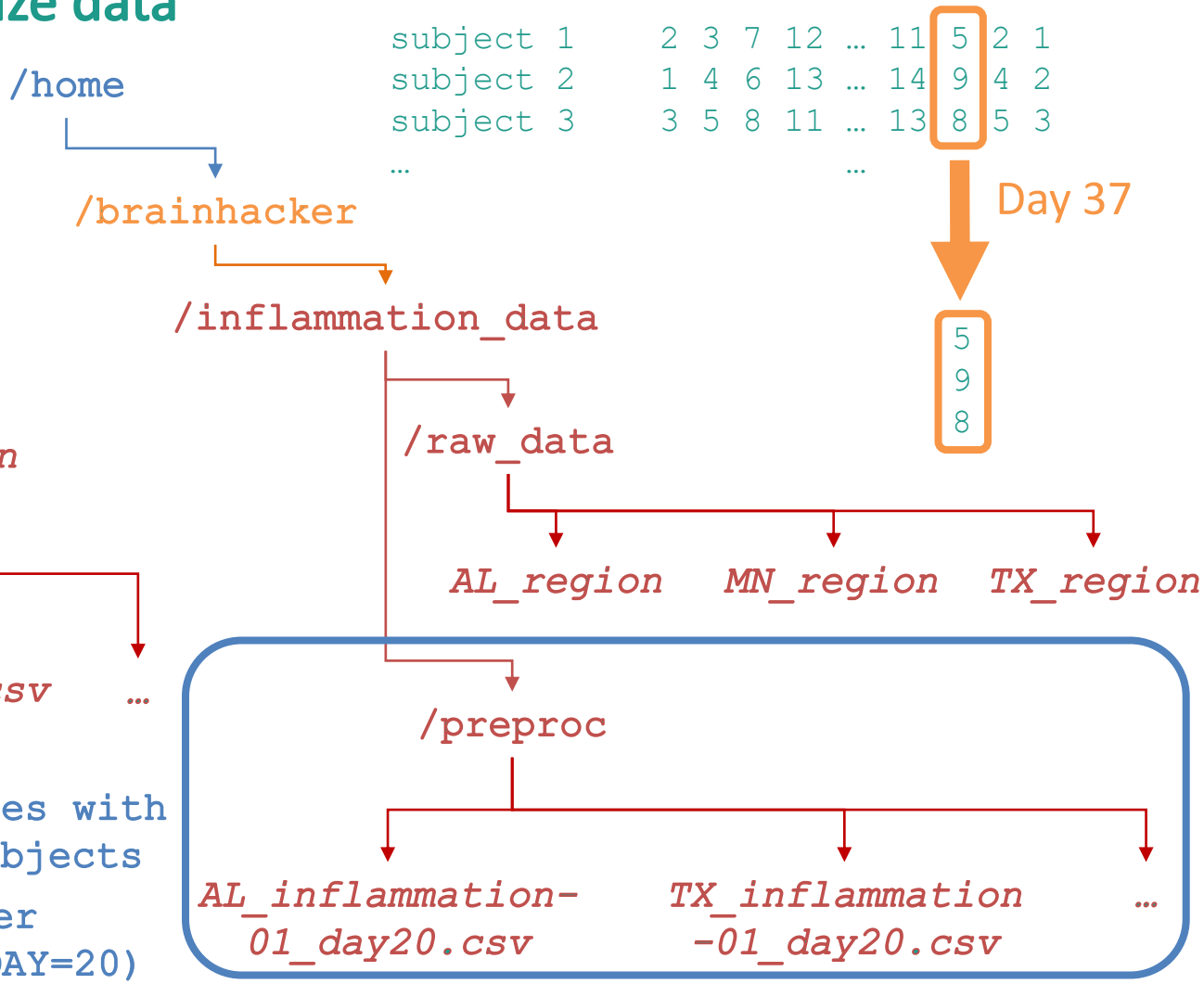
Exit from the server you SSH'ed in (TIP: type `exit`)

Close the new terminal you used to SSH in the server

WRITING SHELL SCRIPTS – LINUX CAPSTONE HOMEWORK

GOAL: preprocess (with get_day) and reorganize data





subject 1	2	3	7	12	…	11	5	2	1
subject 2	1	4	6	13	…	14	9	4	2
subject 3	3	5	8	11	…	13	8	5	3
…						…			

Day 37

5
9
8

Only for CSV files with more than 30 subjects

DAY set by user
(in this example DAY=20)

WRITING SHELL SCRIPTS – LINUX CAPSTONE HOMEWORK

TASKS part 1

Create a directory `/home/brainhacker/inflammation_proj`

Copy recursively the content of `/home/brainhacker/inflammation_data` to a directory `raw_data` inside `/home/brainhacker/inflammation_proj`

Create a new script call `preproc_day_v1` which:

- assigns to a variable `IN_DIR` the path `/home/brainhacker/inflammation_proj/raw_data`
- assigns to a variable `DAY` an integer between 1 and 40
- assigns to an array `csv_files` all the CSV files which:
  - are in the subdirectories of `${IN_DIR}`
  - include "`inflammation`" in their filenames

(tip: find the right "globbing" expression)

- loop on each item of the resulting array and print:
  - the name of the file parent directory (e.g. `MN_region`) [TIP: use two string manipulation statement to get that name]
  - the name of the file (e.g. `inflammation-02.csv`) [TIP: use an additional string manipulation]
  - the number of subjects in that file (TIP: use `cat`, `|` and `wc -l`)

Make the script executable and run it. Use `shellcheck preproc_day_v1` to check for errors

WRITING SHELL SCRIPTS – LINUX CAPSTONE HOMEWORK

TASKS part 2

Look at the usage of the command `get_day` by typing the command name without arguments

Test this command on a test output file to check it works. Then delete the test output file.

Save `preproc_day_v1` as `preproc_day_v2` and edit is so that it also:

- assigns to a variable `OUT_DIR` the path `/home/brainhacker/inflammation_proj/preproc`
- create the `OUT_DIR` directory if the directory does not exist [TIP: use an if-else statement with a unary condition]
- loop on each item of `csv_files` and:
  - if the file has less than 30 subjects, print a message indicating the file is excluded
    TIP: assign the number of subjects to a variable using command substitution, i.e. `my_var=$(my_command)`
  - If it doesn't print a message indicating the file is included for preprocessing and print what would be the desired path for the day-extracted data to be created (e.g. if `DAY=20` then `/home/brainhacker/inflammation_proj/preproc/AL_inflammation-01_day20.csv` is to be created from `/home/brainhacker/inflammation_data/AL_region/inflammation-01.csv`)
  - print the command `get_day` which would need to be used to produce the output in the desired path above

Make the script executable and run it. Use `shellcheck preproc_day_v2` to check for errors.
Check the printed commands make sense.

WRITING SHELL SCRIPTS – LINUX CAPSTONE HOMEWORK

TASKS part 3

Save `preproc_day_v2` as `preproc_day_v3` and edit is so that the printed `get_day` command are replaced by the real commands

Test the script

Save `preproc_day_v3` as `preproc_day_v4` and use `getopts` for the user to indicate the `DAY` and the output directory `OUT_DIR` on the command line so that:

- `OUT_DIR` is a compulsory argument (the program should exit with error code 1 if not entered by user)
- `DAY` is optional and has default value 20 if not indicated on the command line
- `DAY` should be between 1 and 40 (program should exit if this is not the case)
- the program displays a `usage` function (indicating optional argument in square brackets) when:
  - the script is called without arguments
  - the script is called with improper parameters (e.g. `DAY=50`)

Tip: create a `usage` function and call it every time necessary

Test the script

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 11th: 20-min slots Tuesday & Wednesday + full morning support for homework and capstone homework project on Friday morning (room H8-01-B)

→ 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!

HNP METHODS & DATA

campus biotech

Fondation Campus Biotech Geneva


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