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-git1`

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 **`#git`**

On site support (including coding):


Maël


Louis


Nathan

Remote support (including coding):


Serafeim

HNP METHODS & DATA

campus biotech

Fondation Campus Biotech Geneva

LECTURE OBJECTIVES

GIT lectures objectives (you should be able to…):

- Know what GIT is useful for and identify some famous actors in the Git ecosystem
- Understand the “promotion model” and “commits” at the heart of GIT
- Describe the typical workflow to record code changes
- Examine the code history and compare different code versions
- Understand the concepts of branches and branch merging

GIT Part 1

- Know how to collaborate with Git using Github
- Understand the concepts of remote repository / remote branches
- Synchronize code changes between local and remote repositories
- Distinguish between different kinds of merge

GIT Part 2

- Collaborate on public projects with Github
- Know advanced merging techniques and deal with conflicts
- Debugging with GIT

GIT Part 3

→ Track your code changes with GIT and collaborate on your own or public projects with Github

WHY LEARNING GIT?

→ Because everyone is using it?

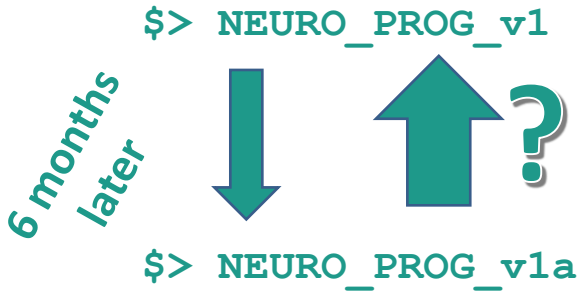
→ To store (and restore) code versions

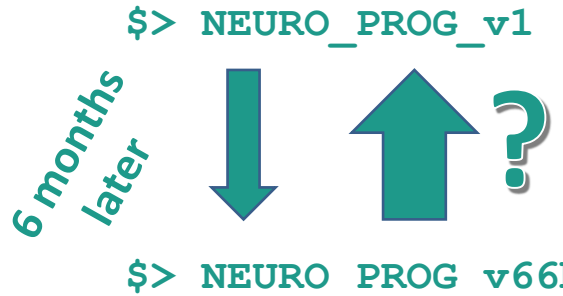
Version history of 'Macworld 2013 Talk.pages'

Dropbox keeps a snapshot every time you save a file. You can preview and restore 'Macworld 2013 Talk.pages' by choosing one of the versions below:

Version 11 (current)	Edited by Christian Boyce (cboyce-MacBook-Pro)	8 mins ago	55.44 KB
Version 10	Edited by Christian Boyce (cboyce-MacBook-Pro)	58 mins ago	55.43 KB
Version 9	Edited by Christian Boyce (cboyce-MacBook-Pro)	1 hr ago	55.43 KB
Version 8	Edited by Christian Boyce (cboyce-MacBook-Pro)	1/13/2013 9:39 PM	35.74 KB
Version 7	Edited by Christian Boyce (cboyce-MacBook-Pro)	1/13/2013 9:34 PM	36.33 KB
Version 6	Edited by Christian Boyce (cboyce-MacBook-Pro)	1/13/2013 9:28 PM	35.95 KB
Version 5	Edited by Christian Boyce (cboyce-MacBook-Pro)	1/13/2013 9:27 PM	52.77 KB
Version 4	Edited by Christian Boyce (cboyce-MacBook-Pro)	1/13/2013 9:26 PM	34.18 KB
Version 3	Edited by Christian Boyce (cboyce-MacBook-Pro)	1/13/2013 9:25 PM	31.4 KB
Version 2	Edited by Christian Boyce (cboyce-MacBook-Pro)	1/13/2013 9:24 PM	31.4 KB
Version 1 (oldest)	Added by Christian Boyce (cboyce-MacBook-Pro)	1/13/2013 9:22 PM	35.62 KB

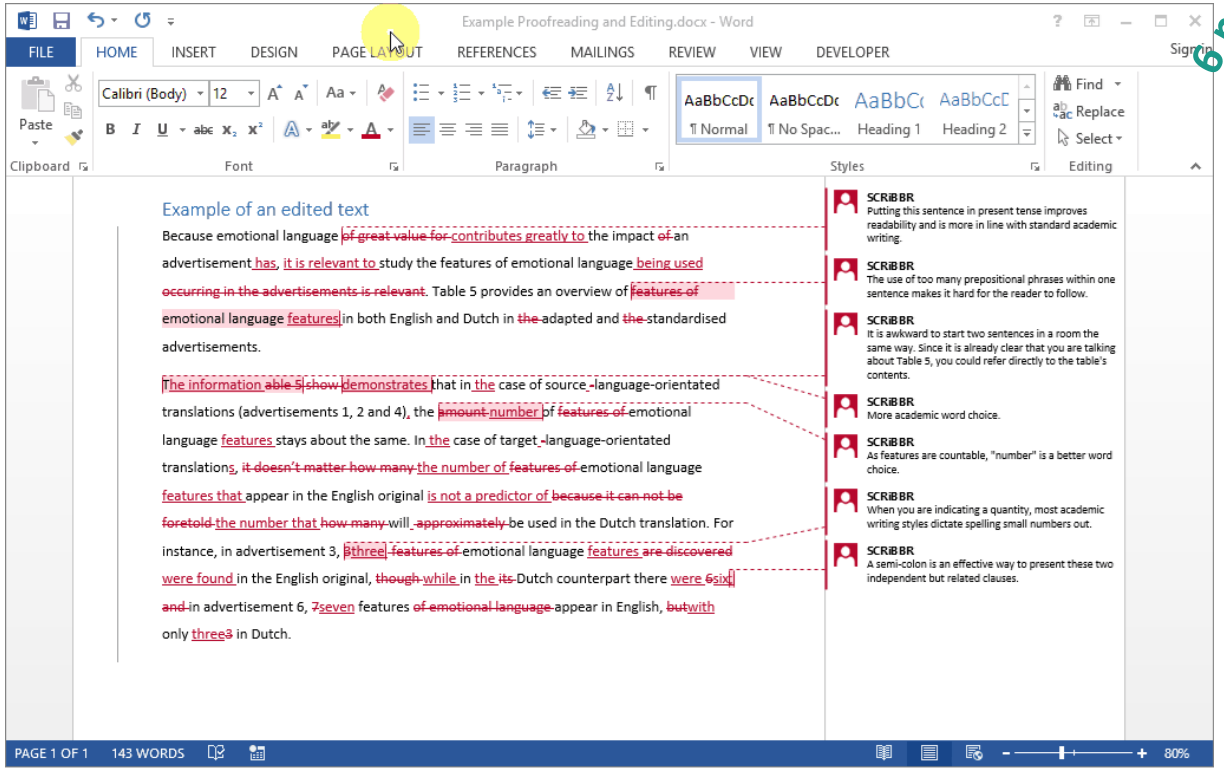
Restore Cancel

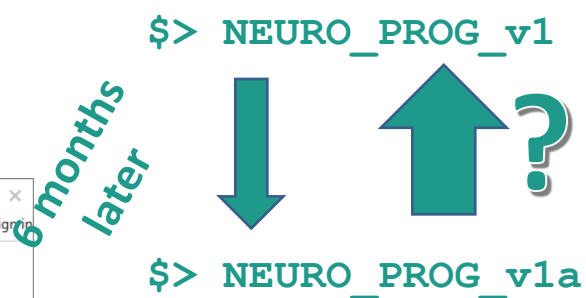


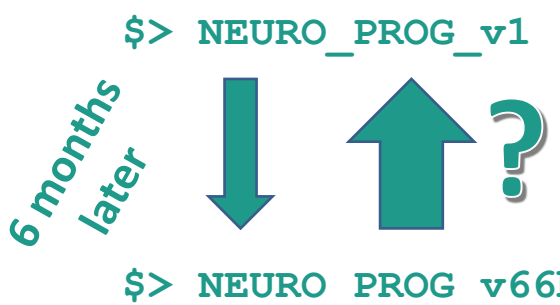


WHY LEARNING GIT?

- → Because everyone is using it?
- → To store (and restore) code versions
- → To track changes







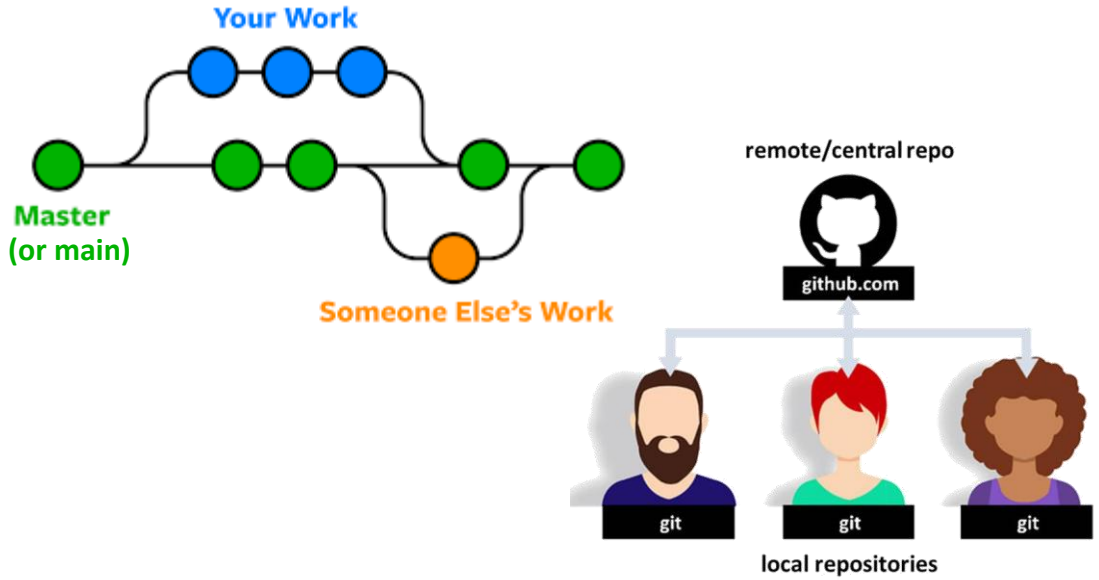
HNP METHODS & DATA

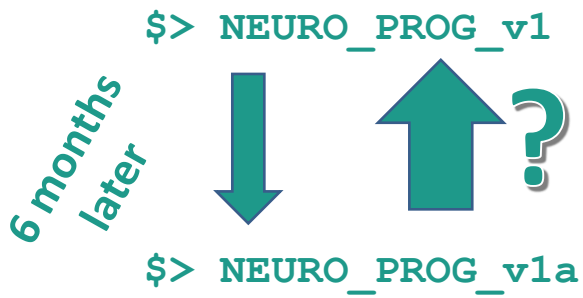
campus biotech

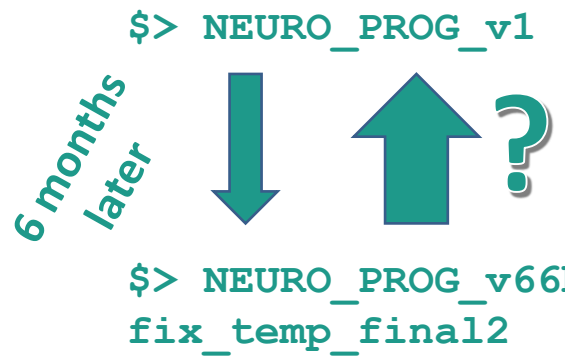
Fondation Campus Biotech Geneva

WHY LEARNING GIT?

- → Because everyone is using it?
- → To store (and restore) code versions
- → To track changes
- → **To collaborate**







- → Because everyone is using it !

… AND MORE (automation, etc.)

(BRIEF) HISTORY OF GIT

UNiplexed

→ MULTICS: MULTiplexed Information and Computing Service

→ 1969: UNICS

→ 1977: Berkeley Unix
Berkeley Software Distribution (BSD)

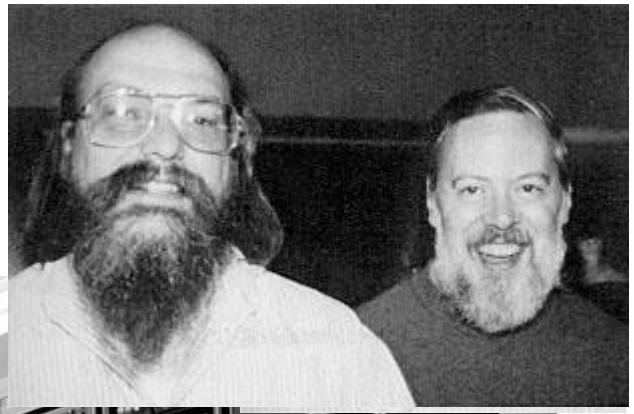
[→ NeXTSTEP → MacOS]

→ 1983: GNU's Not Unix (GNU) General Public License (GPL)

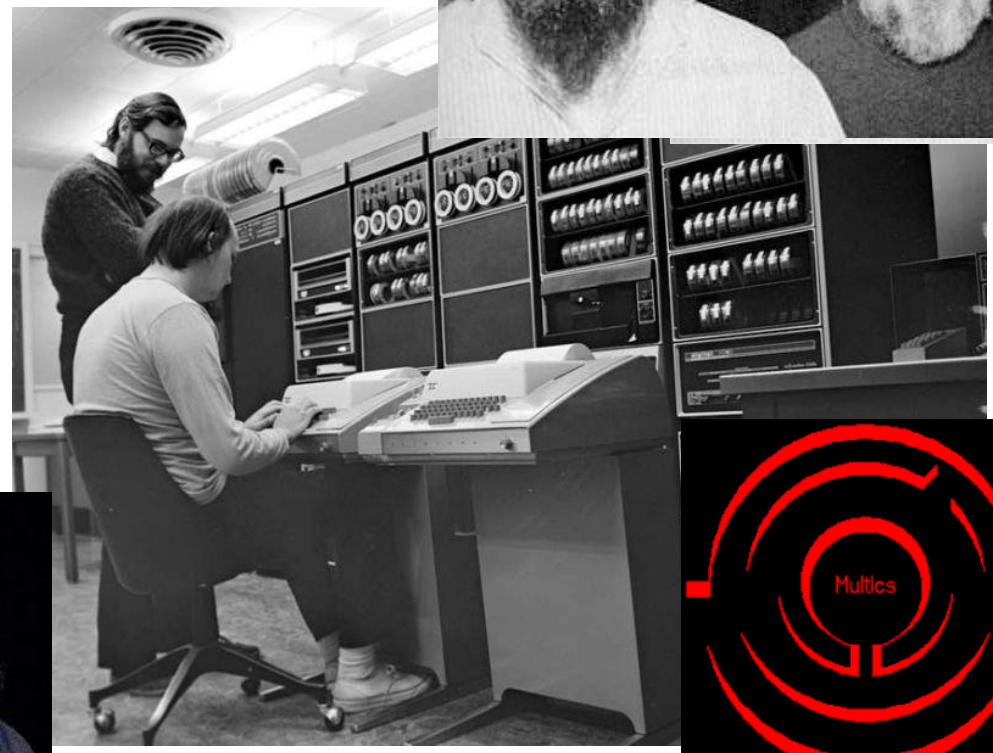
→ 1987: Mini-Unix (MINIX)

→ 1991: Linux

Massachusetts Institute of Technology



Ken Thompson & Dennis Ritchie



AT&T

Richard Stallman

Linus Torvalds

(BRIEF) HISTORY OF GIT

UNiplexed

→ MULTICS: MULtiplexed Information and Computing Service

→ 1969: UNICS

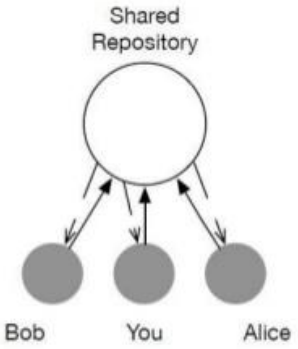
→ 1977: Berkeley Unix Berkeley Software Distribution (BSD)

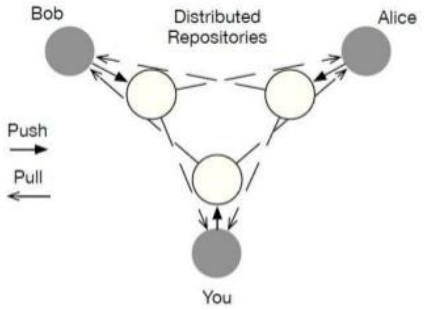
[→ NeXTSTEP → MacOS]

→ 1983: GNU's Not Unix (GNU) General Public License (GPL)

→ 1987: Mini-Unix (MINIX)

→ 1991: Linux





Stopped free use

→ 2002: BitKeeper

Larry McVoy

Andrew Tridgell

→ 2005 : Git (& Mercurial)

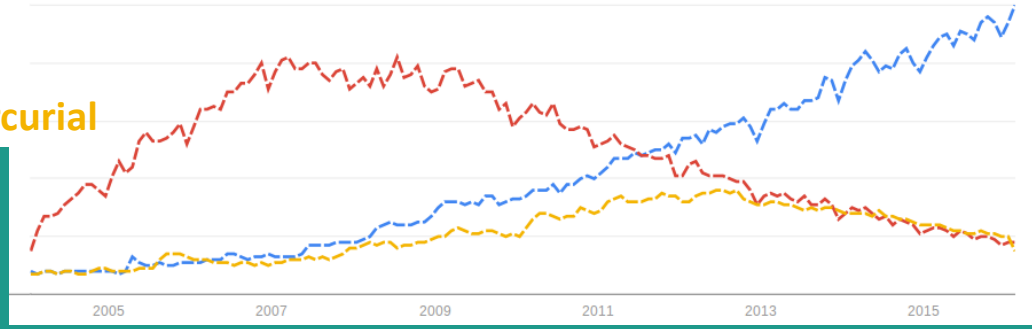
Linus: two-week goal, dev started in 2005 on April 3rd, with advanced capabilities in April 18th and use in production for Linux in June

Junio : maintainer since July 2005 (ver 1.0 in December 2005)

Junio Hamano

Linus Torvalds

SVN
GIT
Mercurial



2020: 50+M GitHub devs

HNP METHODS & DATA

GIT ECOSYSTEM

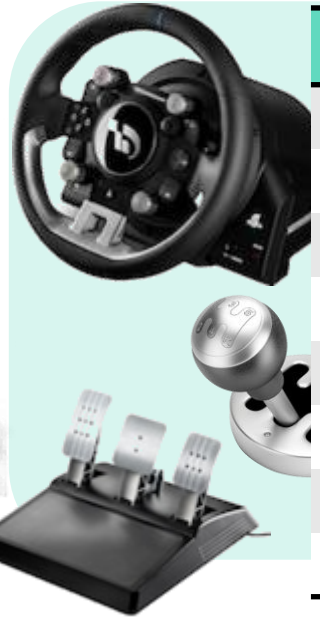
- → Core Git: version control with git command-line tools
- → Git distribution tools: e.g. simple GUI (gitk), bash shell for Windows distribution
- → Git repository hosting sites: e.g. Github, Gitlab, BitBucket, … (facilitating collaboration)
- → Git GUI packages: SourceTree, Git Cola, SmartGit, TortoiseGit, GitHub Desktop

Choice of Git interface?

- → Terminal:
  - Reference interface (universal)
  - All functionalities available
  - Versatile, with flexible interaction with other command-line tools

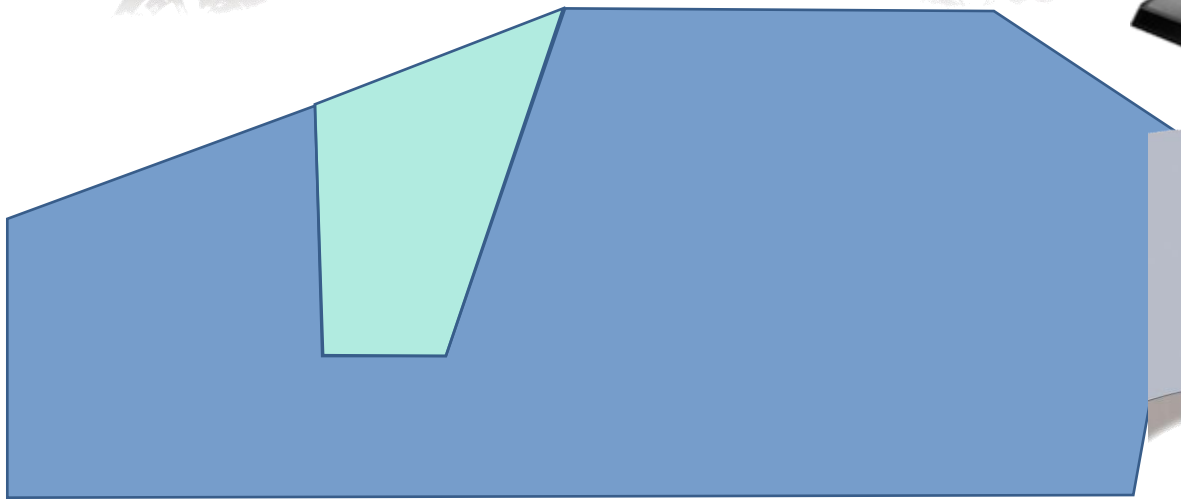
+ gitk, meld

GIT COMMANDS – OVERVIEW



GIT PORCELAIN COMMANDS

`git init`	`git clone`	`git branch`
`git add`	`git help`	`git fetch`
`git commit`	`git tag`	`git merge`
`git status`	`git config`	`git rebase`
`git log`	`git pull`	`git cherry`
`git show`	`git push`	`git cherry-pick`
`git diff`	`git grep`	`git fetch`
`git difftool`	`git checkout`	`git reset/revert`



+ stack-overflow

GIT PLUMBING COMMANDS

`git cat-file`
`git ls-files`
`git read-tree`
`git write-tree`
…

HNP METHODS & DATA

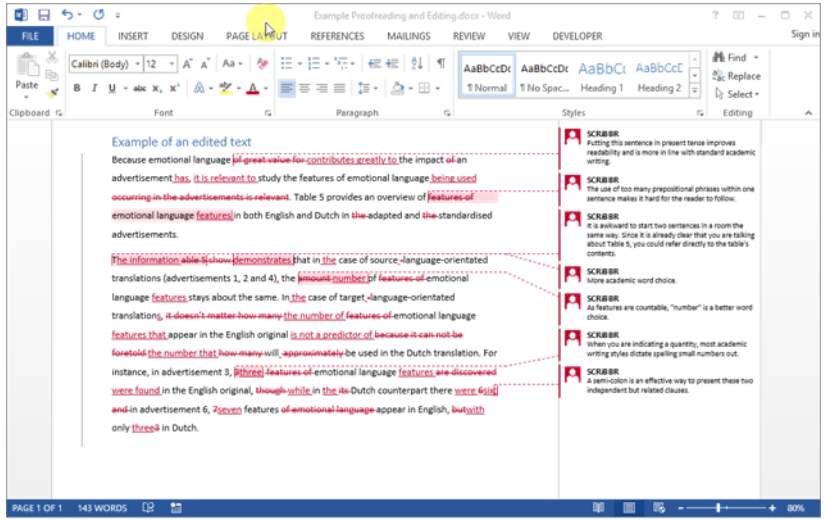
`git` *`COMMAND OPTIONS OPERANDS`*

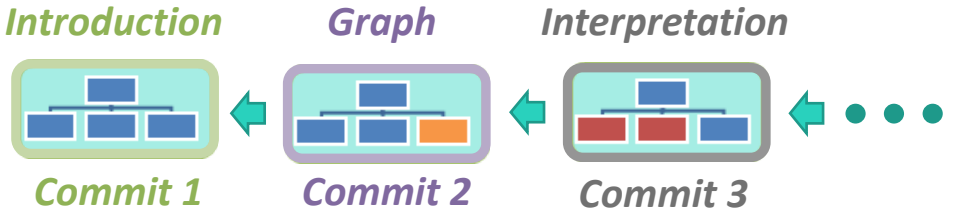
`man git-`*`COMMAND`* *`(e.g. man git-commit)`*

campus biotech

Fondation Campus Biotech Geneva

GIT PROMOTION MODEL





History:

Create introduction

- Add in *intro.txt* "In the field…"

Show graph results

- Add in *results.txt* "As seen on the graph…"
- Add *image* file of graph

- Add in *results.txt* "Our results show…"
- Add in *discussion.txt* "These results suggest…"

Add inter-pretation X

- Add authors / list.txt
- Add authors / affiliation.txt

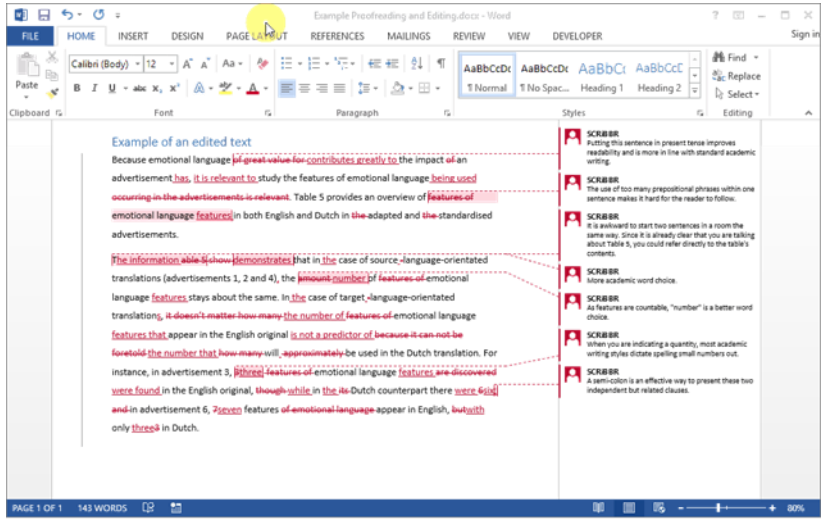
Provide authors info

HNP METHODS & DATA

campus biotech

Fondation Campus Biotech Geneva

GIT PROMOTION MODEL



History:

Create introduction

- Add in *intro.txt* “In the field…”

Show graph results

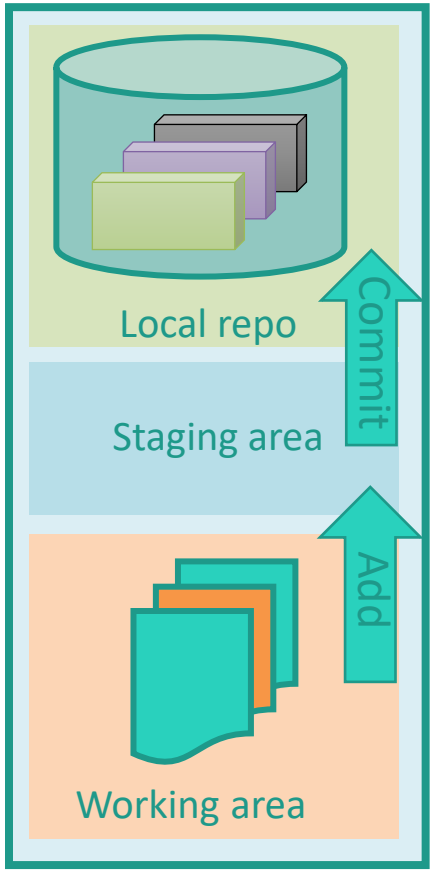
- Add in *results.txt* “As seen on the graph…”
- Add *image* file of graph

- Add in *results.txt* “Our results show…”
- Add in *discussion.txt* “These results suggest…”

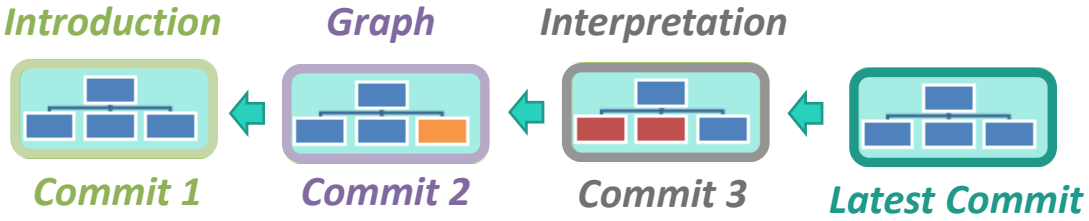
Add inter-pretation X

- Add authors / list.txt
- Add authors / affiliation.txt

Provide authors info



User 1 laptop



Promotion model

- **Working area** (~ working tree ~ sandbox)
- **Staging area** (~ cache ~ index)

→ add progressively changes to the staging area for the next commit with `git add`

Note: added files become “tracked” by `git`

- **Local repository**

→ commit a set of changes as a snapshot with `git commit`

Staging area provides:

- fine-grained control over the set of changes that makes up a commit (i.e. a unit of work)
- a place to deal with file conflicts (cf. merge discussion)
- a place where to amend the last commit

HNP METHODS & DATA

```
git add
git commit
```

campus biotech

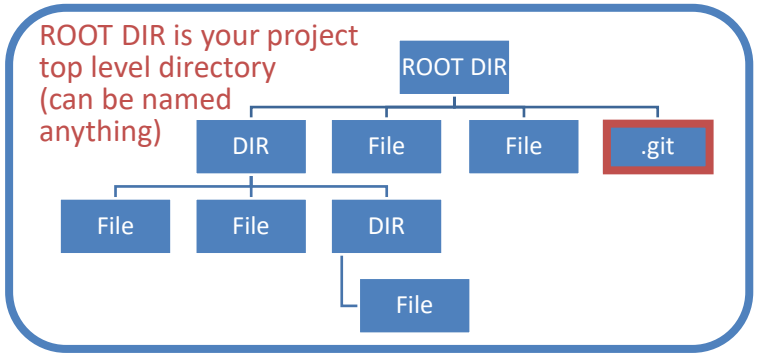
Fondation Campus Biotech Geneva

GIT PROMOTION MODEL

Where is the local repository stored ?

In the `.git` directory in your project root directory

It is created with `git init` (to do once per project)



History:

Create introduction

- Add in *intro.txt* "In the field…"

Show graph results

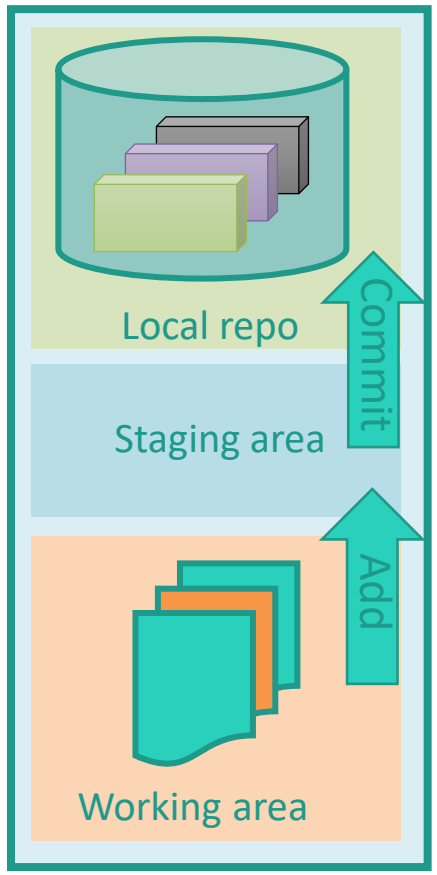
- Add in *results.txt* "As seen on the graph…"
- Add *image* file of graph

- Add in *results.txt* "Our results show…"
- Add in *discussion.txt* "These results suggest…"

- Add authors / list.txt
- Add authors / affiliation.txt

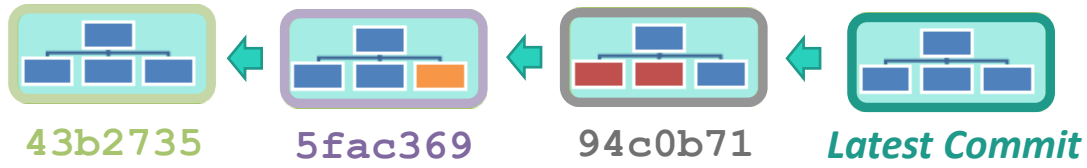
Provide authors info

Add inter-pretation X



User 1 laptop

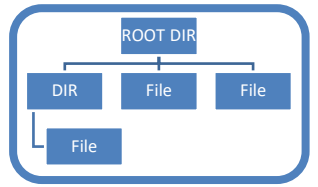
Introduction ***Graph*** ***Interpretation***



43b2735 5fac369 94c0b71 ***Latest Commit***

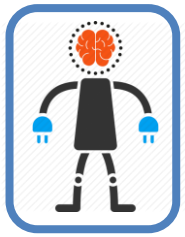
What is exactly a commit ?

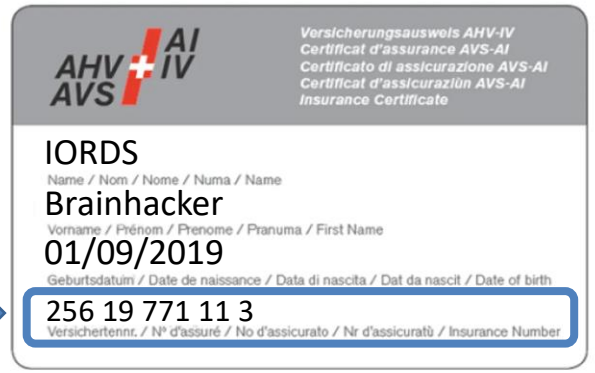
- All files and directories, represented by a single **checksum** (SHA1)
- Previous (parent) commit **checksum**
- Commit description: **message**
- **Author**
- **Date**



cf23df2207d99a74fbe169e3eba035e633b65d94

0de8c7f2bce77531a313
077b687f174e8e49169b





256	19	771	11	3
From name first letters	Year of birth	From sex and birth month & day	« Uniquizer » and nationality	Control number

HNP METHODS & DATA

`git init` (to do only once per project)
`git add`
`git commit`

campus biotech Fondation Campus Biotech Geneva

GIT PROMOTION MODEL

Last two git terminology elements before example

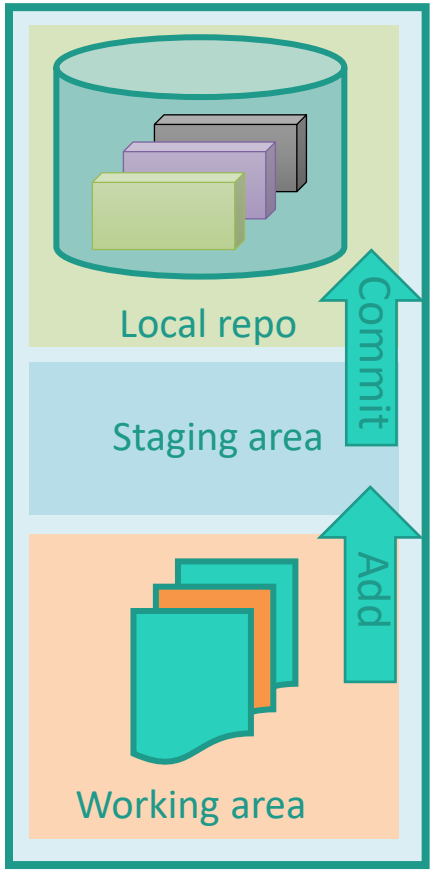
Branch:

- conceptually: a sequence of changes (commits)
- technically: a pointer to the latest commit

Note: the default branch name is usually `master`, however this changed in October 2020 for GitHub for which it is now `main`

HEAD:

- conceptually: the active branch
- technically: the pointer to the active branch



User 1 laptop

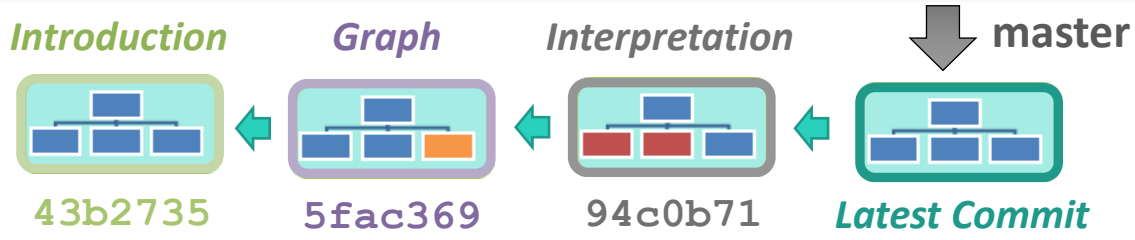
How to investigate what is going on ?

On terminal (within VS Code):

- `git status`
- `git log`
- `git show`
- `git diff`

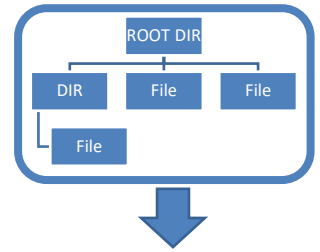
On graphical interface (via RDP during course):

- `git difftool`
- `gitk`



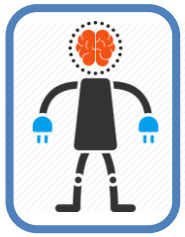
What is exactly a commit

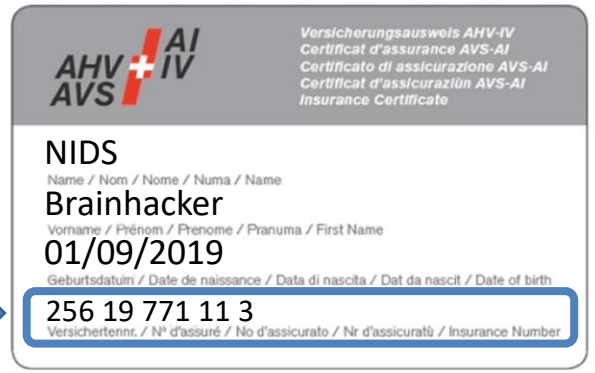
- All files and directories, represented by a single **checksum** (SHA1)
- Previous (parent) commit **checksum**
- Commit description: **message**
- **Author**
- **Date**



cf23df2207d99a74fbe169e3eba035e633b65d94

0de8c7f2bce77531a313
077b687f174e8e49169b





256	19	771	11	3
From name first letters	Year of birth	From sex and birth month & day	« Uniquizer » and nationality	Control number

```
git init
git add
git commit
```

BASIC WORFKLOW EXAMPLE – git init

Create a "professional" (git-tracked) version of our bash greeting project

```
> mkdir pro_hello
> cd pro_hello
> git init
Initialized empty Git repository in /home/brainhacker/pro_hello/.git
> ls -A
```

List all files, including "hidden" files starting with a dot (.)

```
.git
> echo "A greeting program (no program yet)" > README.md
> touch hello
```

Create an empty file where will write the future bash code

```
> git status
On branch master
```

The default branch (more on branches later in this course)

```
No commits yet
```

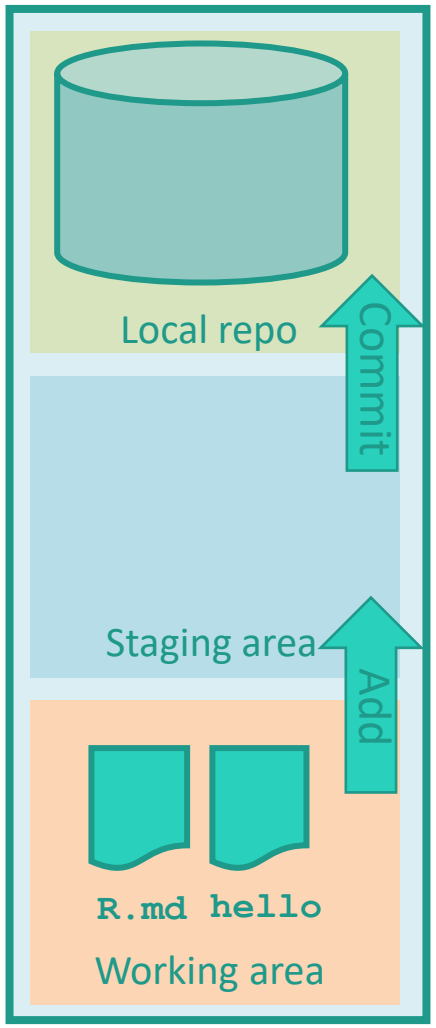
We did not make any commit

```
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        README.md
        hello

nothing added to commit but untracked files present (use "git add" to track)
```

Git detected one new file, which was not promoted to the staging area (area of things "to be committed").

Use `git add` to add it. **`git`** will then track this file, i.e. detect any modification made to it.

Local environment



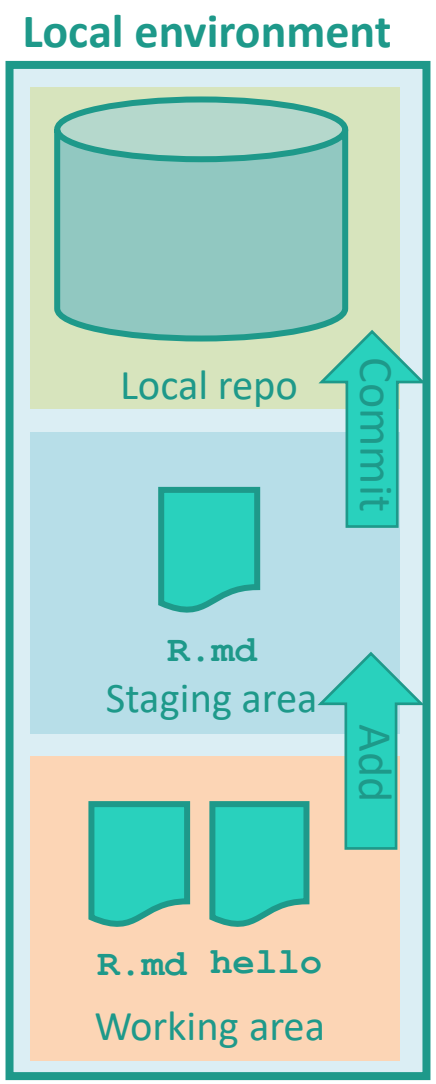
BASIC WORFKLOW EXAMPLE – git add

Promoting to the staging area

```
> git add README.md
> git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   README.md

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        hello
```

Current changes that we can commit ("to be committed")

Still one file in working area, we will stage it after we finish editing it

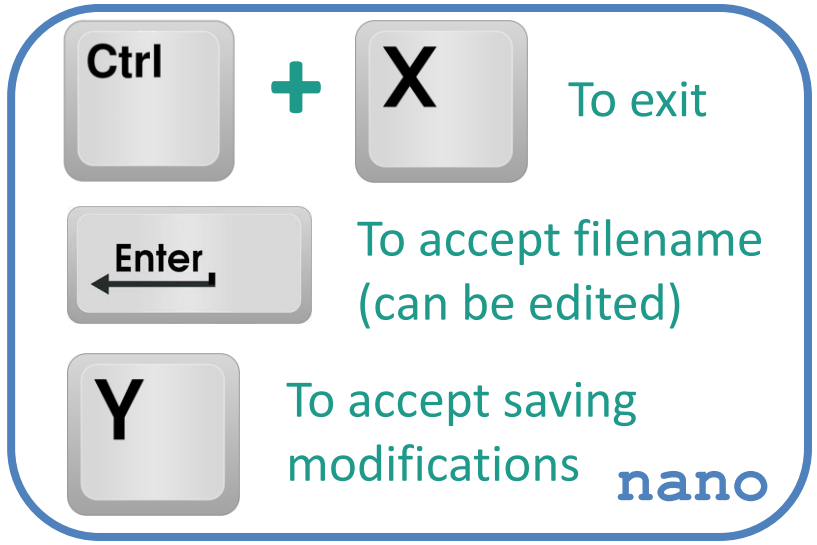


BASIC WORFKLOW EXAMPLE – first git commit

Promoting to a commit in the local repository

Commit messages are essential to understand the history of the codes.
You can provide a message in two ways:

- for changes which require a detailed explanation use `git commit` without options to start an editor (`nano` by default)
  - the first line is the commit message summary / subject
  - skip a line and write the commit message body (as much as you want, with as many blank lines as you want)
- for changes which require just a short commit message use `commit -m "short message description"`

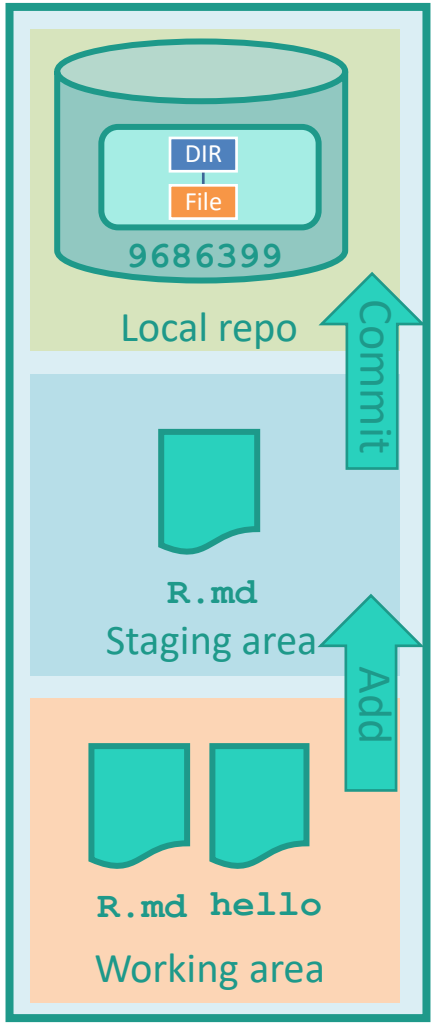



```
> git commit -m "Initialize project with short description"
```

Commit ID

```
[master (root-commit) 9686399] Initialize project with short description
 1 file changed, 1 insertion(+)
 create mode 100644 README.md
```

Non executable file (u-x)



BASIC WORFKLOW EXAMPLE – second git commit

Promoting to the staging area then to a commit in the local repository

Edit the file `hello` with Visual Studio Code (File → Open)

```
#!/bin/bash
#
# Program to greet Leila

echo "Hi Leila"
```

> `chmod u+x hello` Add execution permissions to the `hello` program

Edit the file `README.md` with Visual Studio Code (File → Open)

```
A greeting program (no program yet)
```

> `git add .` Add all untracked files to staging area

> `git commit` This displays the `nano` editor to enter the commit description (first line: commit subject, body after blank line: commit details)

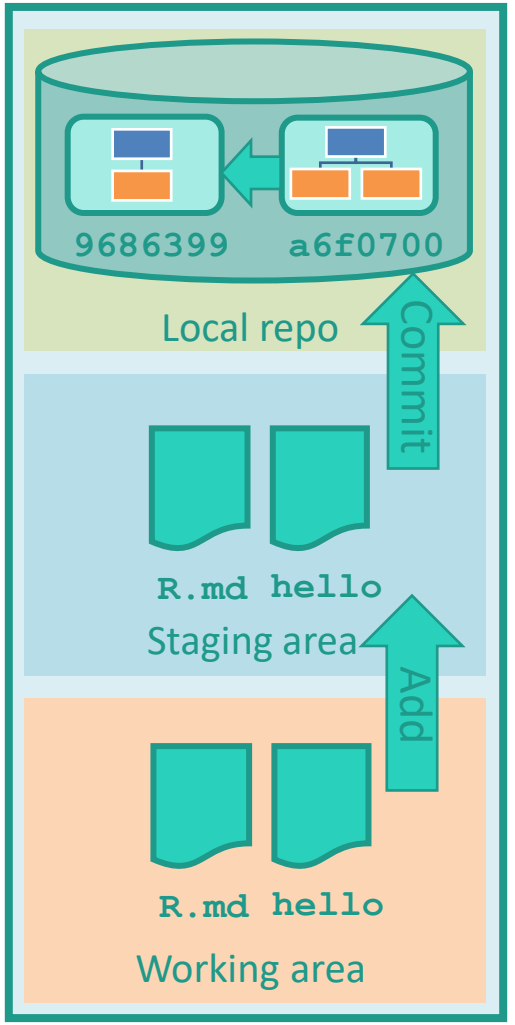
Commit ID

```
[master a6f0700] Add main file to greet people
 2 files changed, 6 insertions(+), 1 deletion(-)
 create mode 100755 hello
```

Executable file (u+x)

Commit message example

```
Add main file to greet people

Print a greeting for a predefined name (Leila).
Update the main README to indicate what the program does.
```



BASIC WORFKLOW EXAMPLE – examining changes history

How to look at whole history ?

```
> git log --summary
```

```
commit a6f0700e513cf2a8e91c575b2bb6dc5d317105e9 (HEAD -> master)
Author: NIDS instructor <methods@fcbg.ch>
Date:   Tue Oct 13 16:15:35 2020 +0000

    Add main file to greet people

    This version prints a greeting for a predefined name (Leila). The main
    README was updated to indicate the program has now a working version.

 create mode 100755 hello
```

Commit IDs

Executable file (u+x)

```
commit 9686399324f8bd494cb3321e52e1fcdc2e7b5264
Author: NIDS instructor <methods@fcbg.ch>
Date:   Tue Oct 13 14:42:38 2020 +0000

    Initialize project with short description

 create mode 100644 README.md
```

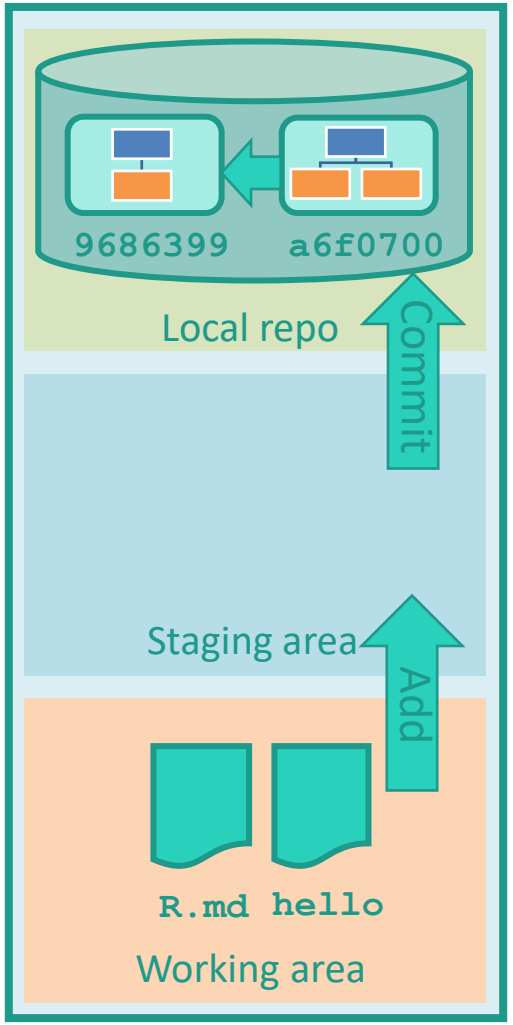
Non executable file (u-x)

```
#!/bin/bash
#
# Program to greet any
# person given their name

name="Leila"

echo "Hi ${name}"
```

Commit defining branch master.

HEAD shows active branch is master

Local environment



3rd commit (to make the history more interesting): the program was modified with a variable added

```
> git log --oneline
```

More concise than `git log --summary`

```
cddb1cb (HEAD -> master) Generalize greeting program by adding a name variable
a6f0700 Add main file to greet people
9686399 Initialize project with short description
```

git init	git status	git diff
git add	git log	git difftool
git commit	git show	gitk

HNP METHODS & DATA

campus biotech

Fondation Campus Biotech Geneva

BASIC WORFKLOW EXAMPLE – examining changes history

How to look at difference between two commits ?

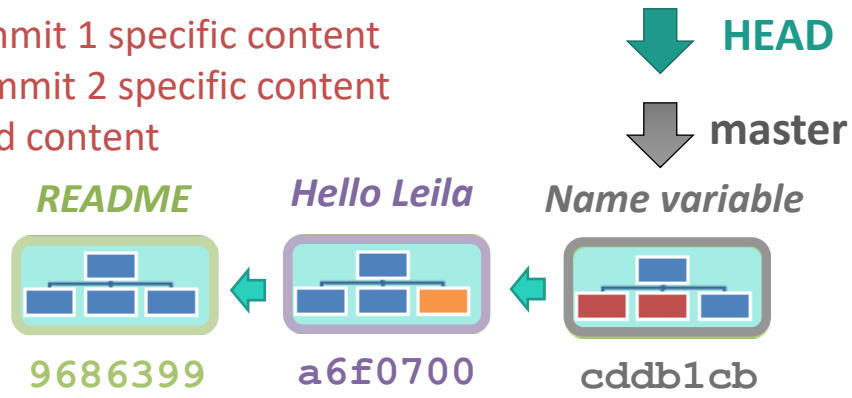
```
> git diff <SHA commit 1> <SHA commit 2>
```

```
> git diff a6f0700 cddb1cb
```

```
diff --git a/hello b/hello
index 815bb9b..4850f2f 100755
--- a/hello
+++ b/hello
@@ -2,4 +2,6 @@
 #
 # Program to greet Leila

-echo "Hi Leila"
+name="Leila"
+
+echo "Hi ${name}"
```

Differences for all files. Add filename as last argument to look at only a specific file (e.g. `git diff a6f0700 cddb1cb hello`)

For the file `hello`:
"-" sign in front of commit 1 specific content
"+" sign in front of commit 2 specific content
no signs for unchanged content




```
diff --git a/README.md b/README.md
index f3b458b..6777973 100644
--- a/README.md
+++ b/README.md
@@ -1 +1 @@
-A greeting program
+A greeting program generalizing to any name
```

For the file `README.md`:
"-" sign in front of commit 1 specific content
"+" sign in front of commit 2 specific content
no signs for unchanged content

```
> git difftool a6f0700 cddb1cb hello
```

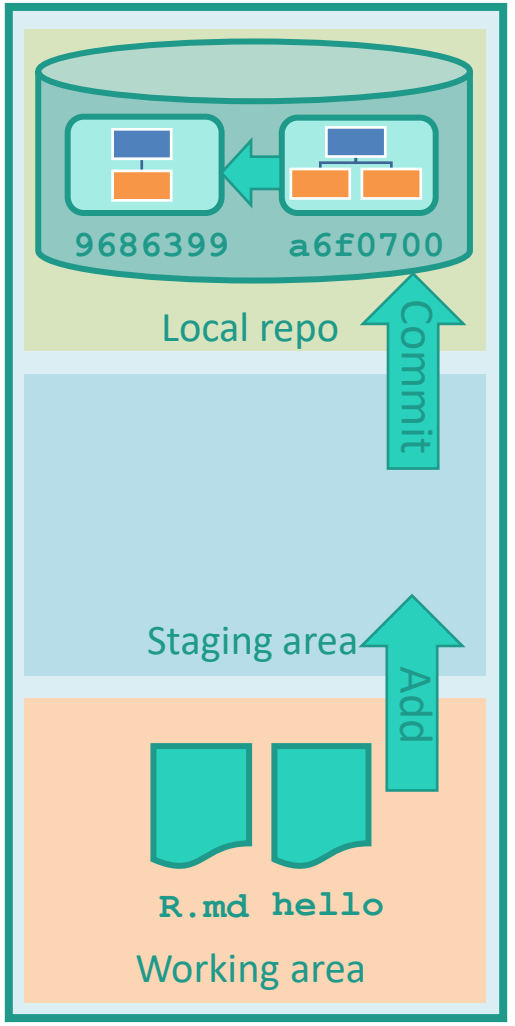
To use graphical interface

To setup your system to use your favorite editor use the following commands:

```
git config --global diff.tool vscode
git config --global difftool.vscode.cmd "code --wait --diff $LOCAL $REMOTE"
```

Note: in all previous commands using cddb1cb or HEAD is the same as HEAD points to that commit

Local environment



HNP METHODS & DATA

git init	git status	git diff
git add	git log	git difftool
git commit	git show	gitk

campus biotech

Fondation Campus Biotech Geneva

USING THE CODE EDITOR

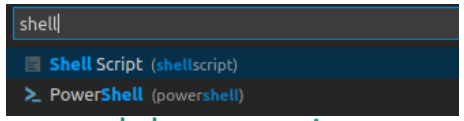
Choose a code editor

Password: braincode!

- Visual Studio (VS) Code

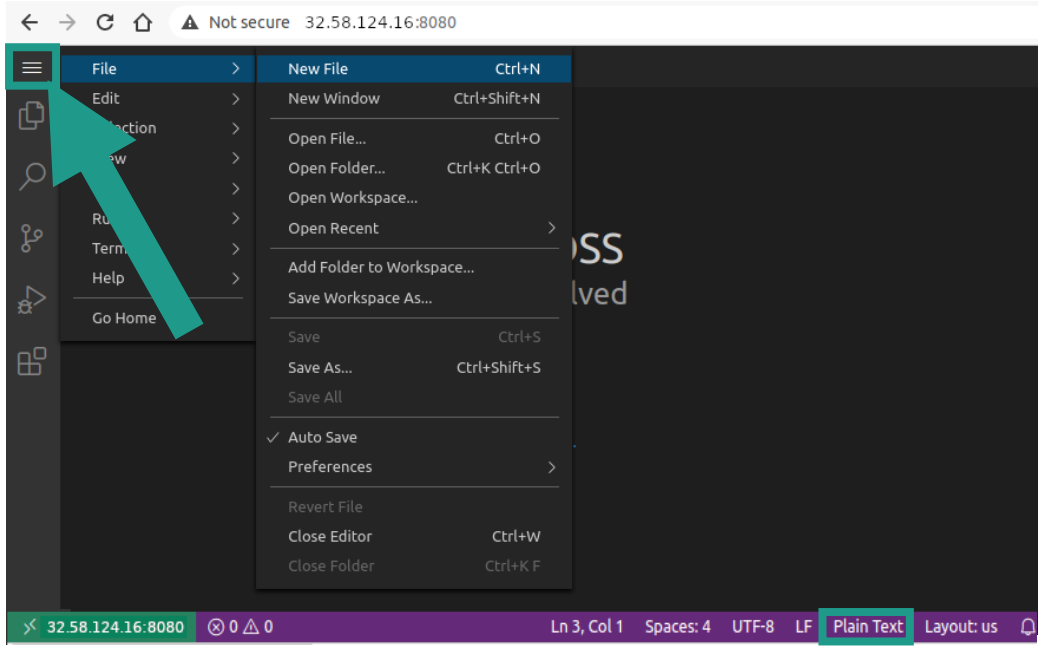
The IP you received by mail

1. Open your web browser at `W.X.Y.Z:8080`
2. Then File → New File
3. Click on “Plain Text” and search for “Shell Script” in the search box popping up



4. To save the file: File → Save As

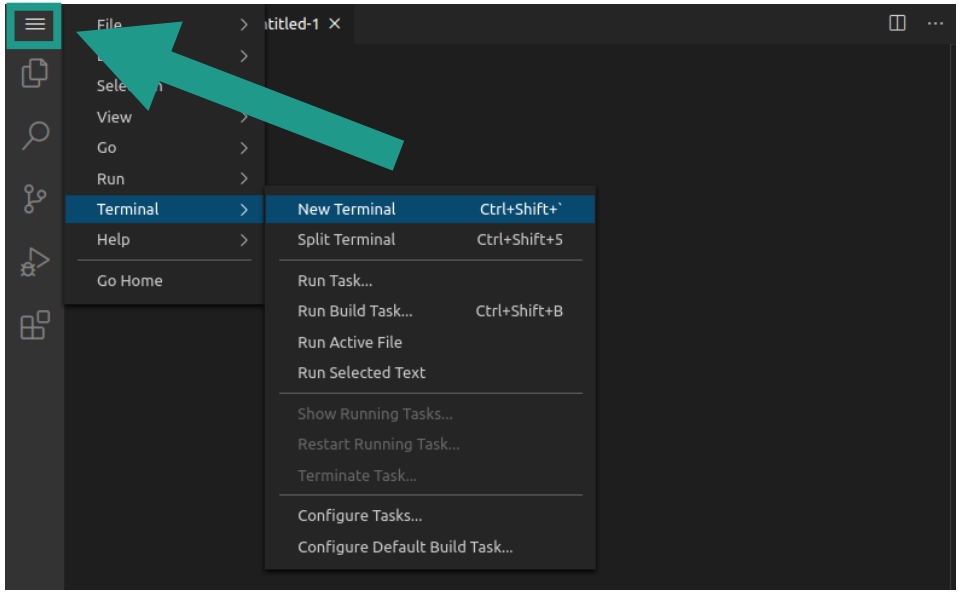
Note: Go up in the filesystem tree by clicking on “. .”



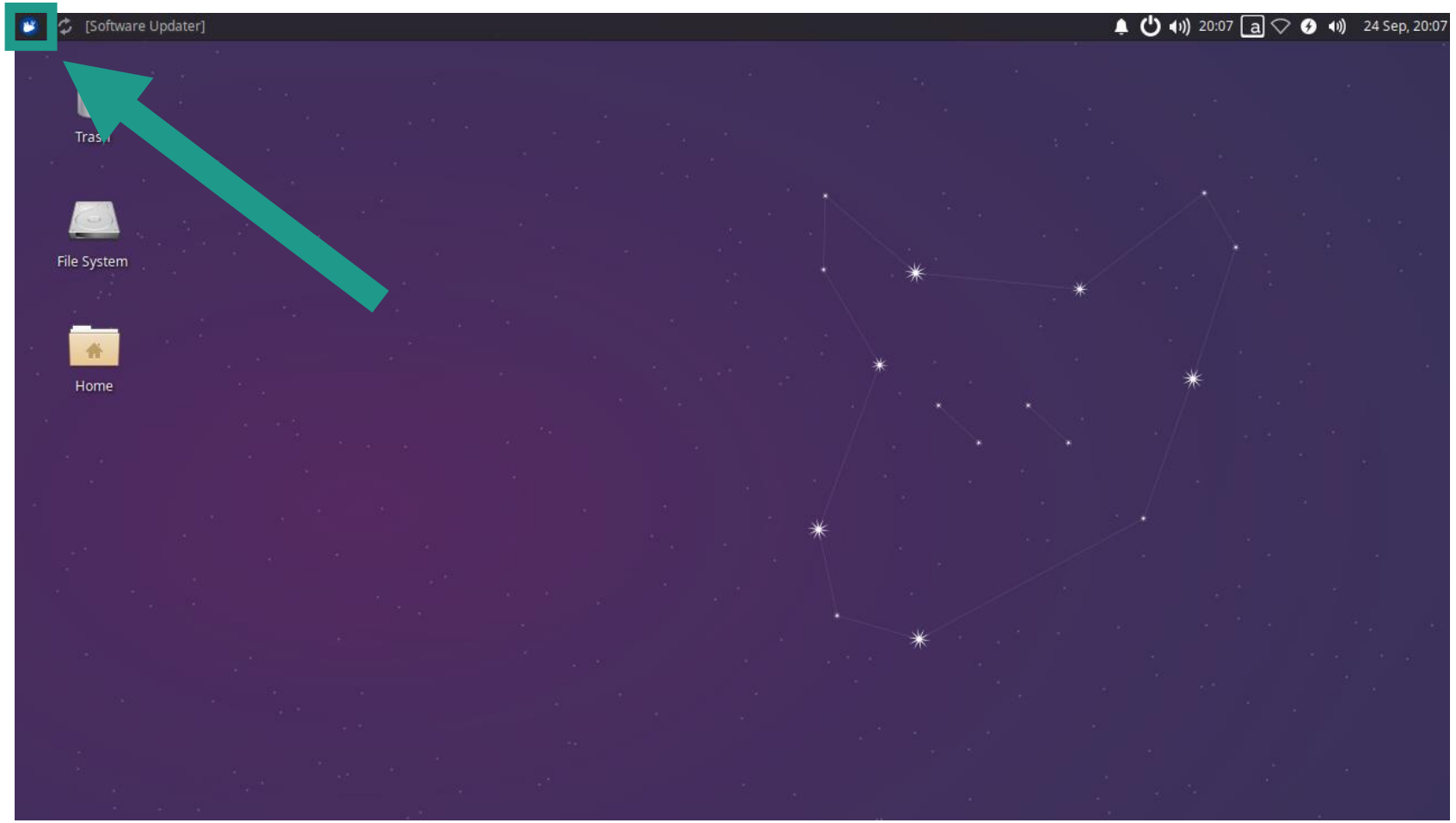
shebang `#!/bin/bash` absolute path to interpreter

5. If no terminal on the bottom half of the screen: Terminal → New Terminal
6. Make sure the script you saved is executable, by typing the following in the terminal:

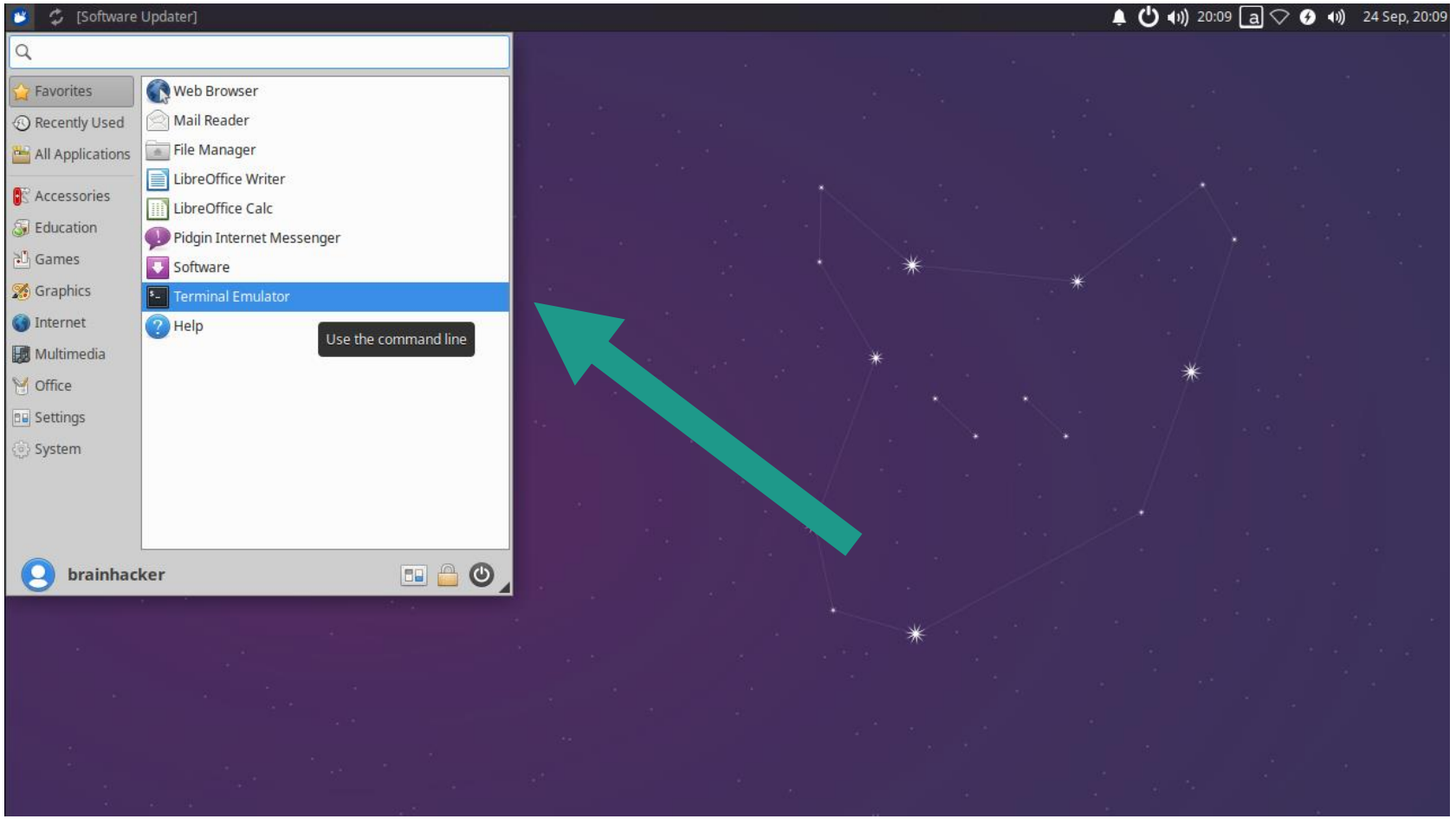
```
chmod u+x <path_to_your_script>
```



USING RDP TO CONNECT TO THE VM & START TERMINAL



USING RDP TO CONNECT TO THE VM & START TERMINAL



OR

Ctrl + Alt + T

Ctrl + X To exit

Enter To accept filename (can be edited)

Y To accept saving modifications

nano

GIT LAB 1

Configure your name and email required for commits

Create a directory `test_hello` in your home dir

Go inside that directory and initialize a local git repository

Create a text file `README.md` and then check git status

Add `README.md` to the staging area and check status again

Commit the set of changes and check status a third time

You saw the whole promotion model and did your first commit !

Create an `hello` bash script inside the `test_hello` directory.
The script should just print the name “Leila”. Test it.

Edit `README.md` to indicate what your project can do.

Add your changes to staging area. Then create a commit.

Modify `hello` so that is uses a variable. Stage and then commit.

Look at the whole project history (try both exhaustive and concise history)

Examine the difference between your last two commits using their SHAs. Then do the same with `difftool` (in the VM using RDP).

```
git config --global user.name NAME  # use quotes for NAME
git config --global user.email EMAIL
mkdir DIR # create directory DIR
cd DIR # change to directory DIR
git init # initialize a local repository
echo TEXT >> FILE # print TEXT to FILE (this creates the
                  # file if it doesn't exist already)
nano FILE # basic editor to edit FILE
# You may as well use Visual Studio code to edits files!
git status # give git status of local environment
git add FILE # stage FILE
git add . # stage all eligible files
git commit # commit staged files to local repo
           # message will be written in an editor
git commit -m MESSAGE # commit staged files to local repo
                      # with message MESSAGE
git log --summary # get exhaustive commit history
git log --oneline # get concise commit history
git diff REF1 REF2 # diff of two commits with SHA refs
                   # REF1 and REF2
git difftool REF1 REF2 # graphical diff
gitk # global graphical interface to see history
```

HNP METHODS & DATA

campus biotech

Fondation Campus Biotech Geneva

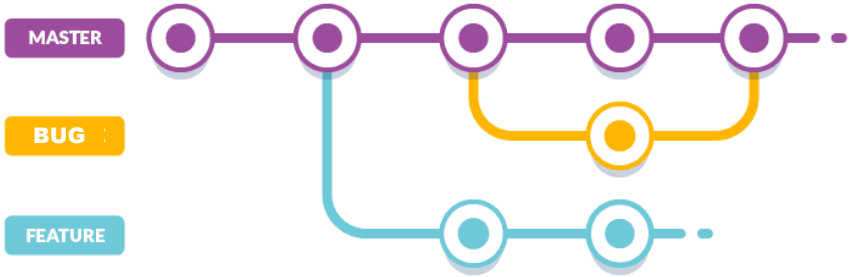
GIT BRANCHES & SIMPLEST MERGE

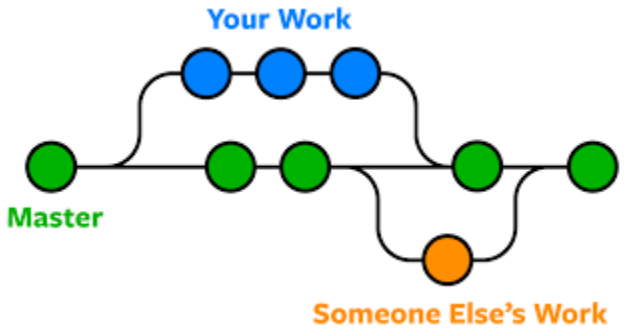
What is a branch?

- → Conceptually: a branch is a line of development (a sequence of commits)
- → Technically: it is a reference (i.e. a pointer) to a commit

For what branches are used for?

- → Topic branch to test an idea of code change
- → Branch to fix a bug
- → Feature branch to develop a new code feature
- → Integration branch to integrate different development branches together





GIT BRANCHES & SIMPLEST MERGE

What is a branch?

- → Conceptually: a branch is a line of development (a sequence of commits)
- → Technically: it is a reference (i.e. a pointer) to a commit

How to build a branch?

- → Create a new pointer to the current commit
  `git branch topic`
- → Activate (i.e. `checkout`) that branch
  `git checkout topic`
- → Generate commits

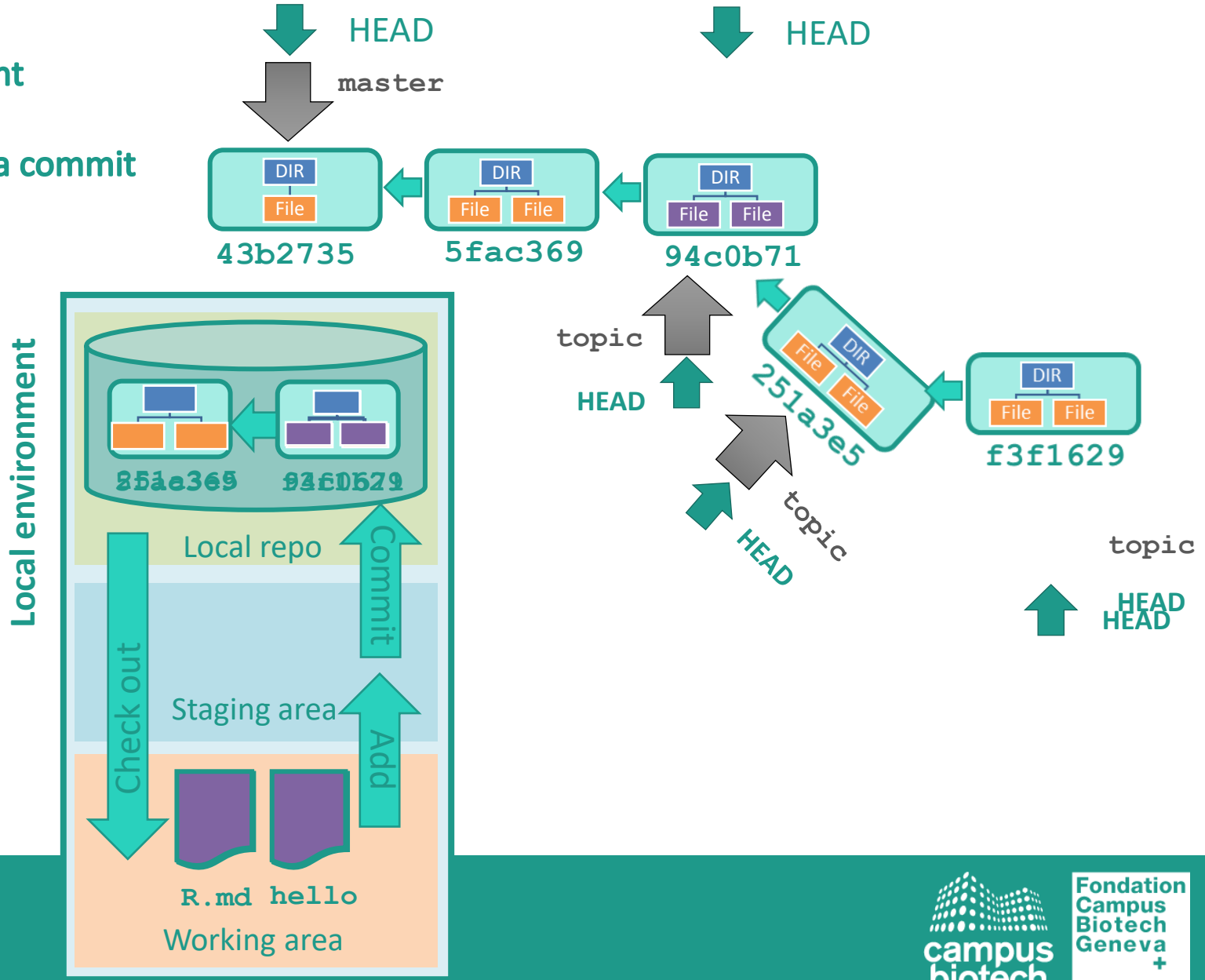
How to switch to another branch?

- → Simply activate it

```
git checkout master
git checkout topic
git checkout master
```

Note: it can be useful to create a branch at a commit of interest <commit ID>. This can be done with:

```
git branch topic <commit ID>
```



GIT BRANCHES & SIMPLEST MERGE

How to merge a branch (e.g. called `topic`) into the main (typically `master`) branch?

Simplest situation: no changes made on main branch

- → **the `master` branch is "paused" compared to the `topic` branch (it is a subset of the `topic` branch)**
- → **the `master` branch pointer simply has to be fast-forwarded to the `topic` branch**

```
> git checkout master
> git merge topic
```

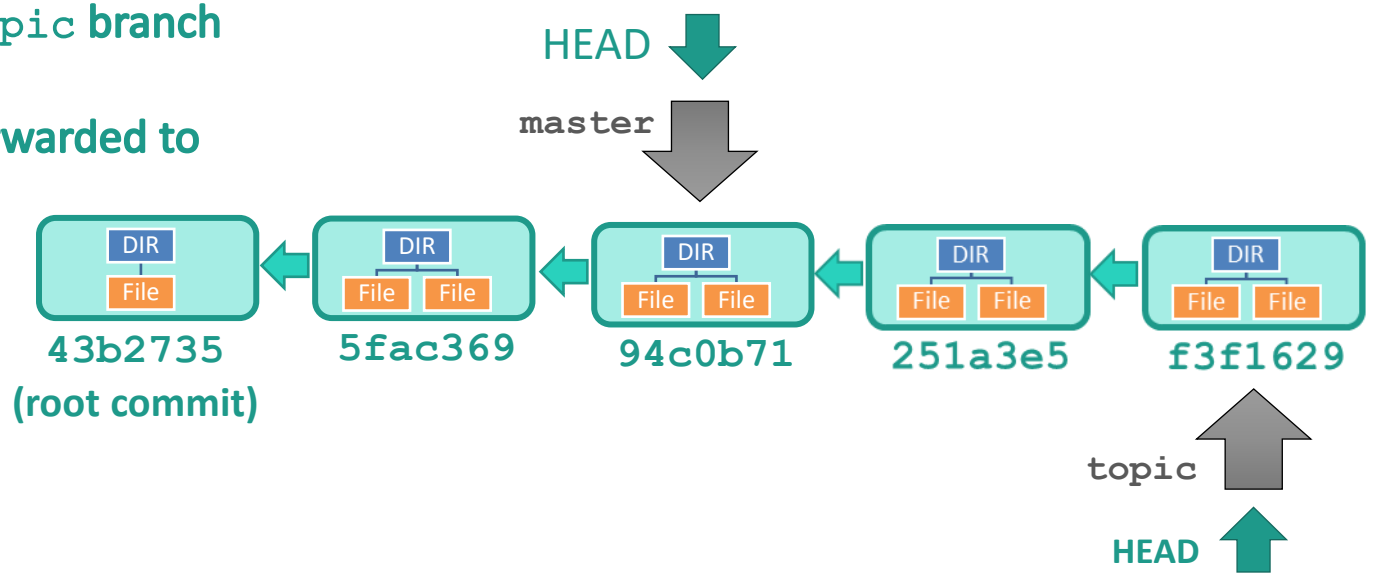
- → **To delete a branch**

```
git branch -d MY_BRANCH
```

```
> git branch -d topic
```

- → **To list branches**

```
git branch [-v]
```



GIT LAB 2 – branch to correct a bug

An embarrassing bug is present in your code. The demonstration name should have been “Leia” not “Leila”. Let’s try correcting the bug without touching at the working code in branch `master`.

- Make sure you are in the `test_hello` directory
- Create a new branch called `bugfix` and checkout that branch
- Make sure you are on the `bugfix` branch by using `git branch` (a * sign should be in front of the branch name)
- Edit hello (and if required `README.md`) to change “Leila” to “Leia” (or whatever modification fits your current code)
- Stage the modified file after making sure it runs fine.
- Create a commit with an informative message.
- Go to the `master` branch (then double-check with git branch)
- Merge your `bugfix` branch into master, and check git status
- Check that your new master branch is bug free and the program now runs as expected
- Have a look at the whole commit history (for long history, press “SPACE” to display next page and “q” to quit)
- Delete the bugfix branch

```
pwd # print path of current directory
cd DIR # change to directory DIR
git branch MY_BRANCH # create branch MY_BRANCH
git checkout MY_BRANCH # checkout branch MY_BRANCH
git branch # list branches (* in front of active branch)
git status # give git status of local environment
git add FILE # stage FILE
git add . # stage all eligible files
git commit # commit staged files to local repo
           # message will be written in an editor
git commit -m MESSAGE # commit staged files to local repo
                      # with message MESSAGE
git merge MY_BRANCH # merge MY_BRANCH into current branch
git log --summary # get exhaustive commit history
git log --oneline # get concise commit history
git branch -d MY_BRANCH # delete the branch MY_BRANCH
```

COURSE SUPPORT

SLACK (iords2021.slack.com)

- Course main channel: `#general`
- Topic channels: `#linux`, `#linux-capstone`, `#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 18th: 20-min slots on Friday

→ 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