

Biomedical Wearable Technologies for Healthcare and Wellbeing

Git

A.Y. 2021-2022

Giacomo Cappon



Outline

- Part 1: Version Control Systems & GIT
- Part 2: Playing around with GIT
- Part 3: The working tree
- Part 4: Remote repositories and best practices
- Part 5: Our development environment

Outline

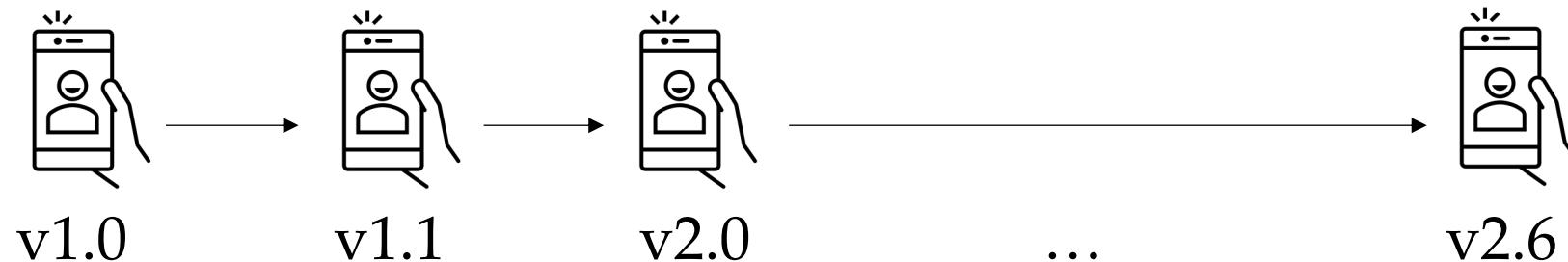
- **Part 1: Version Control Systems & GIT**
- Part 2: Playing around with GIT
- Part 3: The working tree
- Part 4: Remote repositories and best practices
- Part 5: Our development environment

New features means new code

- When you are developing/maintaining a software, sometime new features need to be added

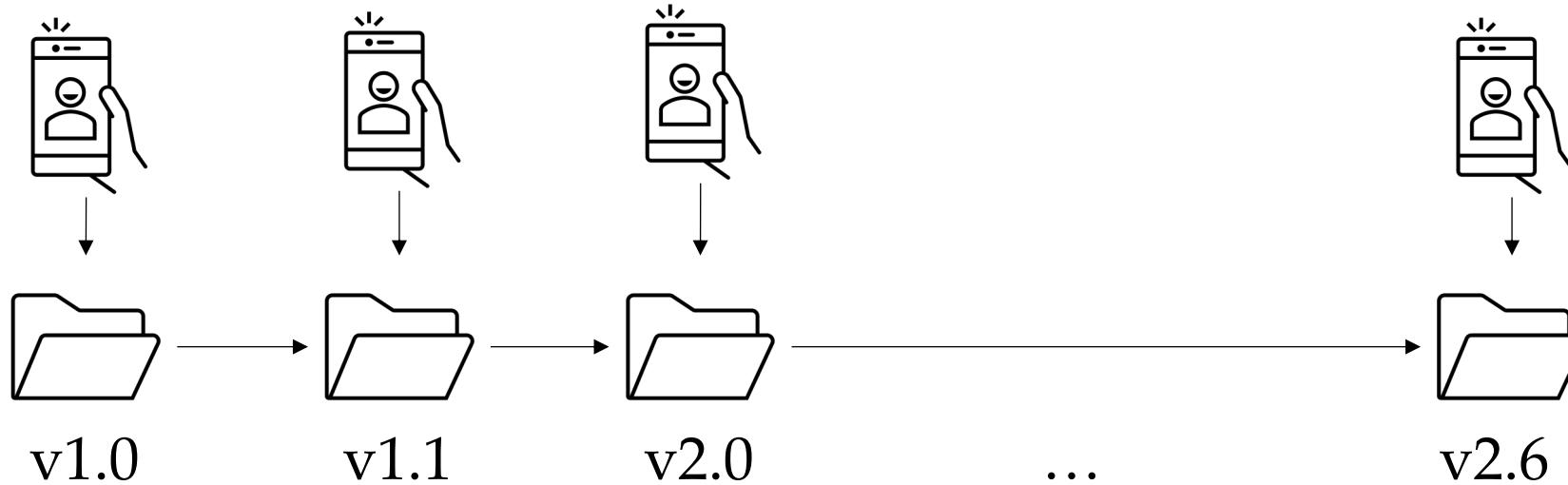


- This translates in multiple software (AKA code) versions



Dealing with code versions: The naïve solution

- How to deal with multiple code versions?
- The simple (naïve) solution is to “make a folder for each code version”



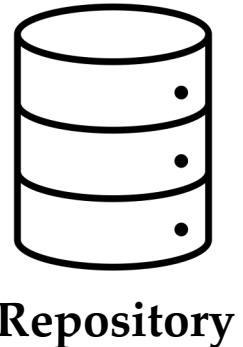
- **Question:** What are the problems here?

Problems of the “multiple folders” approach

- Very error prone approach
- Teams are not able to collaborate (have to manually send the code, who to blame for bug introduction? How to merge changes?)
- Lot of disk space required
- “Where are my code? Which is which?” situations
- ...

The gold standard: Version control systems

- The gold standard to deal with this issue is to use a version control system (VCS)
- VCS are softwares that allow to
 - track the code “history”
 - work together on the same code
 - jump between code versions
 - fix bugs efficiently
 - ...
- Everything (code changes, contributors, deltas,...) is stored in a dedicated repository
- A repository can be stored locally or remotely (in remote repository databases) and connected to the code contributor machine



Repository

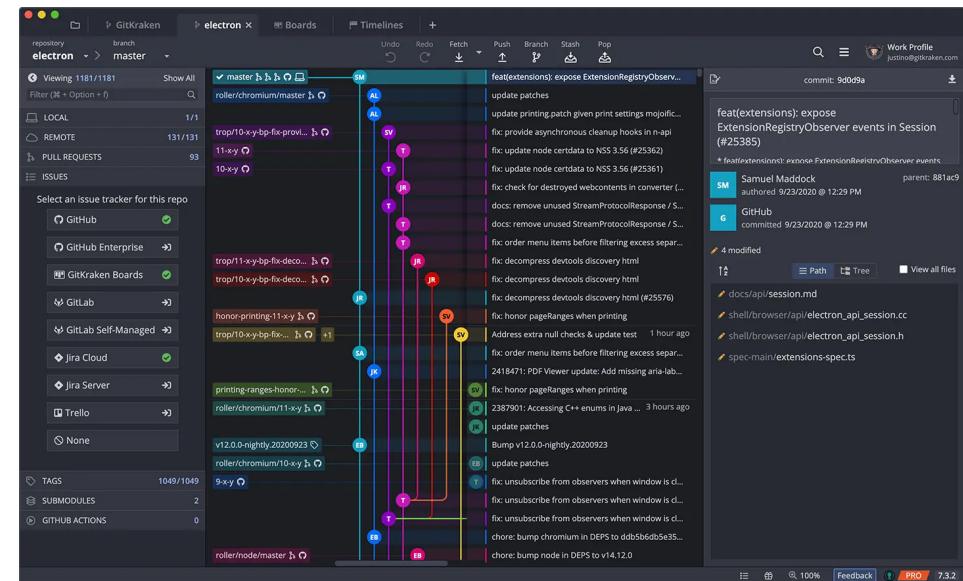


- The most popular VCS is GIT. Created by Linus Torvalds in 2005, it is a free, open source, fast, and scalable solution.
 - We will use GIT in this course

- GIT can be used via
 - Fancy Graphical User Interface (GUI)
 - Old-school terminal command line

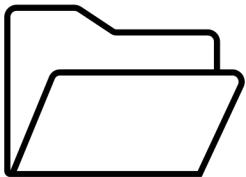
A screenshot of a terminal window on a Linux desktop. The window title is 'MINGW64 /c/Users/User'. The command history shows:

```
MINGW64 /c/Users/User
User@DESKTOP-23CP0AT MINGW64 ~ /Desktop/_MACOSX
$ cd
User@DESKTOP-23CP0AT MINGW64 ~
$ start
$ /usr/bin/startx: line 8: cmd: command not found
User@DESKTOP-23CP0AT MINGW64 ~
```

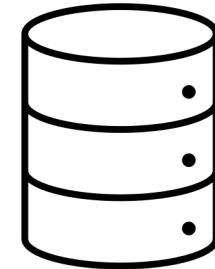


- In this course we will use the command line. Why?
 - GUI have limitations
 - GUI tools are not always available (e.g., remote servers)

GIT workflow



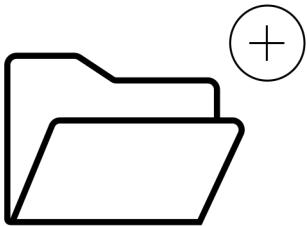
Working directory



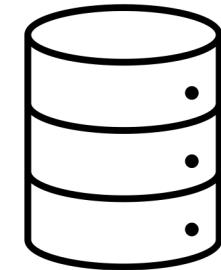
Repository

- **Working directory:** where the code is.

GIT workflow



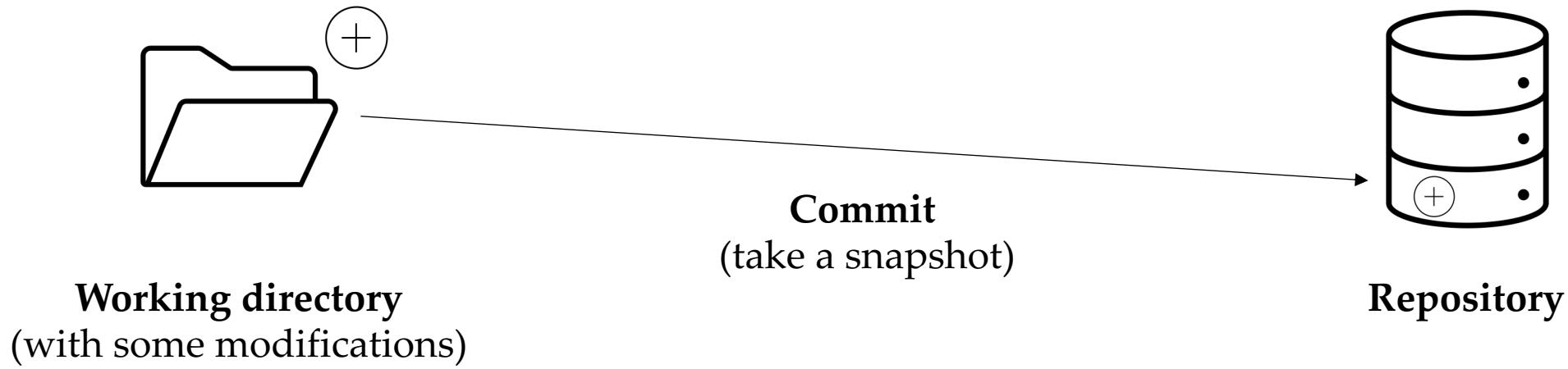
Working directory
(with some modifications)



Repository

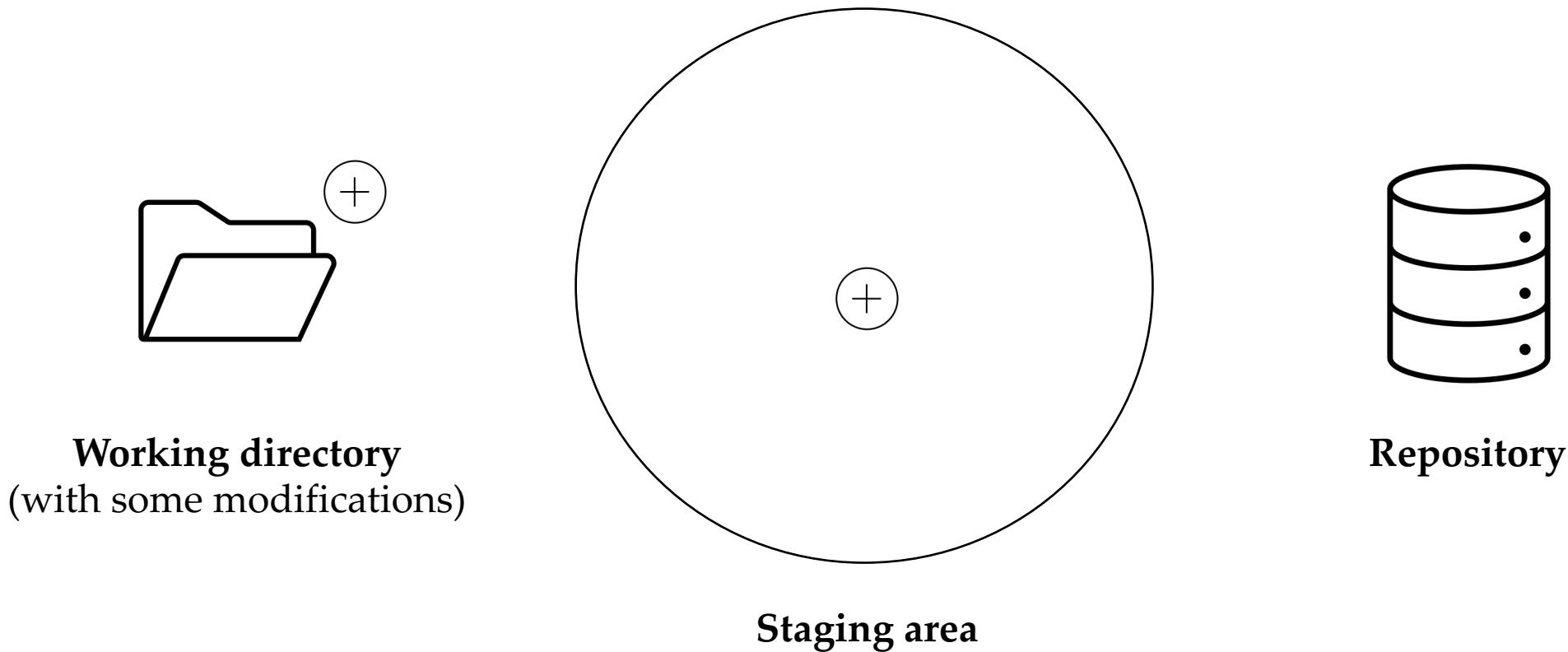
- While you are working, you may decide that the code reached a state that you want to record (might be a new feature, a new release of the code, a bug fix, ...)

GIT workflow



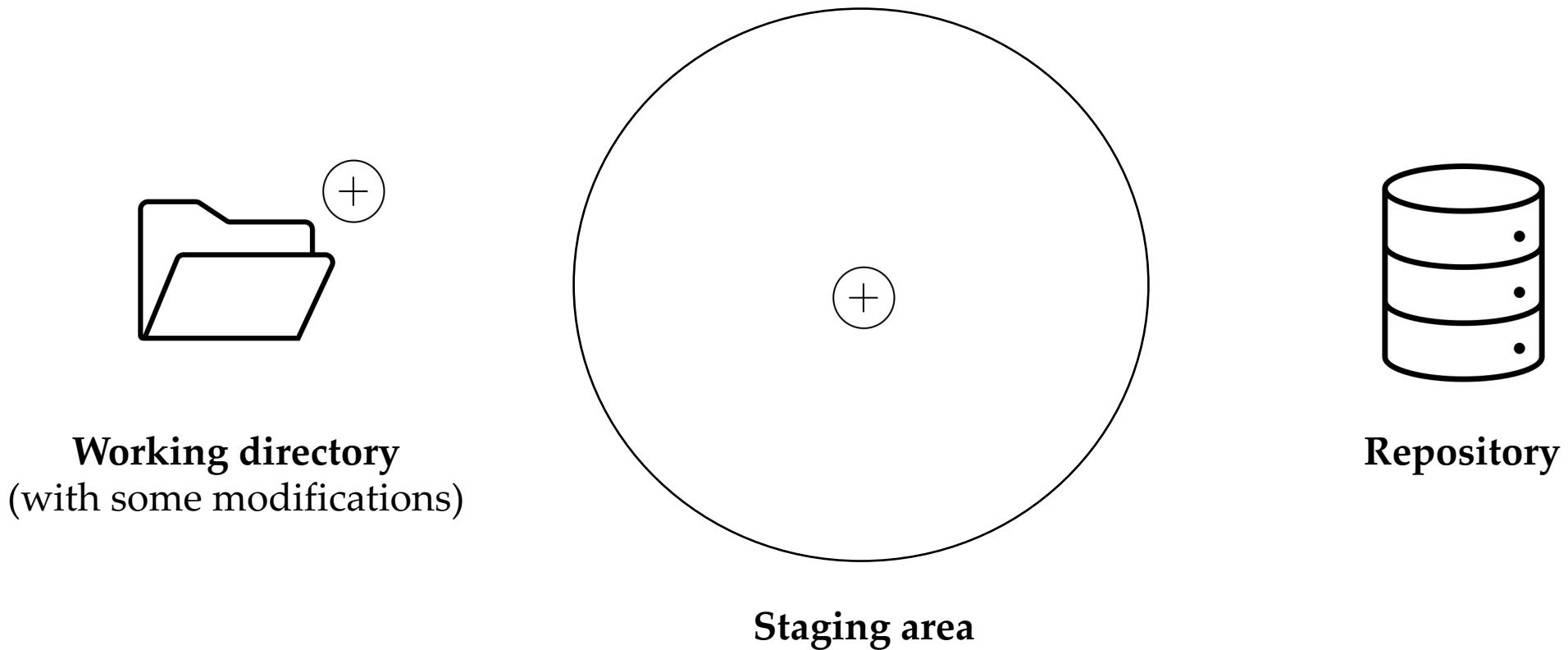
- So, you commit those changes to the repository. Making a commit is like taking a snapshot of the code as it is now.

GIT workflow



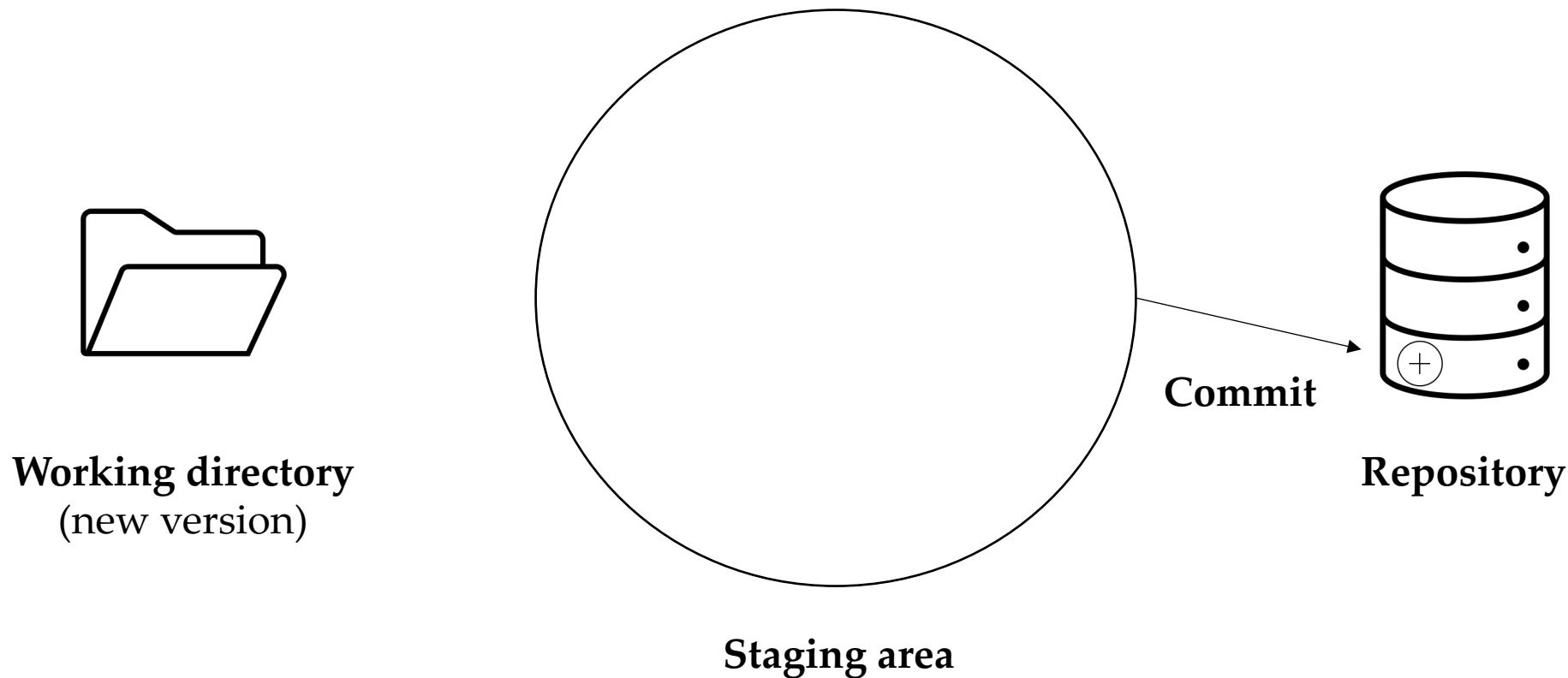
- In GIT, there is a special additional area in between the project folder and the repository: the staging area. It memorizes, while you are working, the changes between the current code and the last commit (in this case $(+)$)

GIT workflow



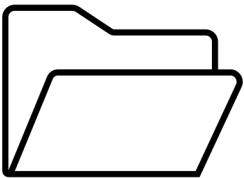
- So, while working, every code modification have to be staged in the staging area.

GIT workflow

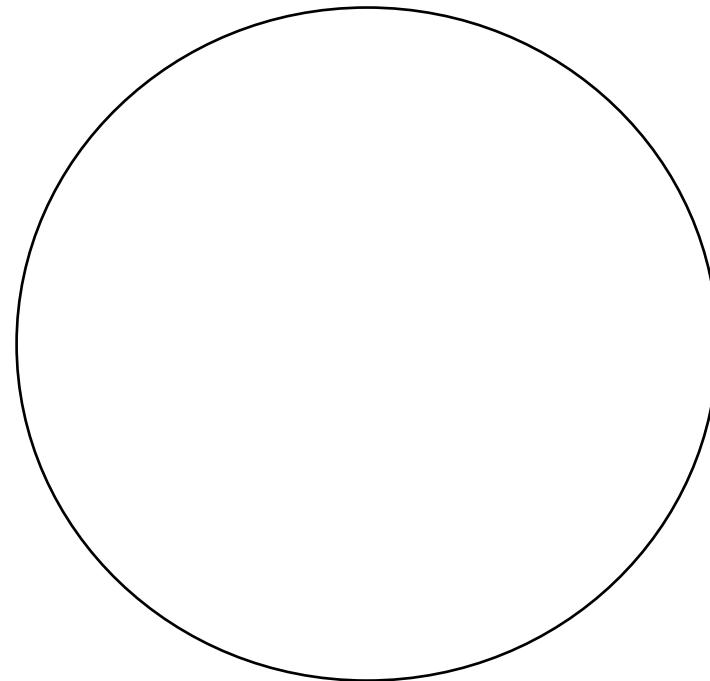


- Then, when ready and tested, those modifications can be committed (or reverted) by the user into the repository.

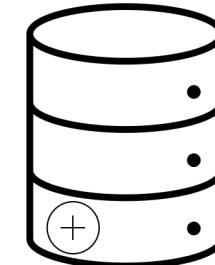
GIT workflow



Working directory
(new version)



Staging area

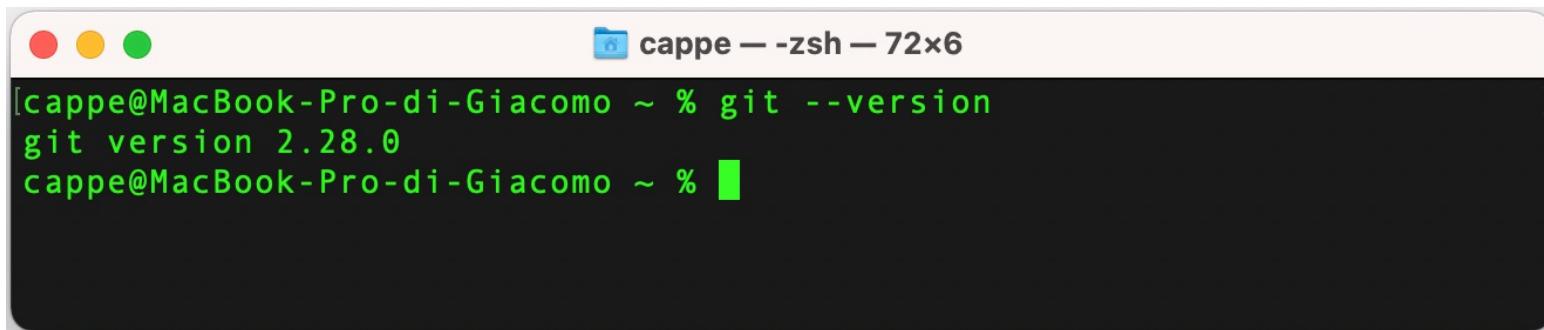


Repository

- This frees the staging area and creates a new code version, which now includes \oplus .

Installing GIT

- First check if it is already installed
 - Open the Terminal > Type “git --version”



A screenshot of a macOS terminal window titled "cappe -- zsh -- 72x6". The window shows the command "git --version" being run in the terminal, and the output "git version 2.28.0" is displayed. The terminal has a dark background with green text.

- If an error appears
 - Go to > <https://git-scm.com/book/en/v2/Getting-Started-Installing-Git>
 - Follow the instructions...
- Install Visual Studio Code (we will use it later)
 - Go to > <https://code.visualstudio.com/download>
 - Follow the instructions...
- I'll give you 15 minutes to do that

Outline

- Part 1: Version Control Systems & GIT
- **Part 2: Playing around with GIT**
- Part 3: The working tree
- Part 4: Remote repositories and best practices
- Part 5: Our development environment

Getting started – Our case of study

- To get familiar with GIT and learn its functionalities we will start with a simple case of study consisting of several steps:
 1. Create a new project folder with a .txt file inside
 2. Create a new repository
 3. Track file history
 4. Commit file in the repository
 5. Check the (new) repository history
 6. Make, check, and commit modifications

Step 1: Create folder and file

- Open the terminal
- Create a directory wherever you want named “git-test” and move inside it

```
>> mkdir git-test && cd git-test
```

- Open some text editor and create a file named “text.txt” containing the following text

```
hello world
```

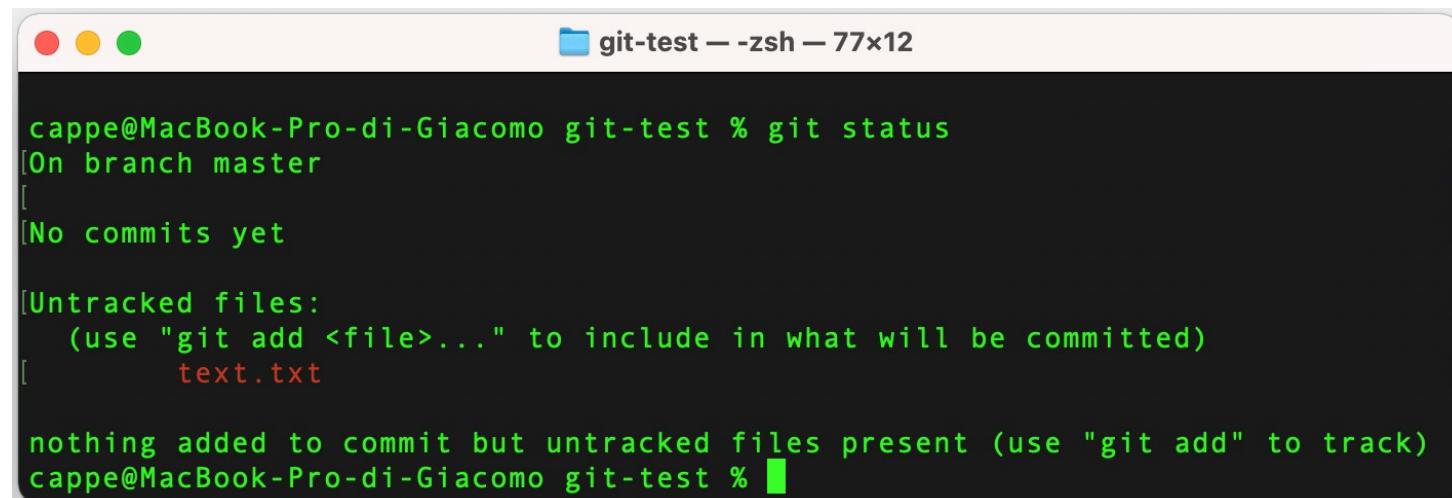
- Save `text.txt`. Now we are ready to play with GIT.

Step 2: Create a new repository

- To create a repository of our project folder, simply write (be sure to be inside “git-test”):

```
>> git init
```

- A new hidden folder called “.git”, will be created
- Now let’s check what we have done using the “git status” command



```
git-test — -zsh — 77x12
cappe@MacBook-Pro-di-Giacomo git-test % git status
[On branch master
[
[No commits yet
[Untracked files:
  (use "git add <file>..." to include in what will be committed)
[      text.txt
[nothing added to commit but untracked files present (use "git add" to track)
cappe@MacBook-Pro-di-Giacomo git-test %
```

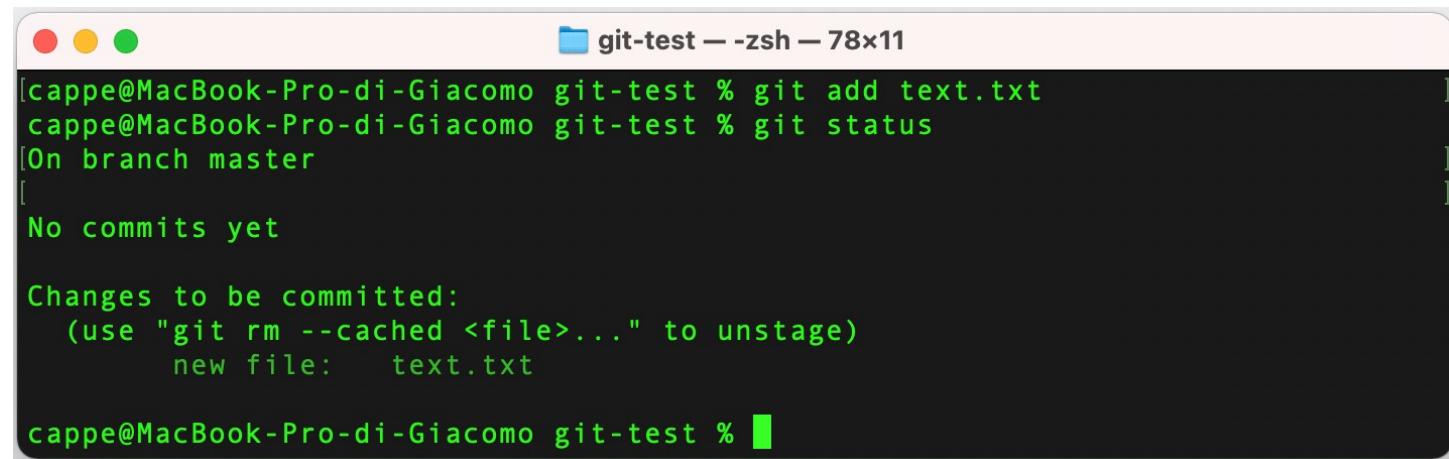
Step 3: Track the file history

- To track and stage “text.txt” use the “git add” command

>> git add text.txt

- Alternatively, you can use “git add *” to add all files in the project folders

- Check the status of the repository



A screenshot of a macOS terminal window titled "git-test -- zsh -- 78x11". The window shows the following command-line session:

```
[cappe@MacBook-Pro-di-Giacomo git-test % git add text.txt
cappe@MacBook-Pro-di-Giacomo git-test % git status
[On branch master
[

No commits yet

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

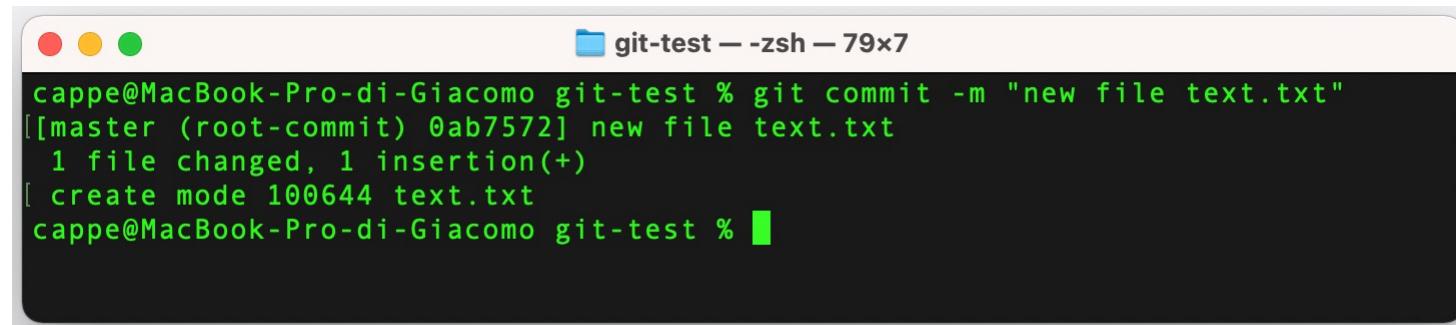
cappe@MacBook-Pro-di-Giacomo git-test % ]]
```

The terminal window has a dark background with light-colored text. It features the standard macOS window controls (red, yellow, green) at the top left. The title bar is "git-test -- zsh -- 78x11". The text area contains the command "git add text.txt", its confirmation, the "git status" command, its confirmation, and the resulting status message. The status message indicates there are no commits yet and a new file named "text.txt" is staged for commit. The prompt "cappe@MacBook-Pro-di-Giacomo git-test %" is at the bottom.

Step 4: Commit the file

- To commit the file

```
>> git commit -m "new file text.txt"
```



A screenshot of a macOS terminal window titled "git-test — zsh — 79x7". The window shows the command "git commit -m \"new file text.txt\"". The output indicates a root-commit at 0ab7572, adding a new file "text.txt" with 1 insertion (+). The file has a mode of 100644. The prompt "cappe@MacBook-Pro-di-Giacomo git-test %" is visible at the bottom.

```
git-test — zsh — 79x7
cappe@MacBook-Pro-di-Giacomo git-test % git commit -m "new file text.txt"
[ master (root-commit) 0ab7572] new file text.txt
  1 file changed, 1 insertion(+)
  [ create mode 100644 text.txt]
cappe@MacBook-Pro-di-Giacomo git-test %
```

- Note that we added a message! This should be meaningful.

Step 5: Check the new history

- Check what we have done using the “git status” command



```
[cappe@MacBook-Pro-di-Giacomo git-test % git status
On branch master
nothing to commit, working tree clean
cappe@MacBook-Pro-di-Giacomo git-test %
```

- Even more things with “git log”



```
[cappe@MacBook-Pro-di-Giacomo git-test % git log
commit 0ab75724246bf8216fadb58df754a848fe132372 (HEAD -> master)
Author: Giacomo Cappon <cappongiacomo@gmail.com>
Date:   Tue Sep 28 17:21:55 2021 +0200

    new file text.txt
cappe@MacBook-Pro-di-Giacomo git-test %
```

Step 6: Make, check and commit modifications

- Now let's modify `text.txt` as

Hello, world!

- And try to do the following operations

- Check the status
- Stage the modifications
- Commit the new modifications

More features: stash and reset

BONUS

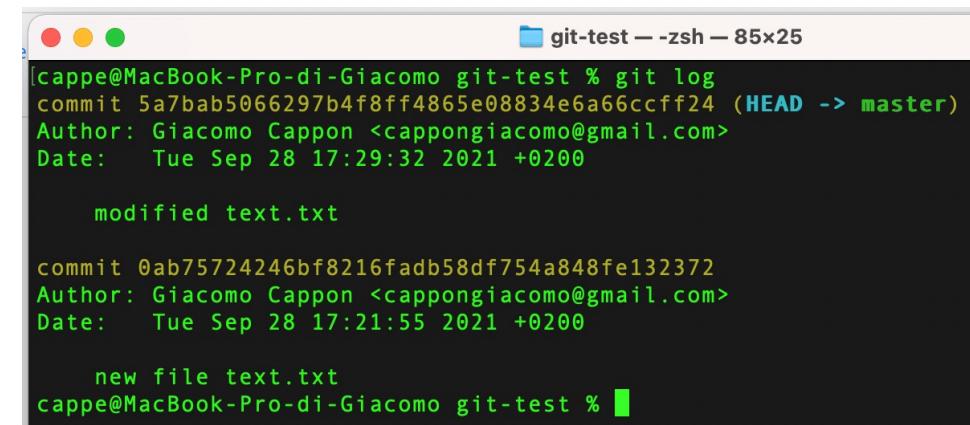
- A useful command is “`git stash`” which stores the current code changes in a stacked temporary working directory and restores the code to the previous valid status.
 - Let’s try the basics (`git stash push`, `git stash pop`)
 - More functionalities in <https://git-scm.com/docs/git-stash>
- Another useful command is “`git reset`” which clears the staging area.
 - Let’s try the basics (`git reset --hard`)
 - More functionalities in <https://git-scm.com/docs/git-reset>

Outline

- Part 1: Version Control Systems & GIT
- Part 2: Playing around with GIT
- **Part 3: The working tree**
- Part 4: Remote repositories and best practices
- Part 5: Our development environment

GIT core: The working tree

- Let's zoom in. What we have actually done?



A screenshot of a terminal window titled "git-test --zsh-- 85x25". The window shows the output of the command "git log". It displays two commits. The first commit, with hash "5a7bab5066297b4f8ff4865e08834e6a66ccff24", modified the file "text.txt". The second commit, with hash "0ab75724246bf8216fadbf58df754a848fe132372", created a new file "text.txt". Both commits were made by "Giacomo Cappon" on "Tue Sep 28 17:29:32 2021 +0200".

```
[cappe@MacBook-Pro-di-Giacomo git-test % git log
commit 5a7bab5066297b4f8ff4865e08834e6a66ccff24 (HEAD -> master)
Author: Giacomo Cappon <cappongiamoco@gmail.com>
Date:   Tue Sep 28 17:29:32 2021 +0200

    modified text.txt

commit 0ab75724246bf8216fadbf58df754a848fe132372
Author: Giacomo Cappon <cappongiamoco@gmail.com>
Date:   Tue Sep 28 17:21:55 2021 +0200

    new file text.txt
cappe@MacBook-Pro-di-Giacomo git-test % ]
```

GIT core: The working tree

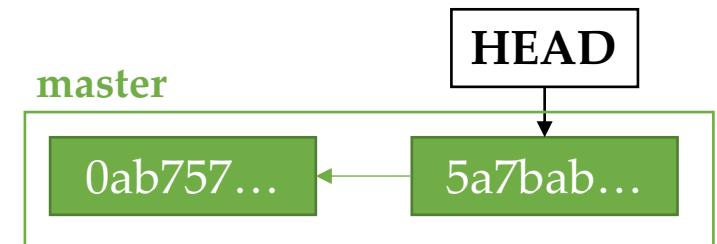
- The repository tracks the history thanks to a tree-based structure called **working tree** which memorizes the filesystem of all code versions.
- The working tree has a main branch called **master**.
- Every commit is associated to a unique hash code.
- GIT moves through the working tree using the **HEAD** pointer.

```
git-test -- zsh -- 85x25
[cappe@MacBook-Pro-di-Giacomo git-test % git log
commit 5a7bab5066297b4f8ff4865e08834e6a66ccff24 (HEAD -> master)
Author: Giacomo Cappon <cappongiamoco@gmail.com>
Date:   Tue Sep 28 17:29:32 2021 +0200

    modified text.txt

commit 0ab75724246bf8216fad58df754a848fe132372
Author: Giacomo Cappon <cappongiamoco@gmail.com>
Date:   Tue Sep 28 17:21:55 2021 +0200

    new file text.txt
cappe@MacBook-Pro-di-Giacomo git-test %
```



Moving through the working tree

- Move through the working tree using the “git checkout” command

```
[cappe@MacBook-Pro-di-Giacomo git-test % git log
commit 5a7bab5066297b4f8ff4865e08834e6a66ccff24 (HEAD -> master)
Author: Giacomo Cappon <cappongiamoco@gmail.com>
Date:   Tue Sep 28 17:29:32 2021 +0200

    modified text.txt

commit 0ab75724246bf8216fadbf754a848fe132372
Author: Giacomo Cappon <cappongiamoco@gmail.com>
Date:   Tue Sep 28 17:21:55 2021 +0200

    new file text.txt
[cappe@MacBook-Pro-di-Giacomo git-test % git checkout 0ab757
Note: switching to '0ab757'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

  git switch -c <new-branch-name>

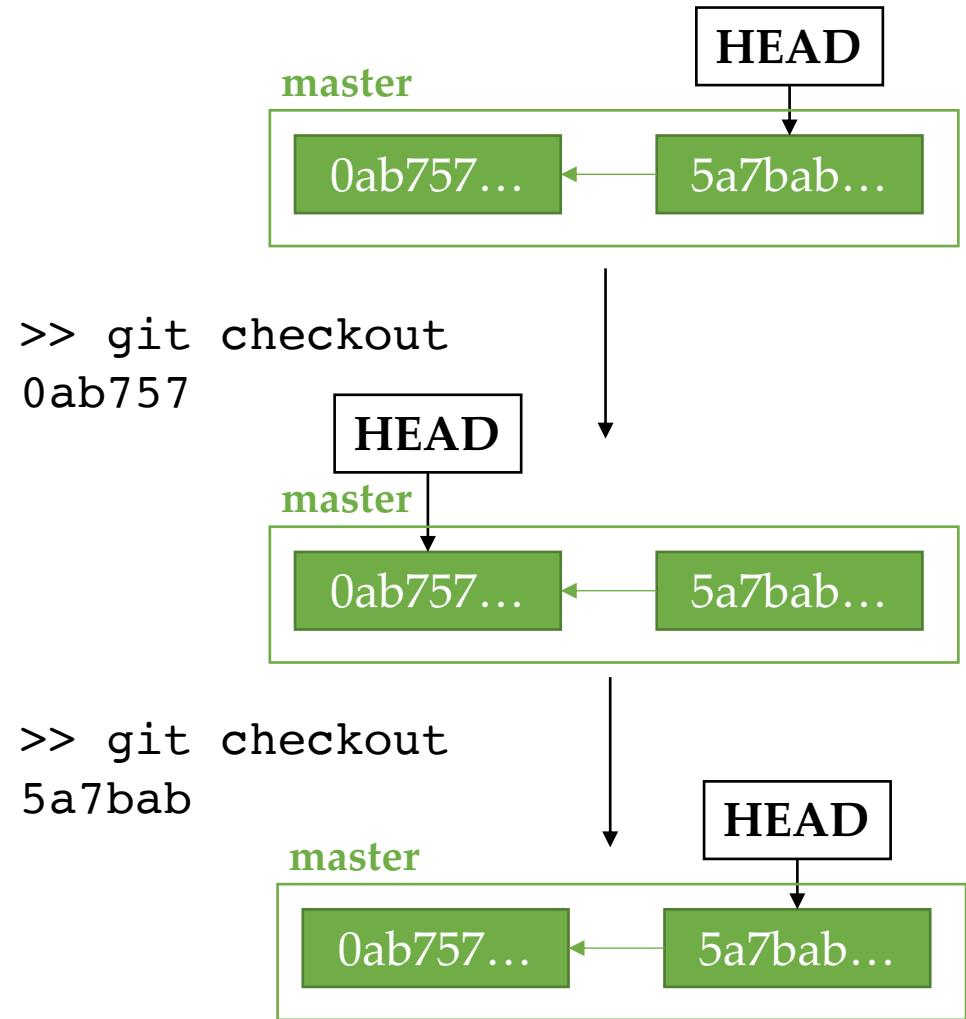
Or undo this operation with:

  git switch -

Turn off this advice by setting config variable advice.detachedHead to false

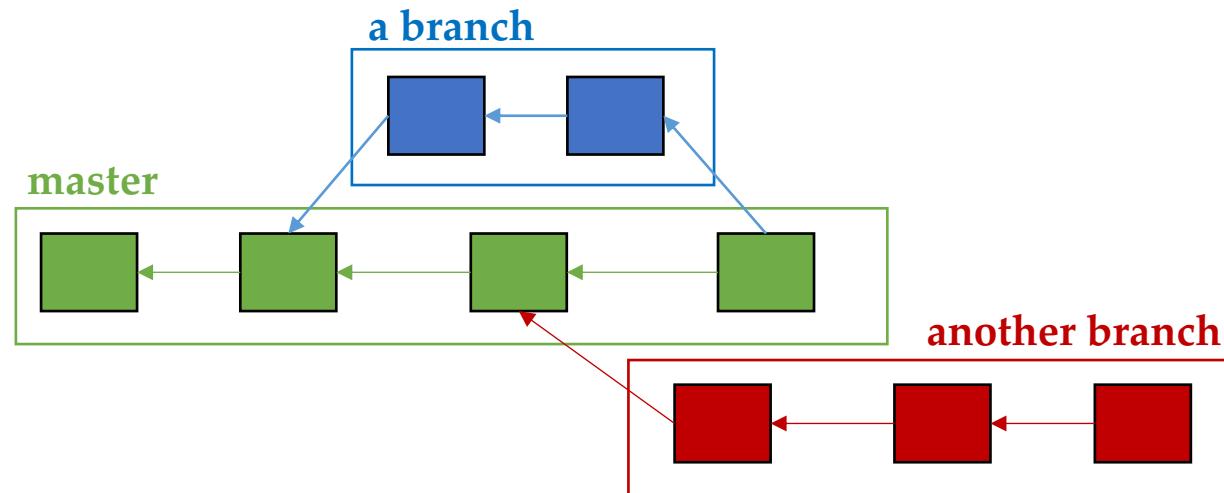
HEAD is now at 0ab7572 new file text.txt
[cappe@MacBook-Pro-di-Giacomo git-test % git log
commit 0ab75724246bf8216fadbf754a848fe132372 (HEAD)
Author: Giacomo Cappon <cappongiamoco@gmail.com>
Date:   Tue Sep 28 17:21:55 2021 +0200

    new file text.txt
[cappe@MacBook-Pro-di-Giacomo git-test % git checkout 5a7bab
Previous HEAD position was 0ab7572 new file text.txt
HEAD is now at 5a7bab5 modified text.txt
[cappe@MacBook-Pro-di-Giacomo git-test %
```



Branching

- We have a tree -> We have branches



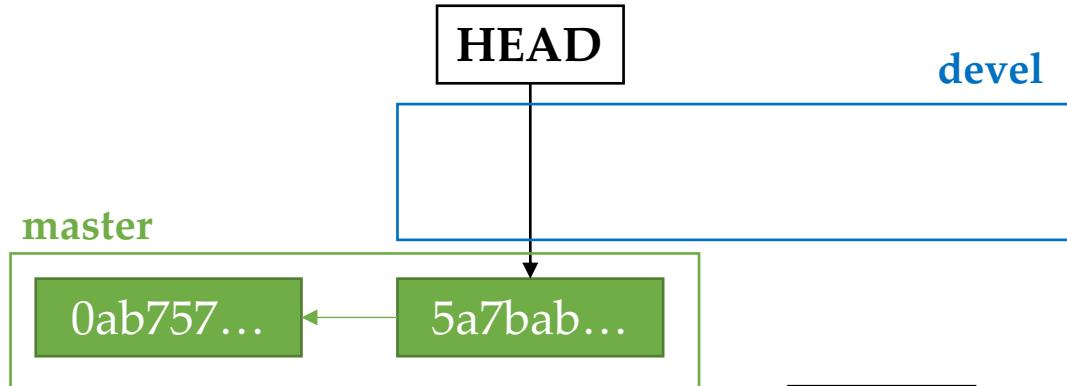
- Branching is one the greatest features of GIT and allows to create parallel branches to master and start working from there without modifying the other branches (master included).
- Why this is game changer? Because now you can
 - Work together on parallel branches
 - Test functionalities in dedicated branches without affecting the main code
 - Fix bugs efficiently

Create new branches

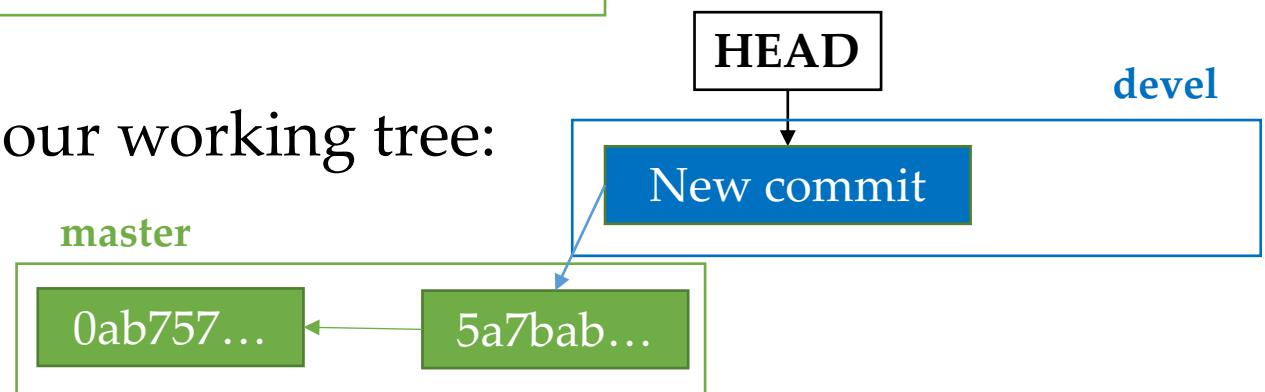
- To create a branch from the current commit you can use the “git branch” command

```
>> git branch devel
```

- Now this is our working tree:
 - We have a new branch called devel
 - devel is currently empty
 - HEAD points to the latest commit



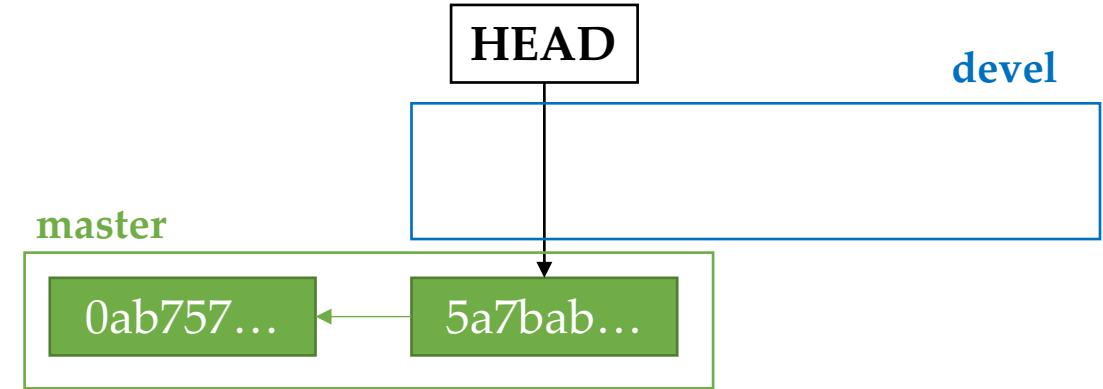
- If a new commit occurs this will be our working tree:



Navigate between branches

- To navigate between branches, it is sufficient to use “git checkout <branch>”:

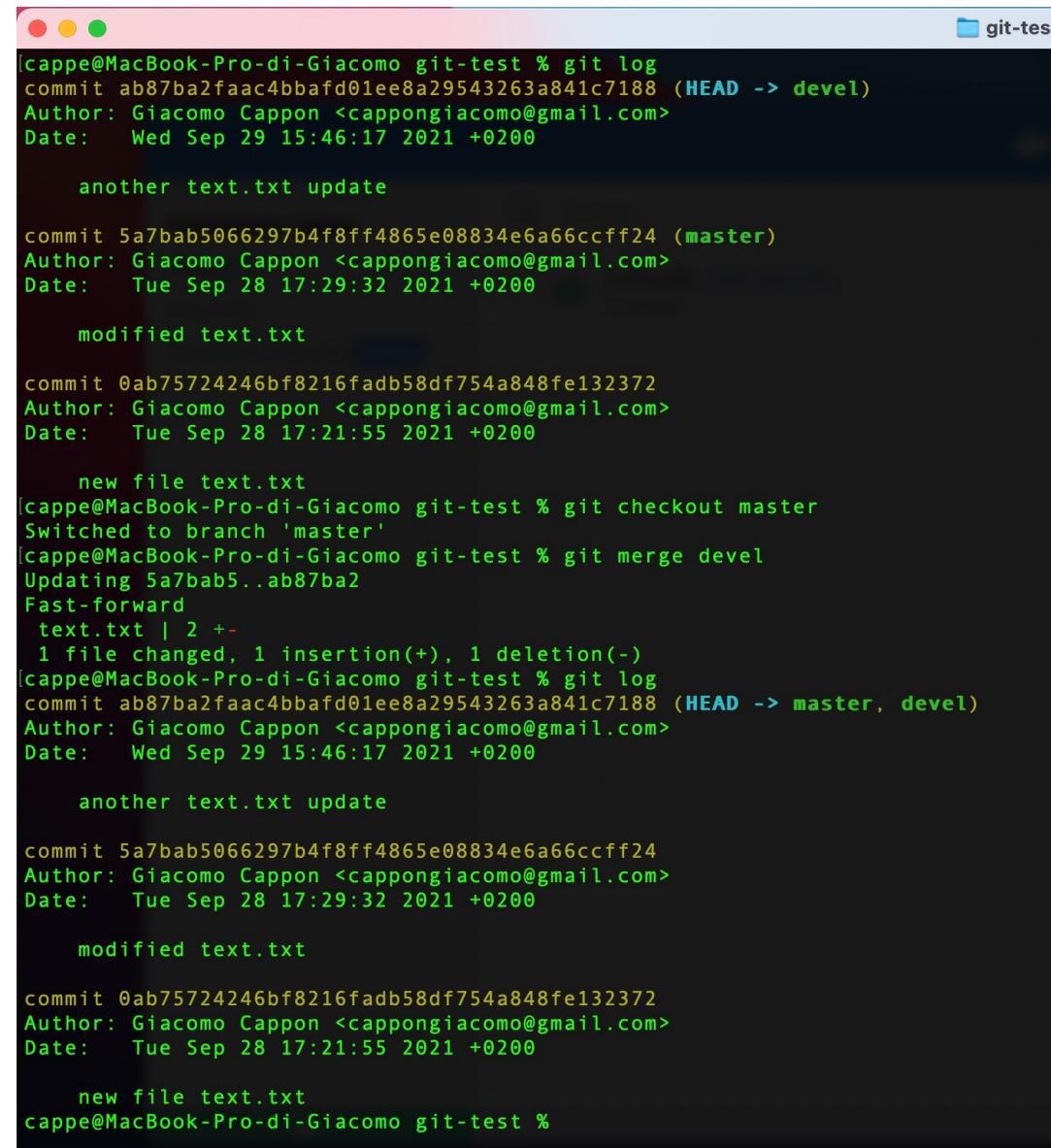
```
>> git checkout master  
>> git checkout devel
```



- **IMPORTANT:** if you have made some modifications, before navigating to another branch, you have to stash or commit those changes first.

Merging branches

- How to apply the changes I made in branch (e.g., `devel`) to another (e.g., `master`)?
- You need to merge the first branch (`devel`) into the other (`master`) using the command “`git merge <branch>`” which allows to apply the `<branch>` history in the current one.



The screenshot shows a terminal window titled "git-test" with the following content:

```
[cappe@MacBook-Pro-di-Giacomo git-test % git log
commit ab87ba2faac4bbaf01ee8a29543263a841c7188 (HEAD -> devel)
Author: Giacomo Cappon <cappongiacomo@gmail.com>
Date:   Wed Sep 29 15:46:17 2021 +0200

    another text.txt update

commit 5a7bab5066297b4f8ff4865e08834e6a66ccff24 (master)
Author: Giacomo Cappon <cappongiacomo@gmail.com>
Date:   Tue Sep 28 17:29:32 2021 +0200

    modified text.txt

commit 0ab75724246bf8216fadbf58df754a848fe132372
Author: Giacomo Cappon <cappongiacomo@gmail.com>
Date:   Tue Sep 28 17:21:55 2021 +0200

    new file text.txt
[cappe@MacBook-Pro-di-Giacomo git-test % git checkout master
Switched to branch 'master'
[cappe@MacBook-Pro-di-Giacomo git-test % git merge devel
Updating 5a7bab5..ab87ba2
Fast-forward
  text.txt | 2 ++
  1 file changed, 1 insertion(+), 1 deletion(-)
[cappe@MacBook-Pro-di-Giacomo git-test % git log
commit ab87ba2faac4bbaf01ee8a29543263a841c7188 (HEAD -> master, devel)
Author: Giacomo Cappon <cappongiacomo@gmail.com>
Date:   Wed Sep 29 15:46:17 2021 +0200

    another text.txt update

commit 5a7bab5066297b4f8ff4865e08834e6a66ccff24
Author: Giacomo Cappon <cappongiacomo@gmail.com>
Date:   Tue Sep 28 17:29:32 2021 +0200

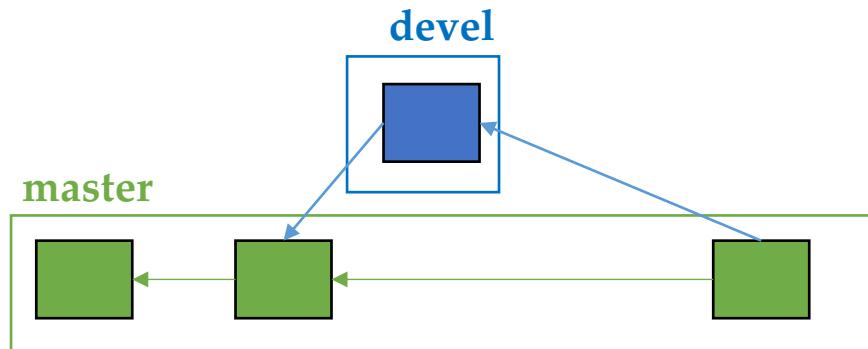
    modified text.txt

commit 0ab75724246bf8216fadbf58df754a848fe132372
Author: Giacomo Cappon <cappongiacomo@gmail.com>
Date:   Tue Sep 28 17:21:55 2021 +0200

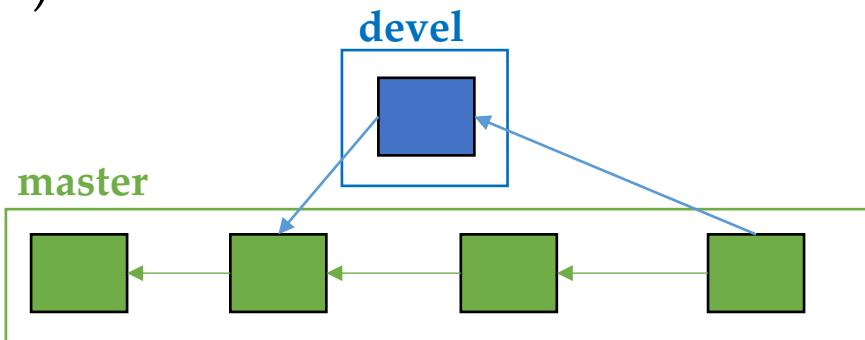
    new file text.txt
cappe@MacBook-Pro-di-Giacomo git-test %
```

Merging branches

- We have two scenarios:
 - Scenario 1 (like in the example):



- Scenario 2 (someone altered the parent branch):



```
[cappe@MacBook-Pro-di-Giacomo git-test % git log
commit ab87ba2faac4bbaf01ee8a29543263a841c7188 (HEAD -> devel)
Author: Giacomo Cappon <cappongiacomo@gmail.com>
Date:   Wed Sep 29 15:46:17 2021 +0200

    another text.txt update

commit 5a7bab5066297b4f8ff4865e08834e6a66ccff24 (master)
Author: Giacomo Cappon <cappongiacomo@gmail.com>
Date:   Tue Sep 28 17:29:32 2021 +0200

    modified text.txt

commit 0ab75724246bf8216fadbf58df754a848fe132372
Author: Giacomo Cappon <cappongiacomo@gmail.com>
Date:   Tue Sep 28 17:21:55 2021 +0200

    new file text.txt
[cappe@MacBook-Pro-di-Giacomo git-test % git checkout master
Switched to branch 'master'
[cappe@MacBook-Pro-di-Giacomo git-test % git merge devel
Updating 5a7bab5..ab87ba2
Fast-forward
  text.txt | 2 ++
  1 file changed, 1 insertion(+), 1 deletion(-)
[cappe@MacBook-Pro-di-Giacomo git-test % git log
commit ab87ba2faac4bbaf01ee8a29543263a841c7188 (HEAD -> master, devel)
Author: Giacomo Cappon <cappongiacomo@gmail.com>
Date:   Wed Sep 29 15:46:17 2021 +0200

    another text.txt update

commit 5a7bab5066297b4f8ff4865e08834e6a66ccff24
Author: Giacomo Cappon <cappongiacomo@gmail.com>
Date:   Tue Sep 28 17:29:32 2021 +0200

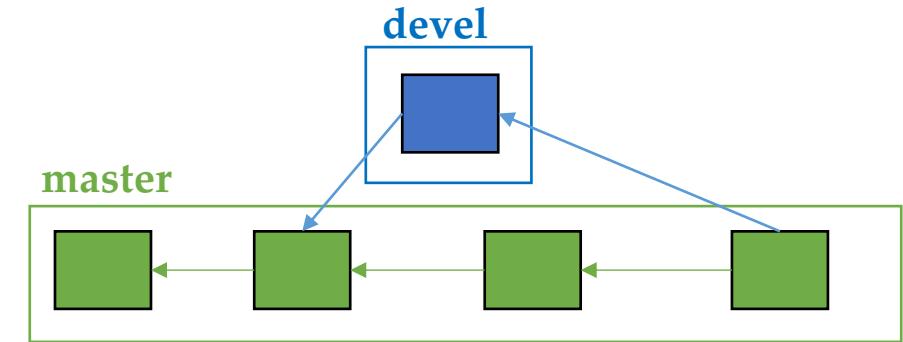
    modified text.txt

commit 0ab75724246bf8216fadbf58df754a848fe132372
Author: Giacomo Cappon <cappongiacomo@gmail.com>
Date:   Tue Sep 28 17:21:55 2021 +0200

    new file text.txt
cappe@MacBook-Pro-di-Giacomo git-test %
```

Managing scenario 2

- Scenario 2 is tricky (and the most common)
- It can happen that someone changed the same lines of code!
- In this case you need to solve the conflicts manually. Let's see how in a representative example



Managing scenario 2

- GIT asks us to fix the conflicts and then commit once solved. So let's do it.
- VSCode will look like this
- The software is helping us in managing the conflicts. Simply chose between “Accept current change” if you want to leave text.txt as in master or “Accept incoming change” if you want to set that line as in devel

The terminal window shows the command `git merge devel` being run, resulting in a merge conflict in `text.txt`. The conflict message indicates that automatic merge failed and suggests fixing conflicts and committing the result.

```
[cappe@MacBook-Pro-di-Giacomo git-test % git merge devel
Auto-merging text.txt
CONFLICT (content): Merge conflict in text.txt
Automatic merge failed; fix conflicts and then commit the result.
cappe@MacBook-Pro-di-Giacomo git-test % ]
```

The VSCode editor shows the file `text.txt` with the following content:

```
text.txt
Users > cappe > Repos > git-test > text.txt
...
1 <<<<< HEAD (Current Change)
2 world!
3 =====
4 Hello,
5 >>>>> devel (Incoming Change)
6 >
```

The status bar at the bottom of the VSCode window indicates: Restricted Mode, 0 changes, 0 conflicts, Ln 6, Col 2, Spaces: 4, UTF-8, LF, Plain Text.

Managing scenario 2

- Now that the conflict is solved, let's commit

- **ERROR:** remember to stage first, then commit!

- And there we go.

```
git-test -- zsh -- 94x18
cappe@MacBook-Pro-di-Giacomo git-test % git checkout devel
Switched to branch 'devel'
cappe@MacBook-Pro-di-Giacomo git-test % git checkout master
Switched to branch 'master'
cappe@MacBook-Pro-di-Giacomo git-test % git merge devel
Auto-merging text.txt
CONFLICT (content): Merge conflict in text.txt
Automatic merge failed; fix conflicts and then commit the result.
cappe@MacBook-Pro-di-Giacomo git-test % git commit -m "solved conflict and merged"
U      text.txt
error: Committing is not possible because you have unmerged files.
hint: Fix them up in the work tree, and then use 'git add/rm <file>'
hint: as appropriate to mark resolution and make a commit.
fatal: Exiting because of an unresolved conflict.
cappe@MacBook-Pro-di-Giacomo git-test % git add *
[cappe@MacBook-Pro-di-Giacomo git-test % git commit -m "solved conflict and merged"
[master a42fe5d] solved conflict and merged
cappe@MacBook-Pro-di-Giacomo git-test %
```

Outline

- Part 1: Version Control Systems & GIT
- Part 2: Playing around with GIT
- Part 3: The working tree
- **Part 4: Remote repositories and best practices**
- Part 5: Our development environment

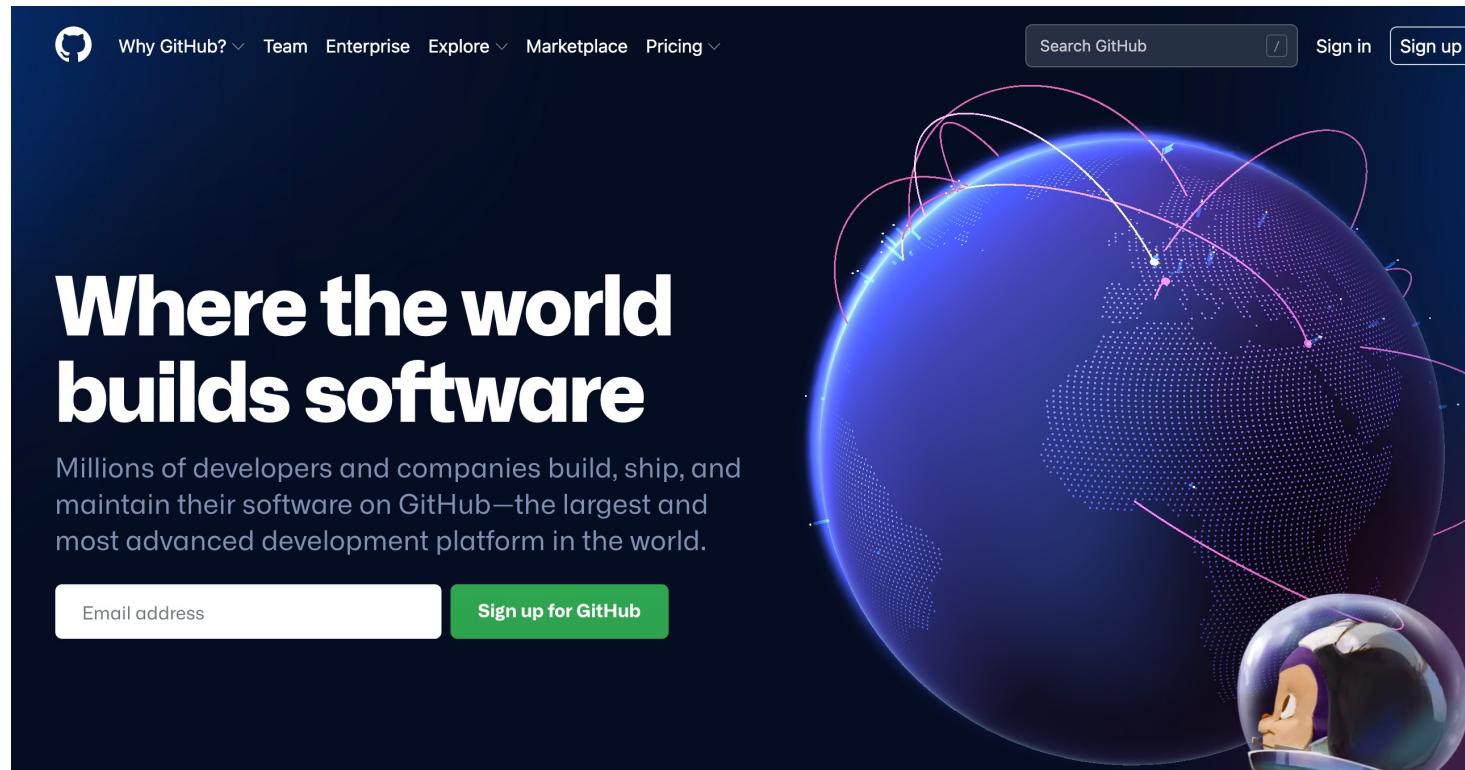
Remote repositories

- Up to this point, we worked on a local repository. This means that collaborating is still a problem.
- How to share code among collaborators? We have to go remote!
- Simple procedure:
 - Step 1: Setup a remote (online) repository hosted by a GIT provider (e.g., GitHub, GitLab, Bitbucket,...)
 - Step 2: Connect your local repository to the remote repository



Step 1: Setup a remote repository

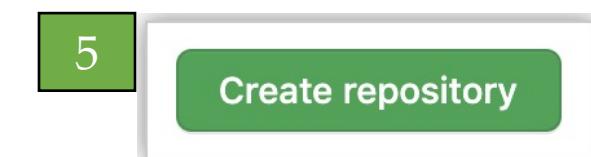
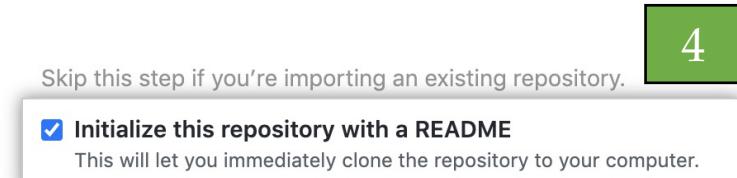
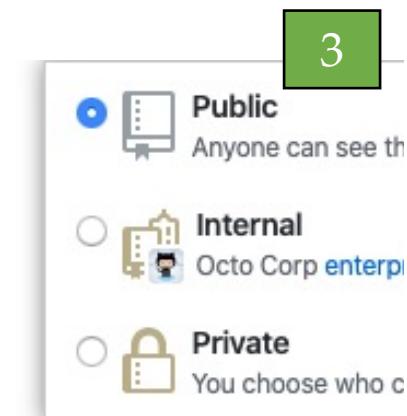
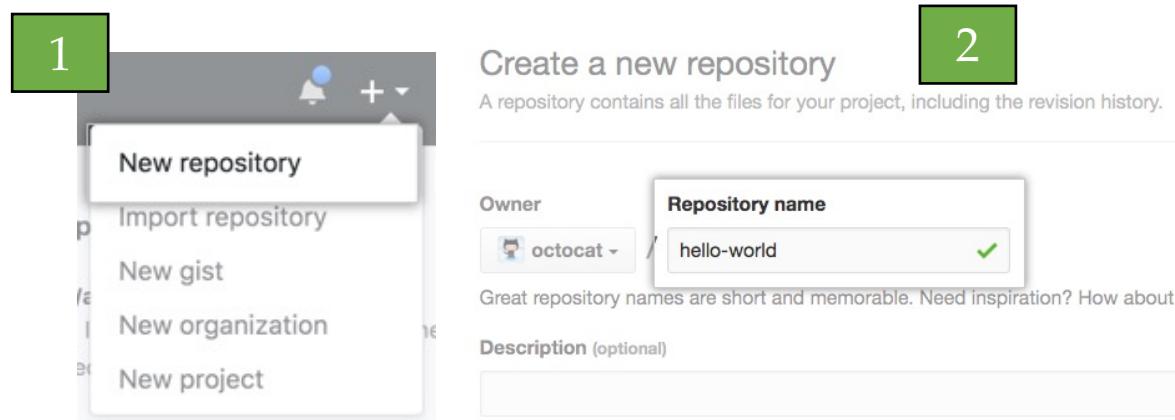
- To setup a remote repository, start by creating an account on one of the many GIT providers available
 - In this course, I will refer to GitHub but you are free to choose whatever you want.



Step 1: Setup a remote repository

➤ Once created, sign in and follow these steps

1. In the upper-right corner, use the + drop-down menu, and select “New repository”
2. Type a name for your repository (the same as your local one)
3. Choose a visibility
4. Uncheck the “README” box
5. Click “Create repository”



Step 2: Connect the local repository

- Then, to link your local repository to the online follow these steps:
 1. Open a terminal and move inside your working directory
 2. Type

```
>> git branch -M master
>> git remote add origin
https://github.com/<username>/<remote-repo-name>.git
```

Interacting with the remote repository

- Now you are ready to work
- Interacting with the remote repository is quite easy
- To synchronize your local changes to the remote repository (to send online the code) use the “git push” command
 - >> `git push origin <branch-name>`

This will “upload” the local changes of `<branch-name>` to a remote branch with the same name

- To synchronize the remote changes to the local repository (to download the code) use the “git pull” command
 - >> `git pull`

This will “download” the remote changes of your current active branch

Many more things...

- Of course GIT has many more functionalities
- ...and you will need some of them
- Anyway, this is just a primer and learning GIT is a matter of practice
- If you have any questions, doubt, etc..., do not hesitate to ask. I suggest you to make some practice to get familiar with the basics by creating a dummy repo and start working on it.
- You can find useful resources in the following slide

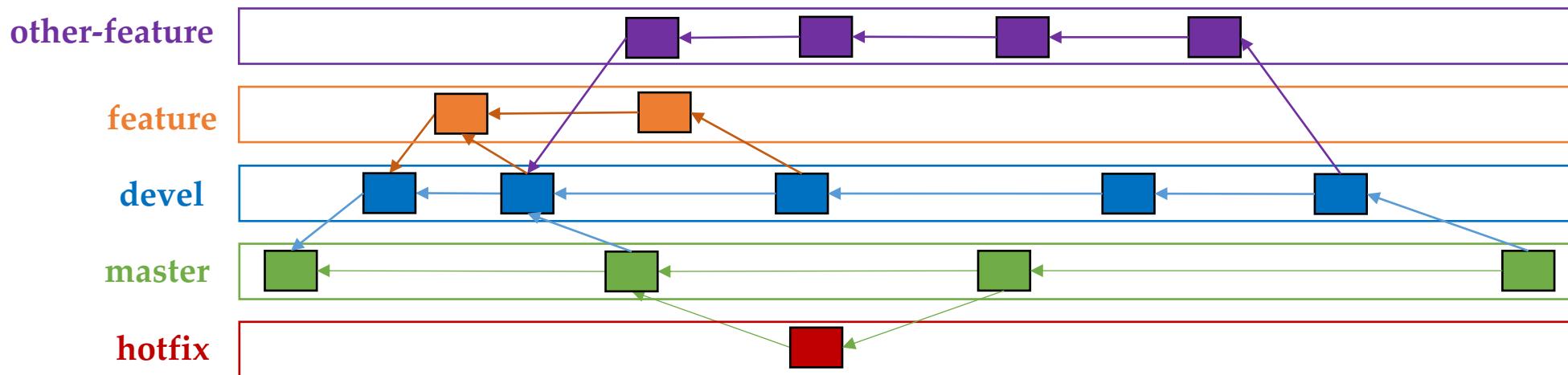
Useful resources

- Git cheatsheet
 - Moodle of the course (gently provided by GitHub)
- Git official book
 - <https://git-scm.com/book/en/v2>
- Git Tutorial for Beginners: Learn Git in 1 hour
 - https://www.youtube.com/watch?v=8JJ101D3knE&ab_channel=ProgrammingwithMosh
- Oh Shit Git
 - <https://ohshitgit.com/>

Best practices

- Start the commit message with the branch name, for example
 - `git commit -m "[master] a message"`

- Setup the working tree somehow like that:



- `git pull` before starting working on main branches (e.g., `devel`, `master`)
- at least at the beginning, designate one member of the team that will merge things
- work on an independent branch (not shared with someone else)

Outline

- Part 1: Version Control Systems & GIT
- Part 2: Playing around with GIT
- Part 3: The working tree
- Part 4: Remote repositories and best practices
- **Part 5: Our development environment**

Our environment



IDE

(To write code,
compile, and test)



Flutter + Dart

(The programming
language)



Android Studio

(For Android
support)



XCode

(For iOS
support)



GIT

(For version
control)

(How to setup) Our environment

- To install the compilers, language, etc... (Flutter + Dart)
 - Go to > <https://flutter.dev/docs/get-started/install>
 - Follow the instructions...
- To install iOS support (Xcode)
 - Open the App Store > Search “Xcode” > Install Xcode (it takes a lot of time)
- To install Android support (Android Studio)
 - Go to > <https://developer.android.com/studio/install>
 - Follow the instructions...
 - Open Android Studio
 - Go to > Preferences > Android SDK
 - Select the Android version of your phone > Click “Apply”