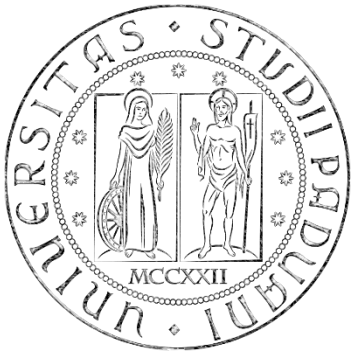
UNIVERSITY OF PADOVA
DEPARTMENT OF INFORMATION ENGINEERING

Biomedical Wearable Technologies
for Healthcare and Wellbeing

Git

A.Y. 2021-2022

Giacomo Cappon



DEPARTMENT OF
INFORMATION
ENGINEERING
UNIVERSITY OF PADOVA

Outline

- **Version Control Systems & GIT**
- Playing with GIT
- The working tree
- Remote repositories and best practices

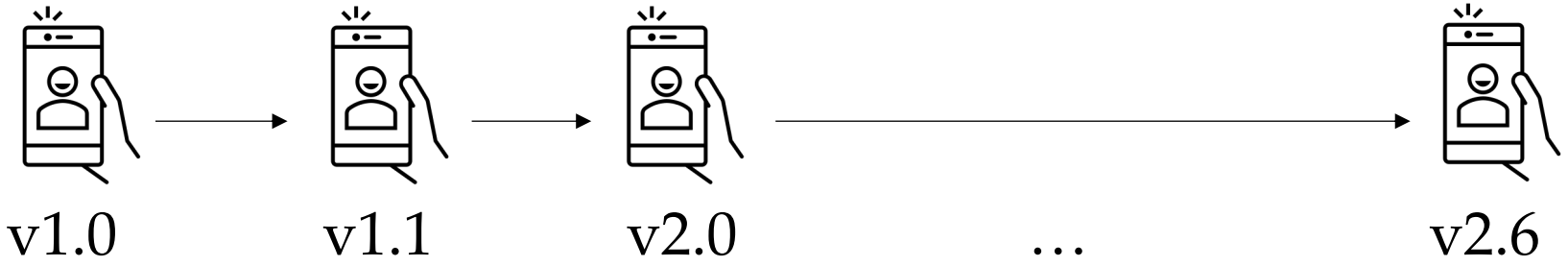
- Homework
- Resources

- Final project overview

New features means new code

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

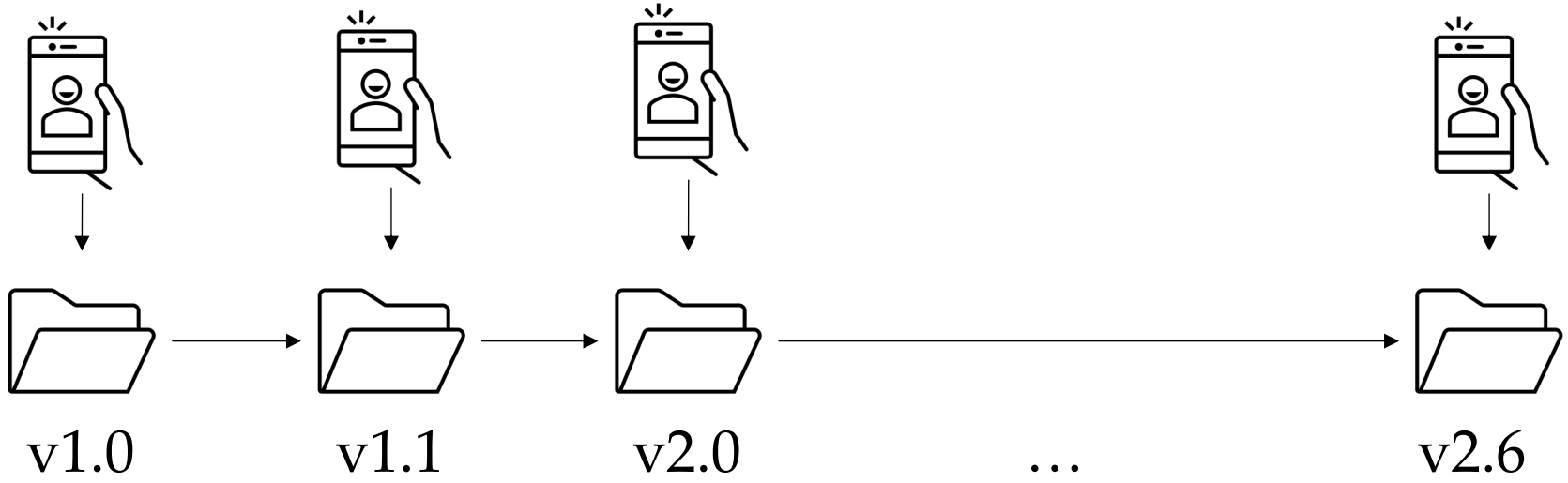
- This translates in multiple software (AKA code) versions



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?

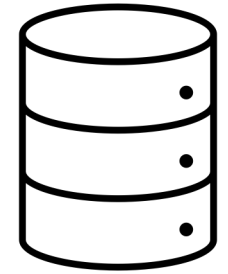
Problems of the “multiple folders” approach

- 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 is 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
  - …



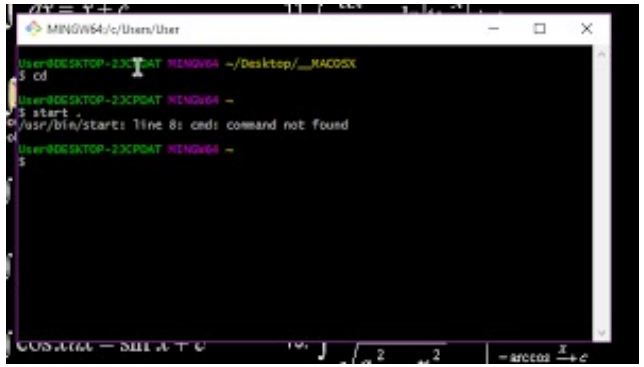
Repository

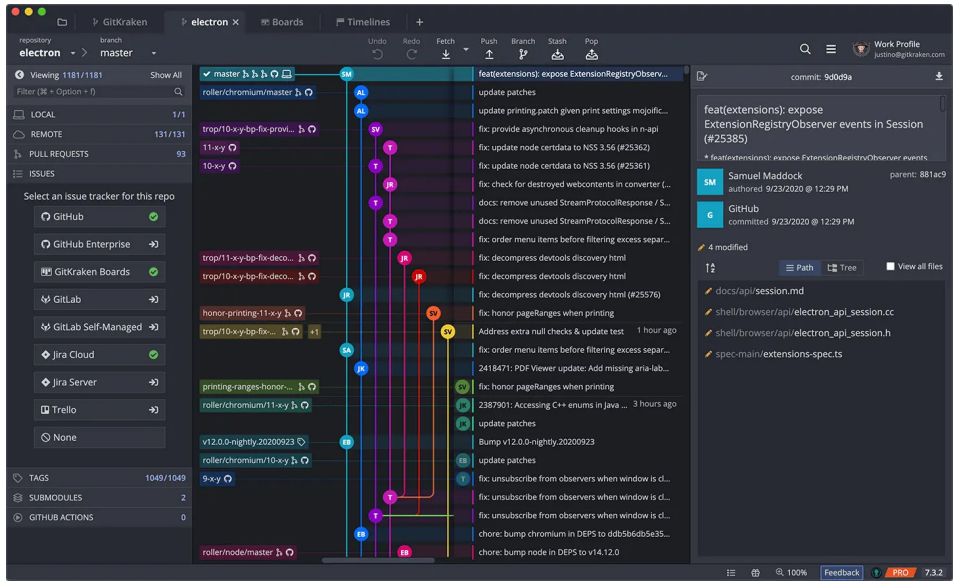
- Everything (code changes, contributors, deltas,…) is stored in a dedicated repository

- A repository can be stored locally or remotely (in remote repository databases) and connected to the code contributor machine

- 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

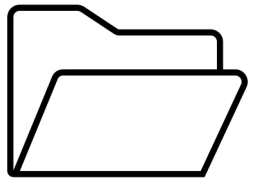
- GIT can be used via
  - Fancy Graphical User Interface (GUI)
  - Old-school terminal command line



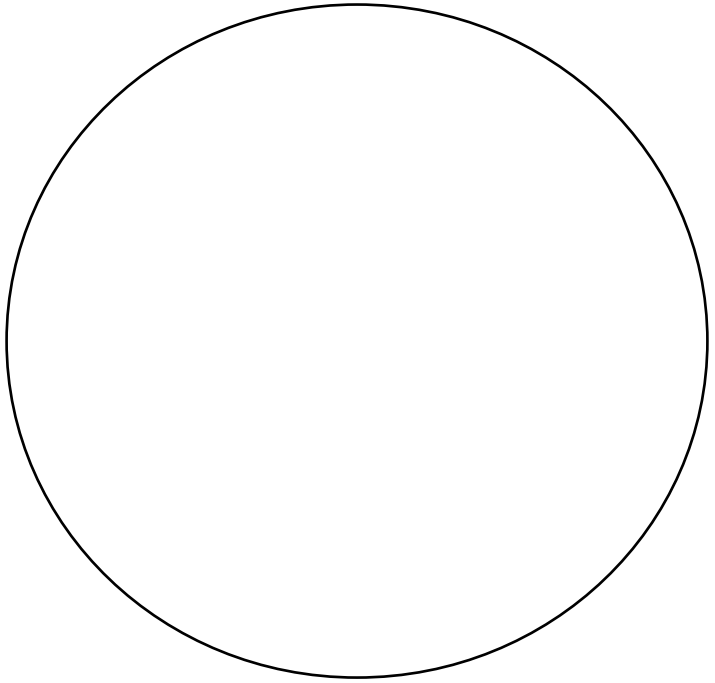


- In this course we will use the command line. Why?
  - GUI have limitations
  - GUI tools are not always available (e.g., remote servers)

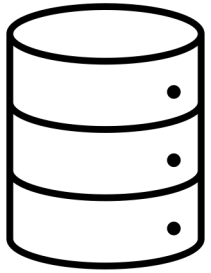
GIT workflow



Working directory



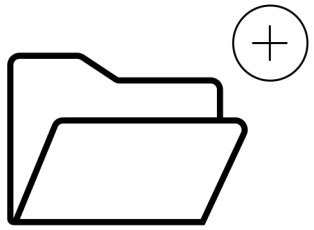
Staging area



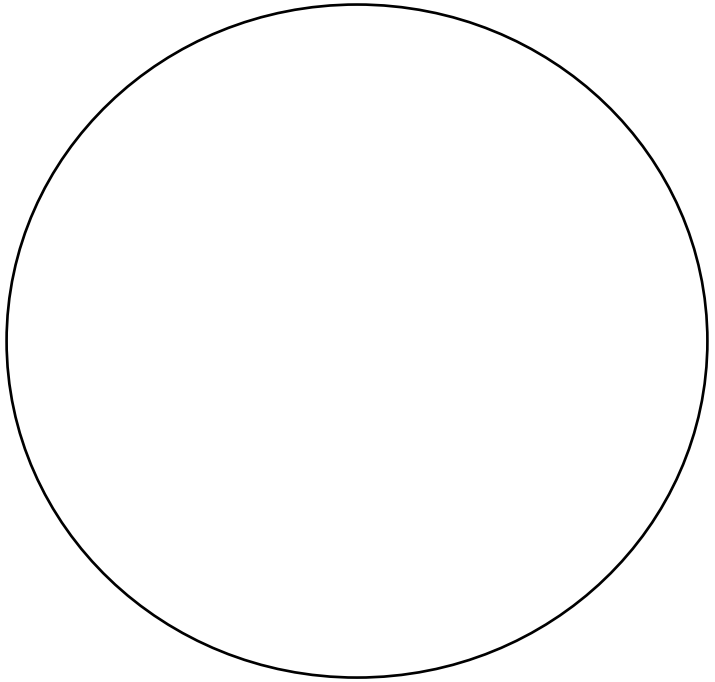
Repository

- **Working directory**: where the code is.
- **Staging area**: where you put the “new code” that you want to become the next code version.

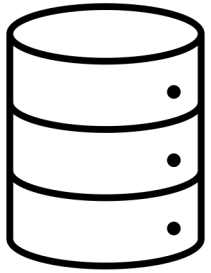
GIT workflow



Working directory
(with some modifications)



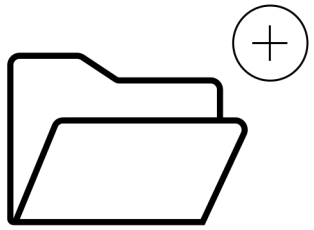
Staging area



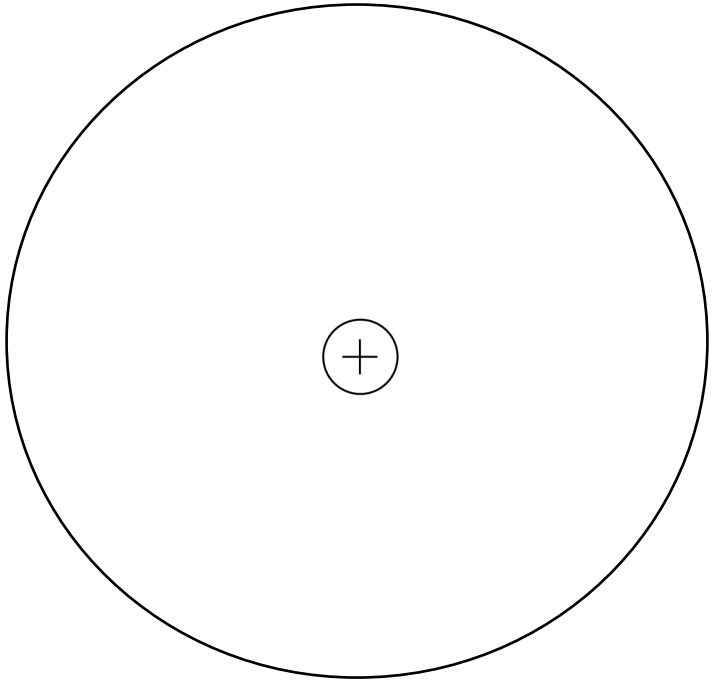
Repository

- 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, …)

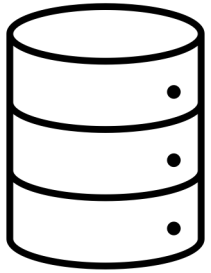
GIT workflow



Working directory
(with some modifications)



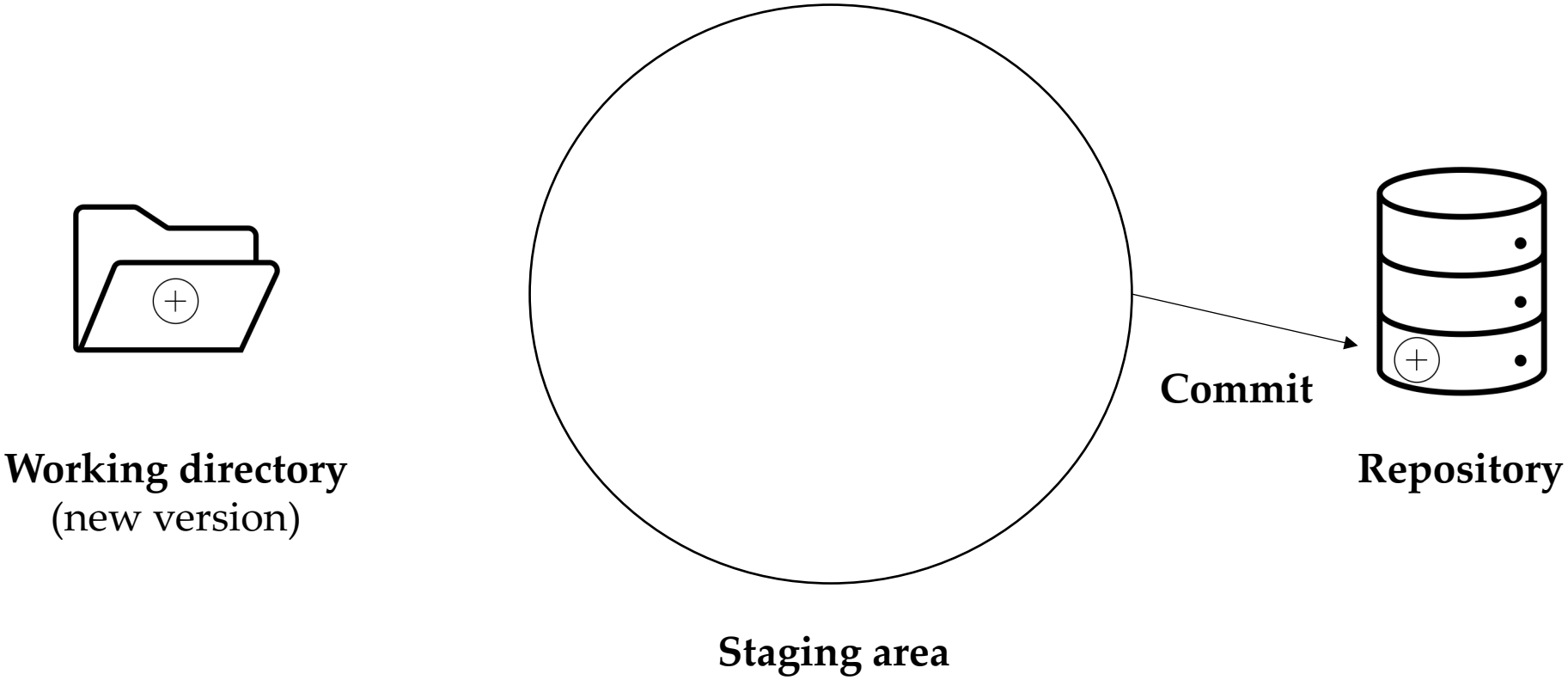
Staging area



Repository

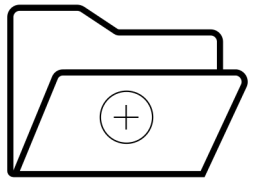
- **Step 1**: stage that code!

GIT workflow

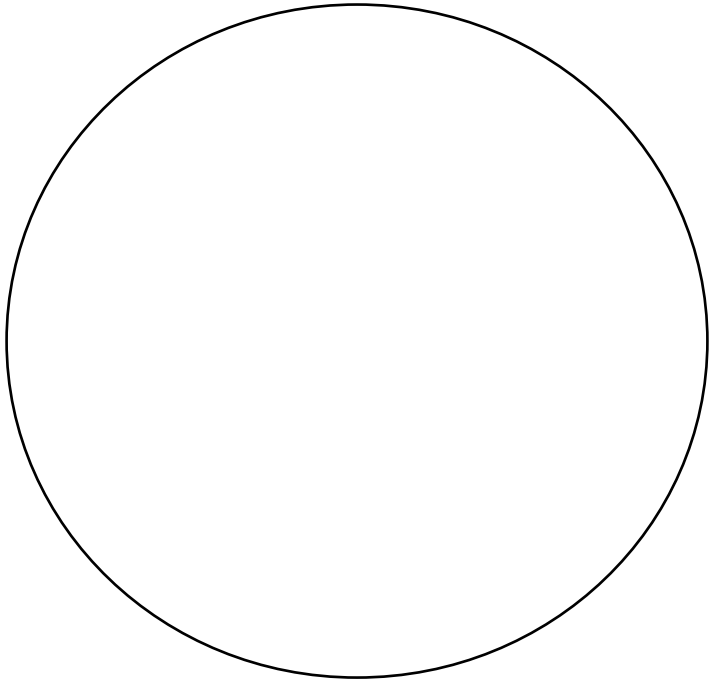


- **Step 2**: make a commit! Making a commit is like taking a snapshot of the code as it is now.

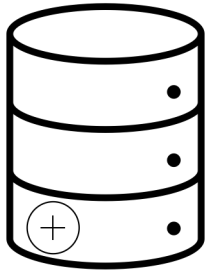
GIT workflow



Working directory
(new version)



Staging area



Repository

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

Outline

- Version Control Systems & GIT
- **Playing with GIT**
- The working tree
- Remote repositories and best practices

- Homework
- Resources

- Final project overview

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
```

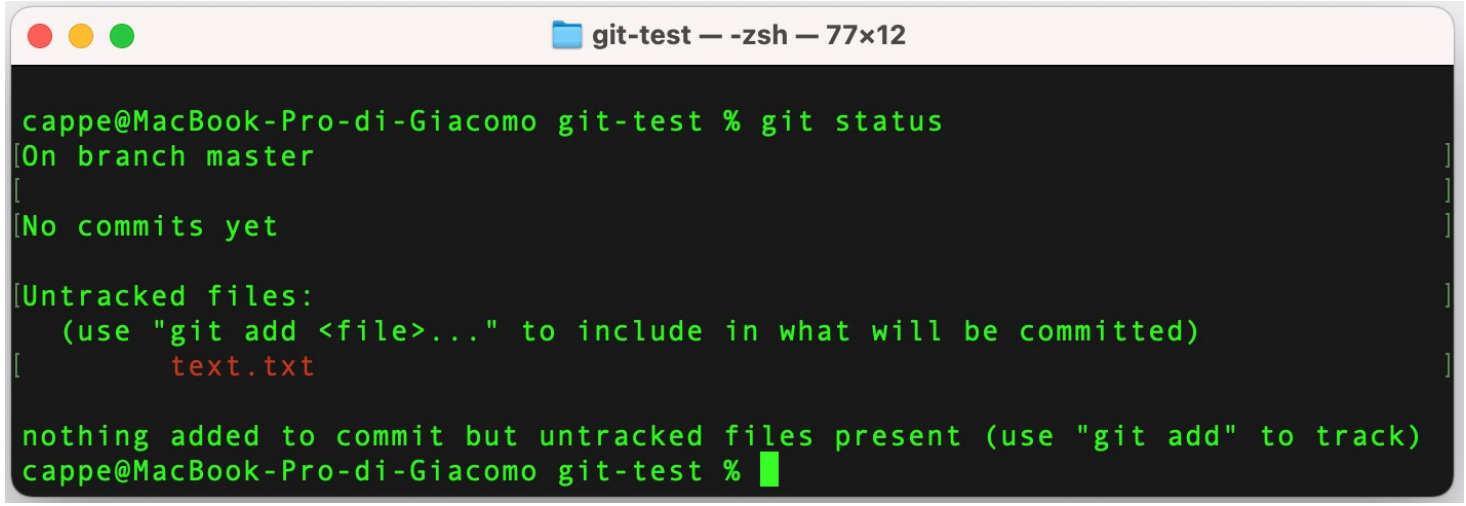
- Save `text.txt`. Now we are ready to play with GIT.

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



Step 3: Track the file history

- To track and stage "`text.txt`" use the "`git add`" command

```
>> git add text.txt
```

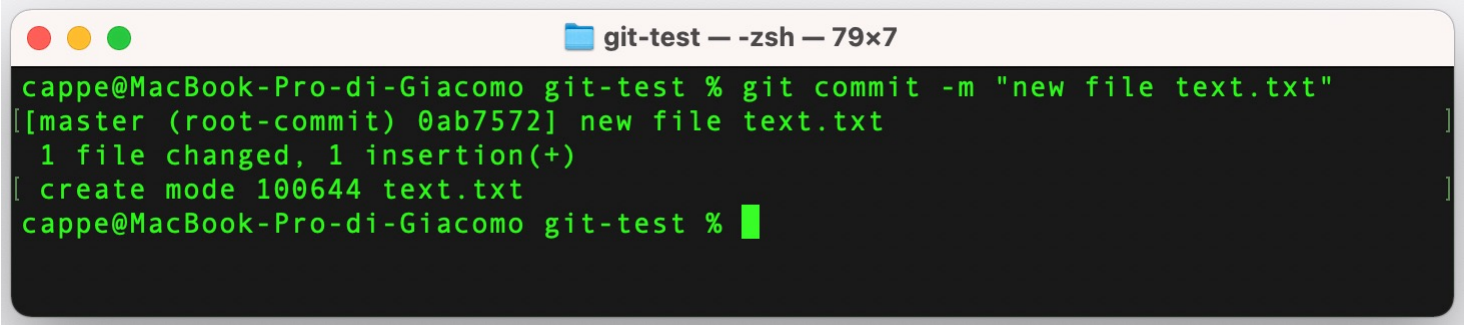
- Alternatively, you can use "`git add *`" to add all files in the project folders

- Check the status of the repository

Step 4: Commit the file

- To commit the file

```
>> git commit –m “new file text.txt”
```

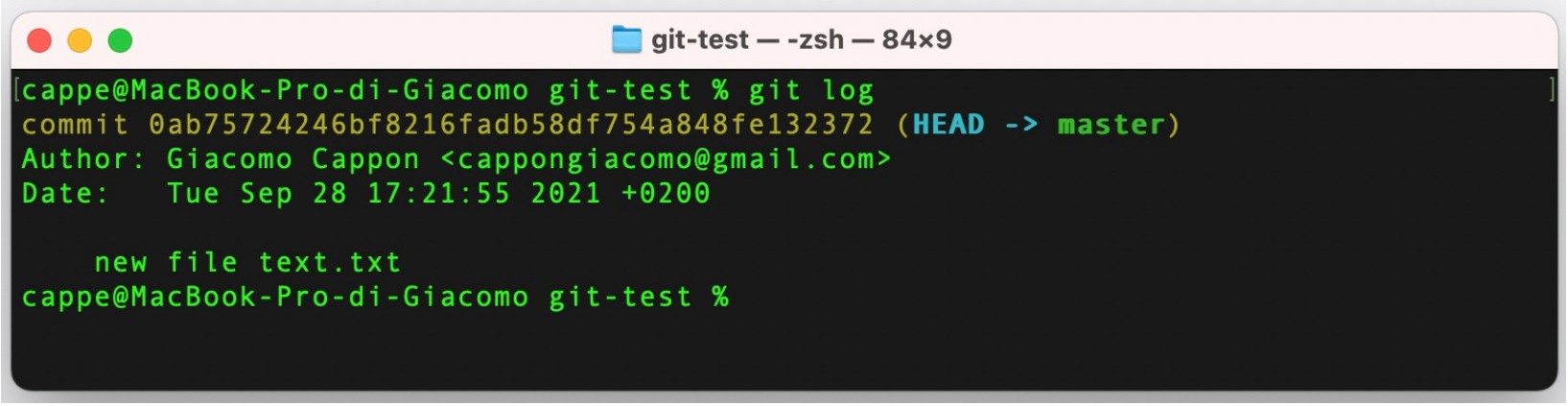


- Note that we added a message! This should be meaningful.

Step 5: Check the new history

- Check what we have done using the "`git status`" command

- Even more things with "`git log`"



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 (`git status` command)
  - Stage the modifications (`git add` command)
  - Commit the new modifications (`git commit` command)
- I'll give you 10 minutes to do that

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

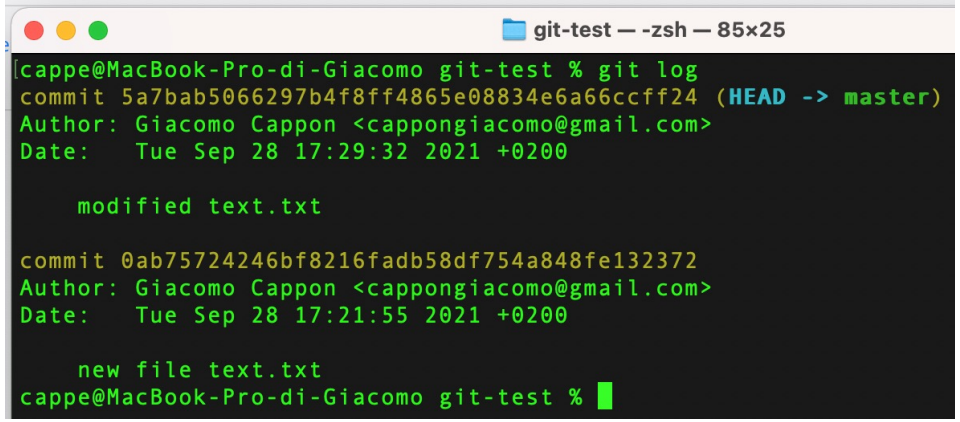
- Version Control Systems & GIT
- Playing with GIT
- **The working tree**
- Remote repositories and best practices

- Homework
- Resources

- Final project overview

GIT core: The working tree

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



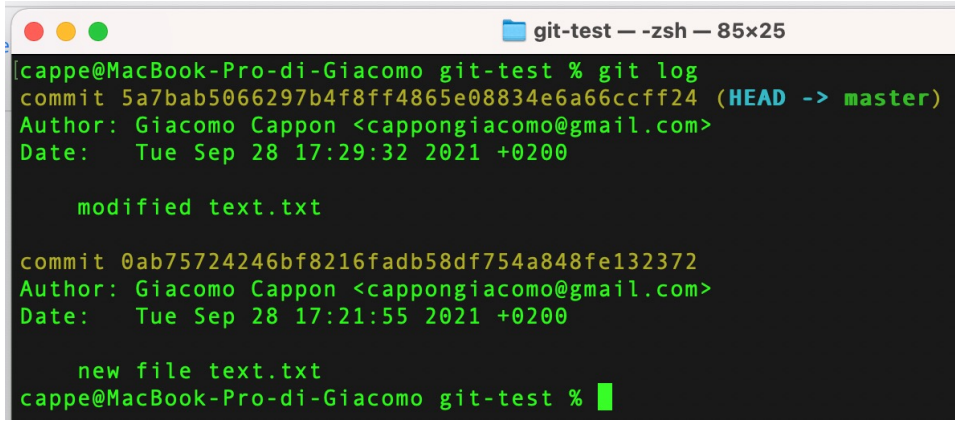
GIT core: The working tree

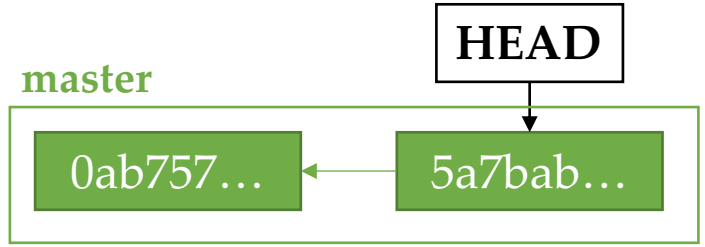
- The repository tracks the history thanks to a tree-based structure called **working tree** which memorizes the filesystem of all code versions.

- 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.

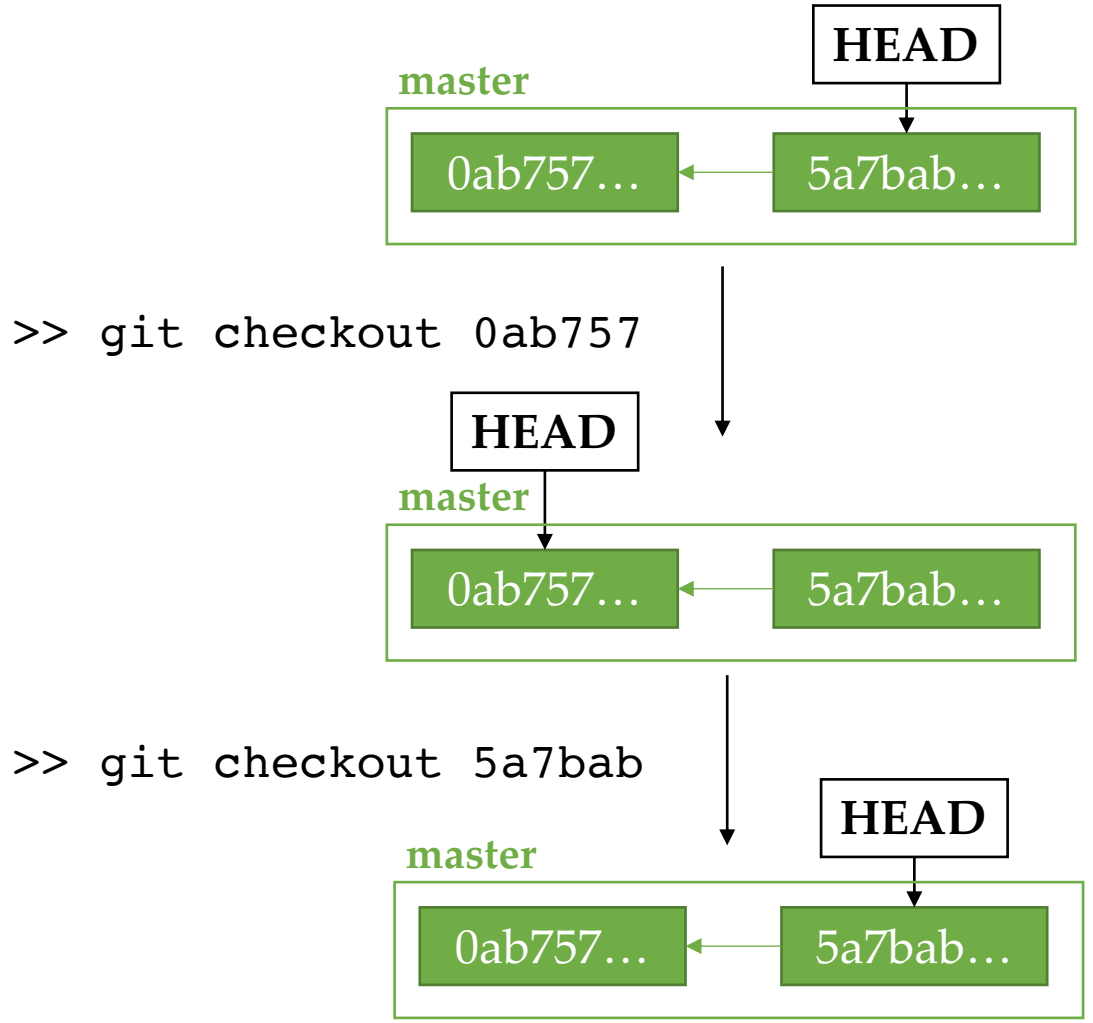




Moving through the working tree

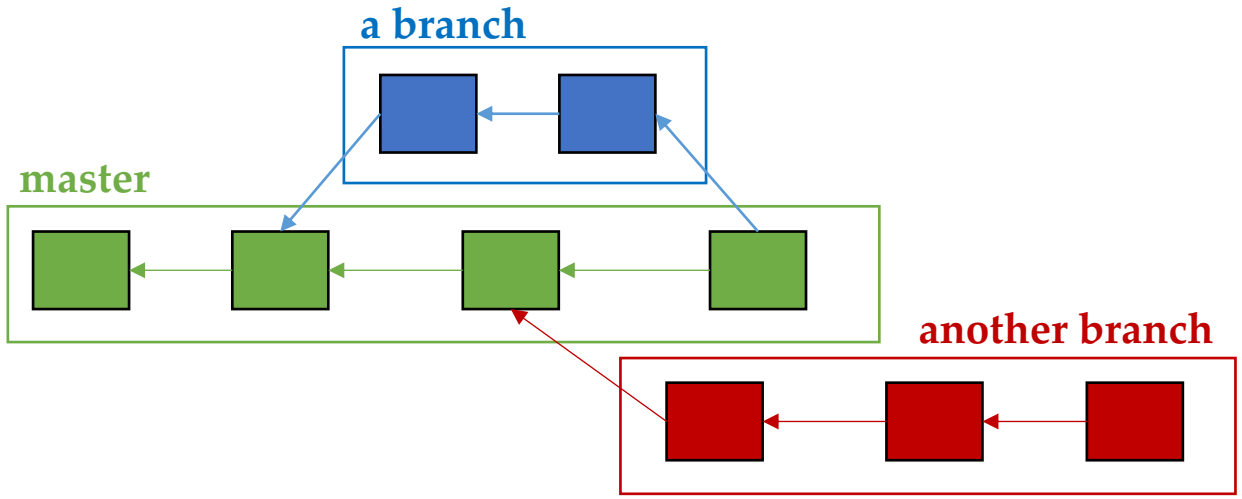
- Move through the working tree using the "`git checkout`" command

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

    modified text.txt

commit 0ab75724246bf8216fadb58df754a848fe132372
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 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 0ab75724246bf8216fadb58df754a848fe132372 (HEAD)
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 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).

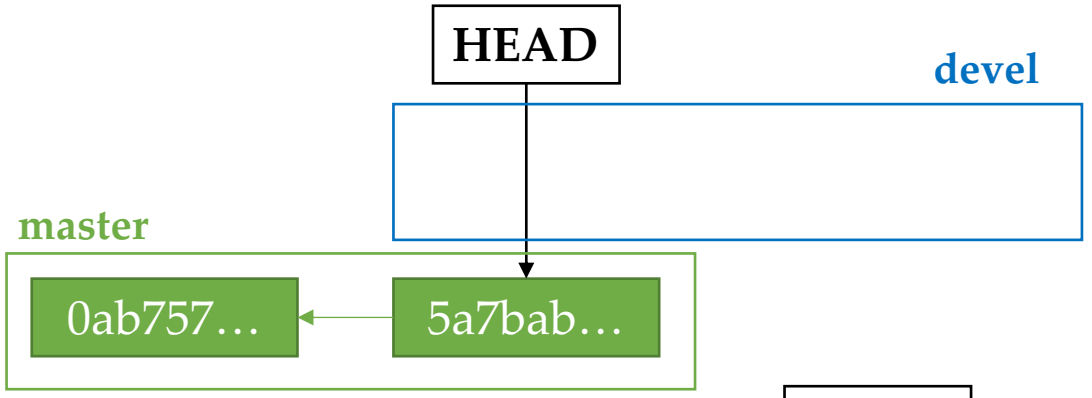
- Why this is game changer? Because now you can
  - Work together on parallel branches
  - Test functionalities in dedicated branches without affecting the main code
  - Fix bugs efficiently

Create new branches

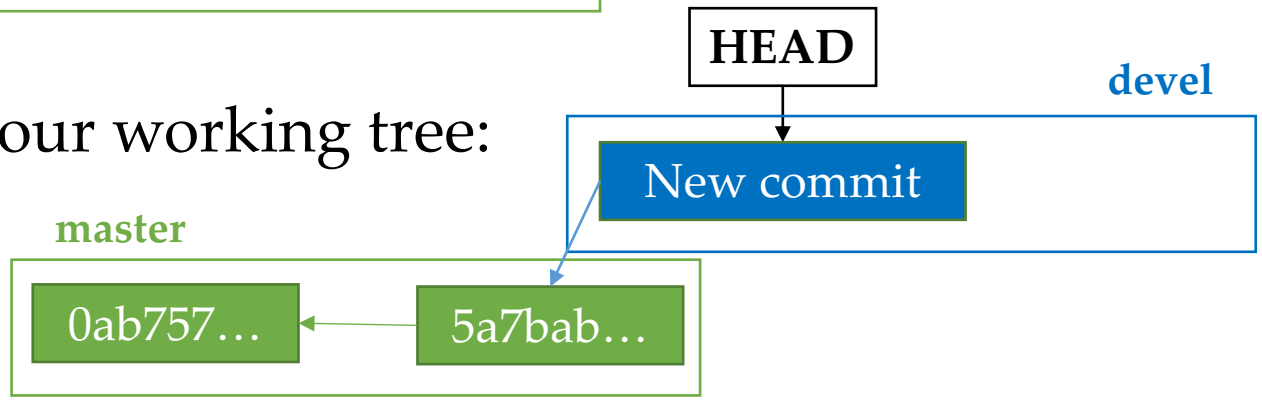
- To create a branch from the current commit you can use the "`git branch`" command

```
>> 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



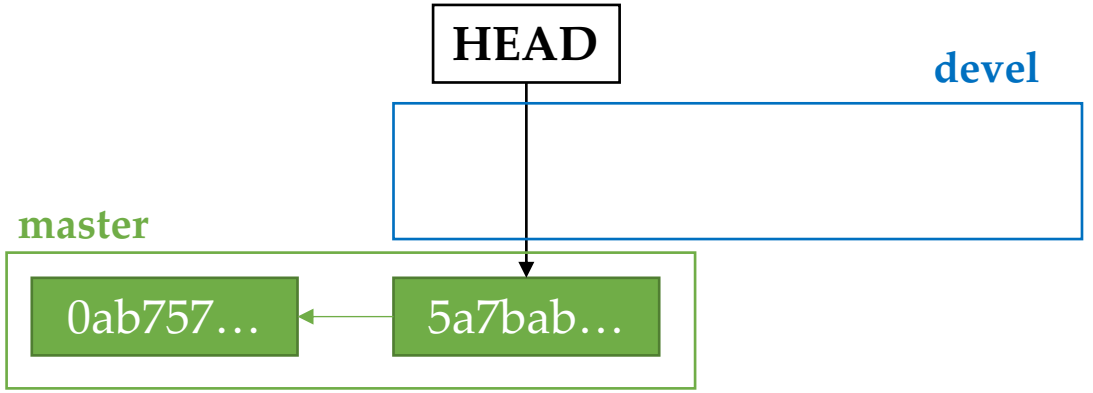
- If a new commit occurs this will be our working tree:



Navigate between branches

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

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

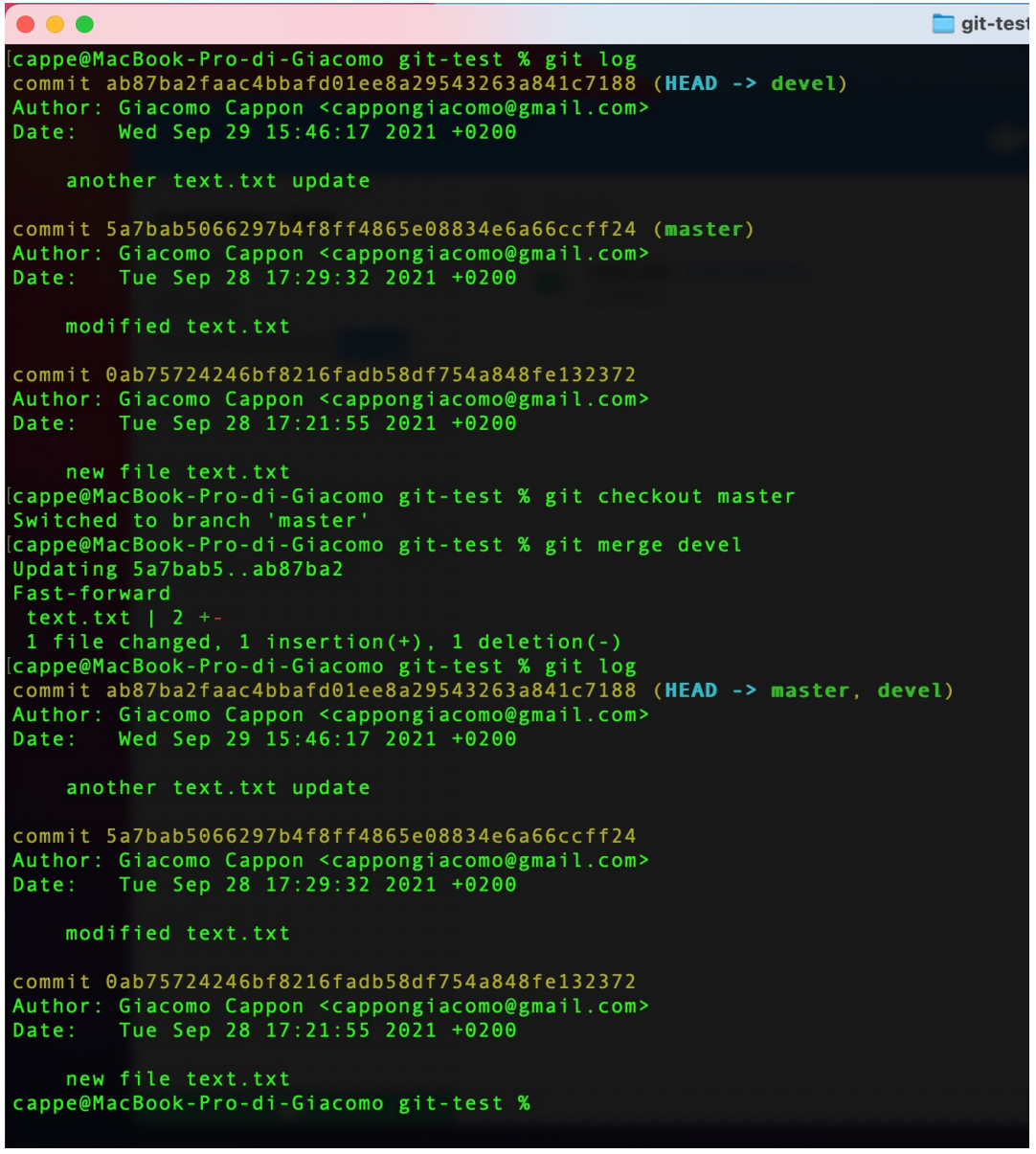


- **IMPORTANT:** if you have made some modifications, before navigating to another branch, you have to stash or commit those changes first.

Merging branches

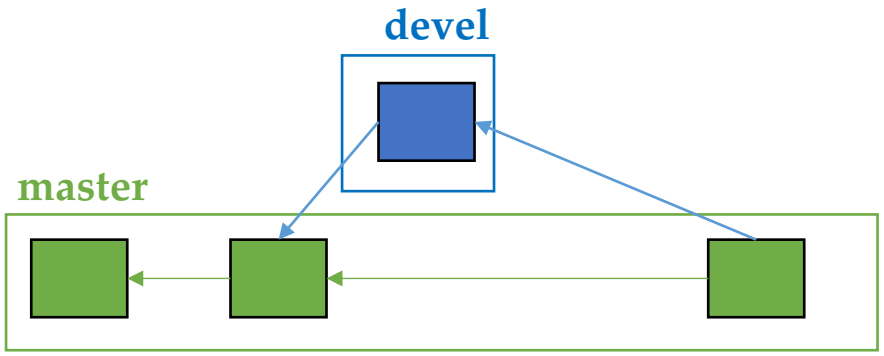
- How to apply the changes I made in branch (e.g., devel) to another (e.g., master)?

- 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.

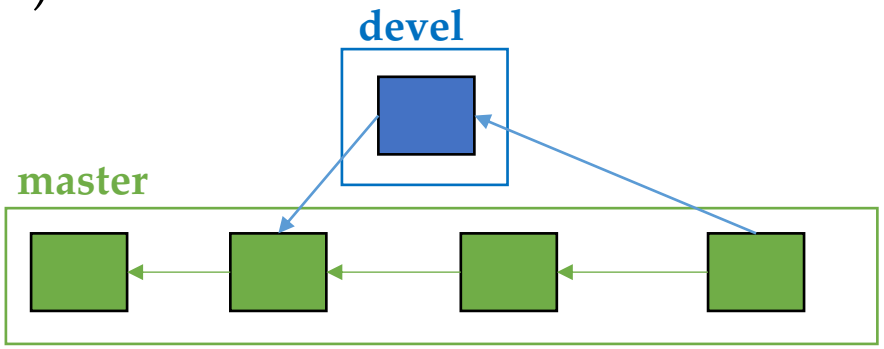


Merging branches

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



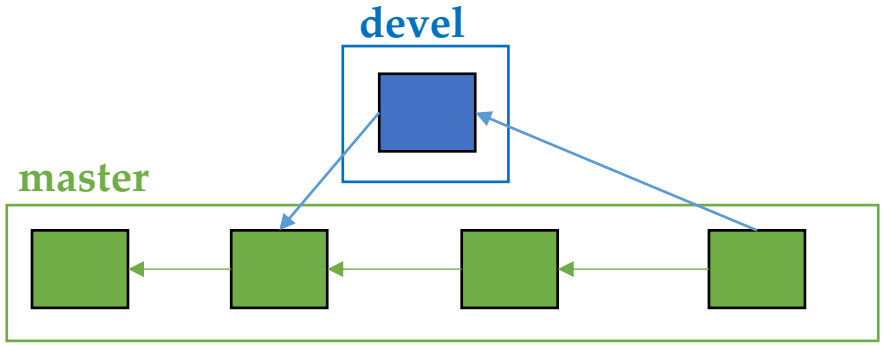
  - Scenario 2 (someone altered the parent branch):




```
[cappe@MacBook-Pro-di-Giacomo git-test % git log
commit ab87ba2faac4bbafd01ee8a29543263a841c7188 (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 0ab75724246bf8216fadb58df754a848fe132372
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 ab87ba2faac4bbafd01ee8a29543263a841c7188 (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 0ab75724246bf8216fadb58df754a848fe132372
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 %
```

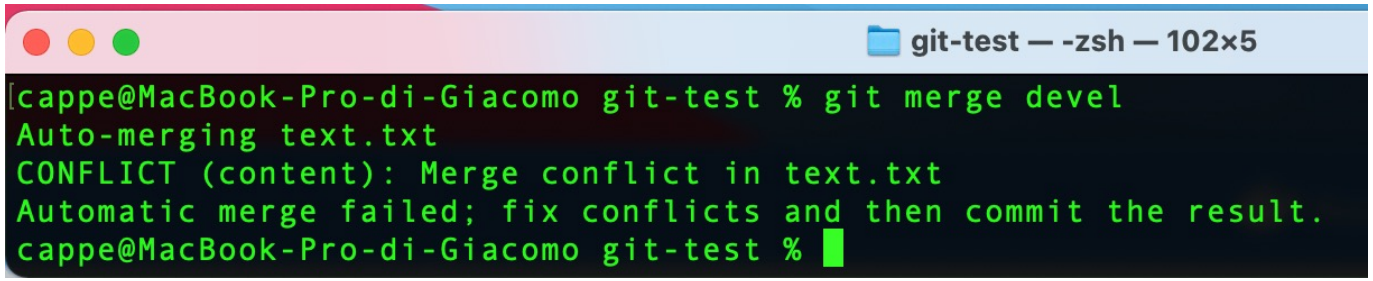
Managing scenario 2

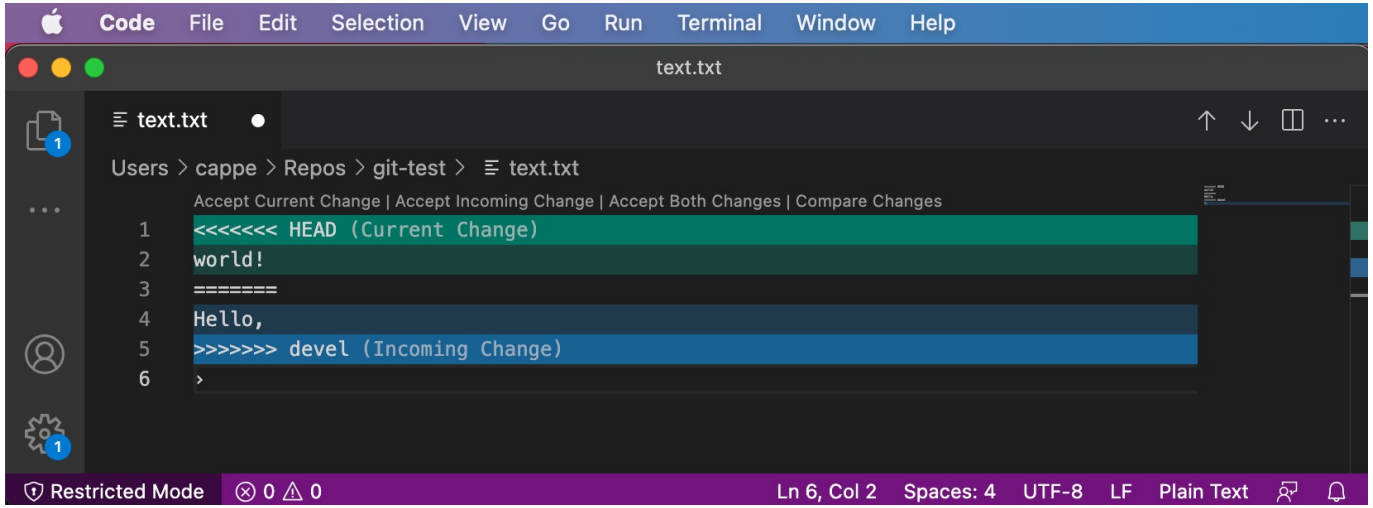
- Scenario 2 is tricky (and the most common)
- It can happen that someone changed the same lines of code!
- In this case you need to solve the conflicts manually. Let's see how in a representative example



Managing scenario 2

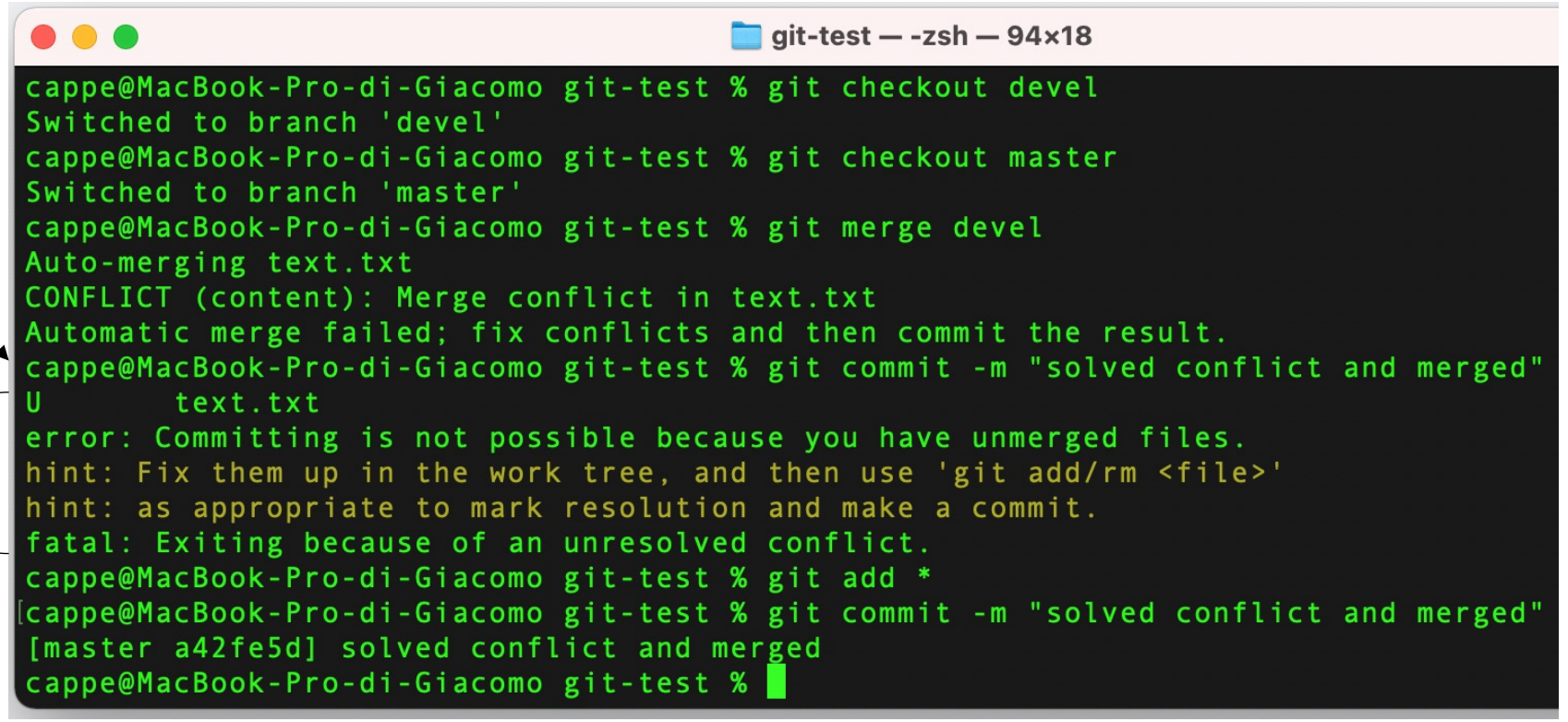
- GIT asks us to fix the conflicts and then commit once solved. So let's do it.
- Visual Studio Code (IDE of our course, see later slides) will look like this.
- The software is helping us in managing the conflicts. Simply chose between "Accept current change" if you want to leave text.txt as in master or "Accept incoming change" if you want to set that line as in devel





Managing scenario 2

- Now that the conflict is solved, let's commit
- **ERROR**: remember to stage first, then commit!
- And there we go.



Outline

- Version Control Systems & GIT
- Playing with GIT
- The working tree
- **Remote repositories and best practices**

- Homework
- Resources

- Final project overview

Remote repositories

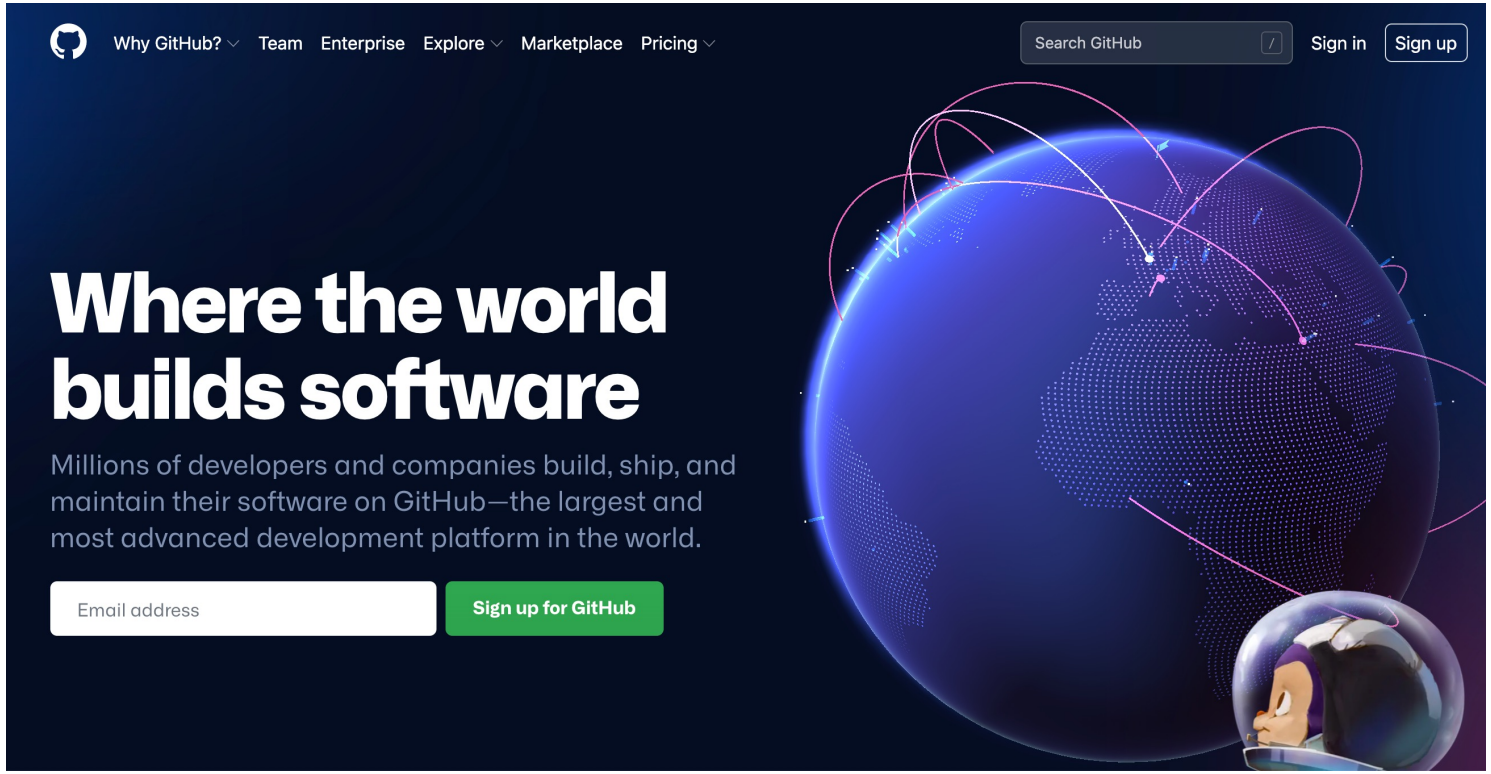
- Up to this point, we worked on a local repository. This means that collaborating is still a problem.

- 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.

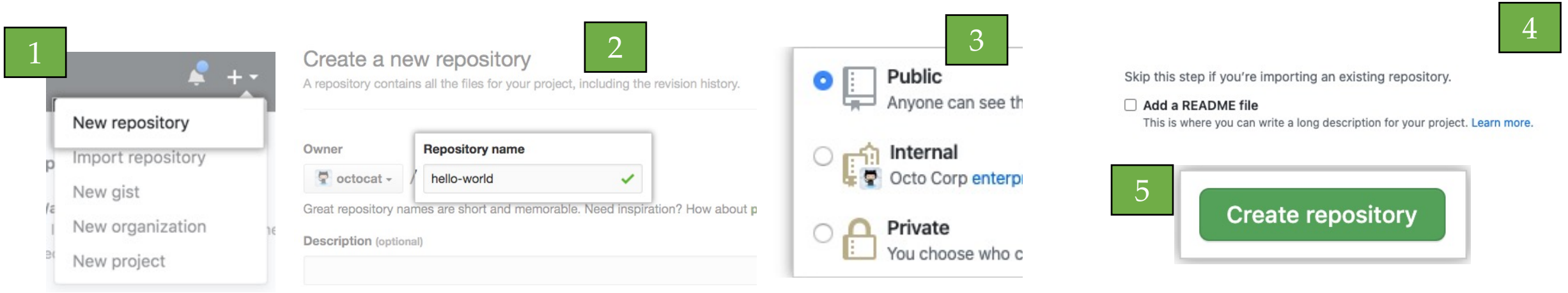


GitHub

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 in 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 sychronize the remote changes to the local repository (to download the code) use the "`git pull`" command

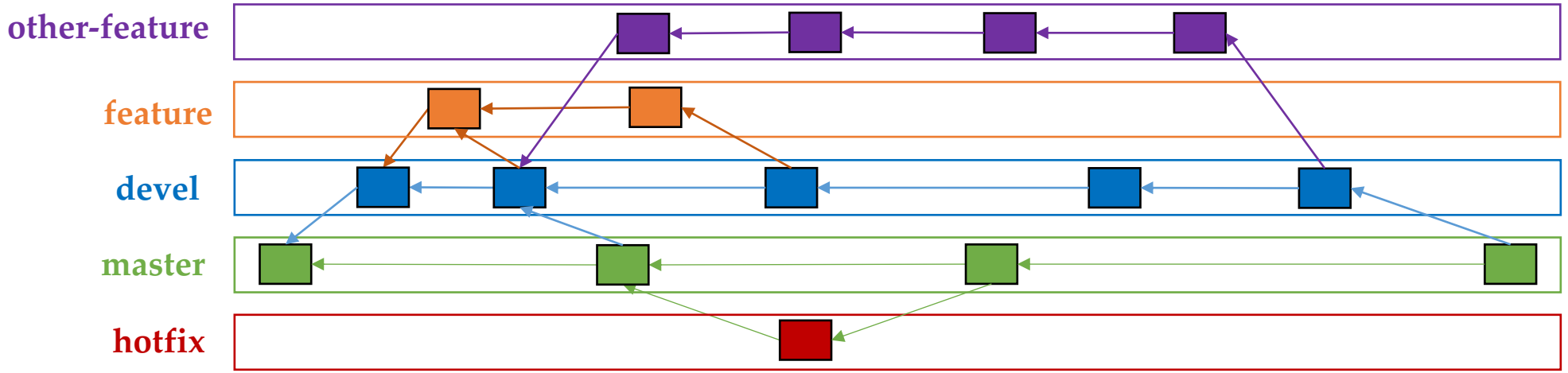
```
>> git pull
```

This will "download" the remote changes of your current active branch

Best practices

BONUS

- 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, design one member of the team that will merge things
- work on an independent branch (not shared with someone else)

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 follwing slide

Outline

- Version Control Systems & GIT
- Playing with GIT
- The working tree
- Remote repositories and best practices

- **Homework**
- Resources

- Final project overview

Homework

- Next time we will start learning about Dart, the programming language we will use to develop mobile apps in this course. To do that, we need our development environment, i.e., required software and libraries to actually develop something, to be ready.

- At DEI, room Te and Ue it will be fully setup.

- If you want to use your PC or Mac, you need to prepare the development environment there. To do that, follow the instruction in the file “Setup the development environment” that you can find in the course page.

Outline

- Version Control Systems & GIT
- Playing with GIT
- The working tree
- Remote repositories and best practices

- Homework
- **Resources**

- Final project overview

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/

Outline

- Version Control Systems & GIT
- Playing with GIT
- The working tree
- Remote repositories and best practices

- Homework
- Resources

- **Final project overview**

What's the spirit here

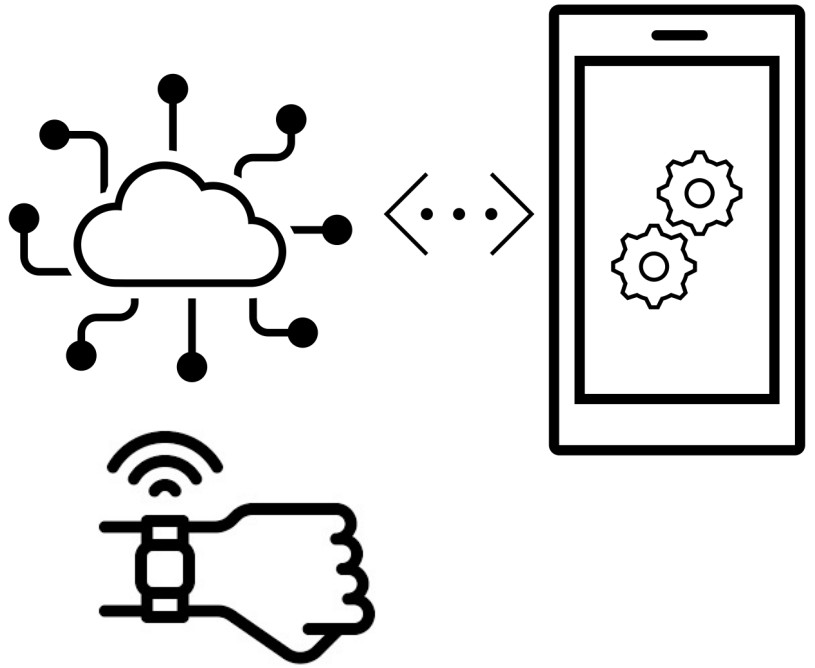
- The goal is learning **"How to do SOMETHING using wearable devices"**

- What **I DO** expect from you
  - To use your imagination to create that SOMETHING
  - To learn to build something from scratch
  - To use code built by others
  - To work together
  - To use Google and StackOverflow

- What **I DO NOT** expect from you
  - To create the new Google
  - To reinvent the wheel (copying others code is fine)
  - To create extra complicated UI with animations and stuff
  - To implement sophisticated design pattern (even if it would be a nice thing to do)

Project structure

- The project consists of building an app for iOS or Android that collects user data from a wearable device through Web APIs, stores them, visualizes them, and does some tricks with them.



- Core functionalities:
  - User authentication and management
  - Data collection
  - Data persistence
  - Data visualization and presentation

- Additional fuctionalities → It's up to you!
  Some examples:
  - Run some analysis on data and provide suggestions to the user
  - Implement some literature algorithm
  - …

Grading

- What are the grading criteria
  - Originality of the additional feature
  - Quality of teamwork
  - Compliance to core functionalities

- The max grade is 32/30 (31 or 32 = 30L)

- How the grade is split?
  - 20 points: core functionalities
  - 8 points: additional app functionalities
  - 4 points: teamwork quality