

Version Control with git and github

R. Pastor-Satorras

Departament de Física
UPC

Eines Informàtiques Avançades / EIA

Version control systems

From computer science:

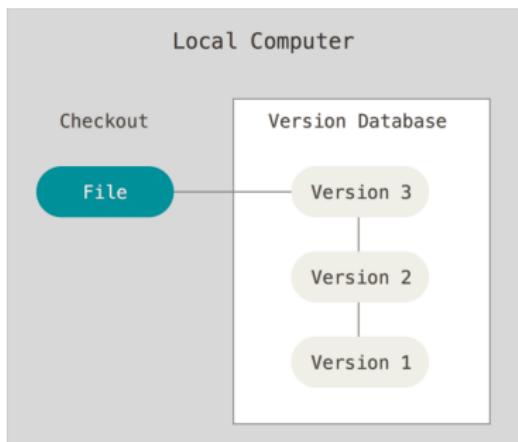
- Version control is a system than records changes performed to a file or set of files in time, and that allows to recall specific versions later

Version control systems permit:

- Revert files back to a previous state
- Revert an entire project back to a previous state
- Compare changes over time
- Know who and when last modified something that does not work
- Recover if you screw things up severely
- Work on different, parallel branches of the code, implementing and experimenting new ideas
- Manage work in the code by a team of people

Local version control systems

Poor-man's VCS

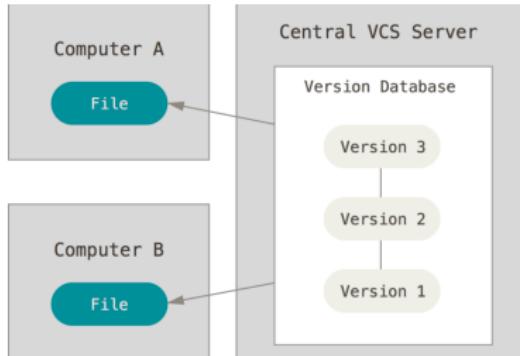


- Copy files into another time-stamped directory
- Allows to keep safe different versions of the code, separated from our working directory
- Error prone
 - ▶ Easy to forget in which directory you are, and mess with your saved versions
- Drawback: Everything is on the same disk, if it fails, all is lost

And how do you collaborate in this framework?

Centralized version control systems

Specially to favor collaboration, but can be used stand-alone

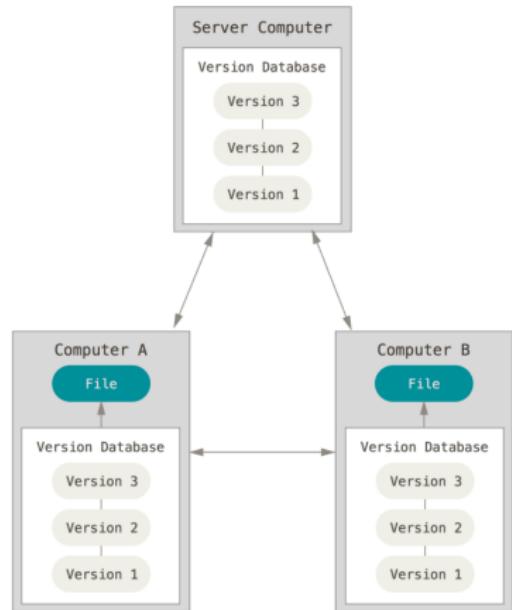


- A single server contains all versioned files
- A number of clients check out files from the server
- The files are updated and later committed back to the server
- Drawback: the single point of failure that the centralized server represents
- If the disk of the server becomes corrupted, all is lost

Distributed version control systems

Latest, most popular VCS technology

- Clients don't just check out the latest version of files, they fully mirror the repository
- Every client is a clone of the server repository; if any server dies, any of the client repositories can be copied back up to the server to restore it
- Clients can work in parallel on different parts of the code, and commit the changes to the central server when they are done



The git distributed version control systems

git is one of the most popular distributed version control systems.

Developed by Linus Torvalds in need of a VCS capable for the development of the Linux kernel

git is extremely powerful

- Ex. code base of Linux kernel (mainline 6.2): 2.6 million lines of code distributed over 79.400 files and 1.4 Gb of disk space, subject by all kinds of complex manipulations by hundreds of contributors simultaneously

However ...

Git is a saw with no guard that makes it easy to cut your own arm off.

But it also comes with an easy arm reattachment kit.

And you can even attach the arm to your knee if you want.

—Wayne Conrad

We have to work carefully with git

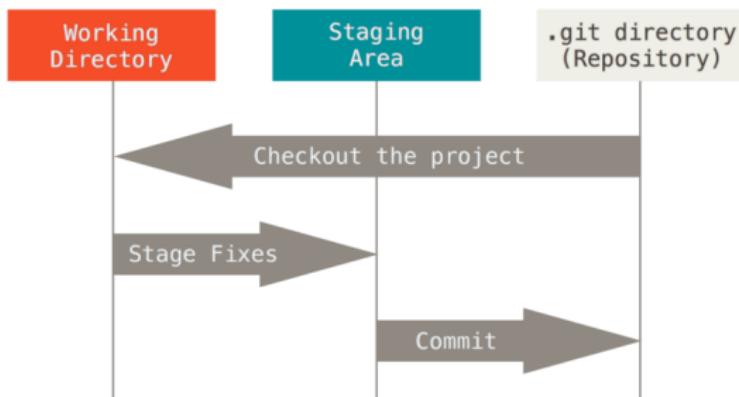
How git works

- git stores complete snapshots of a project, identified by a *repository*, i.e. a folder
- Operations consist essentially in adding data to create and update the different snapshots. Since there are no deletions, it is very difficult to destroy your work
- git performs a check-sum of the data, so that is impossible to change the contents of the folder without git knowing it
- git identifies many things by a SHA-1 (*Secure Hash Algorithm*) hash string, of the form

24b9da6552252987aa493b52f8696cd6d3b00373

We refer to these strings (just the first few characters) to identify the different changes and snapshots

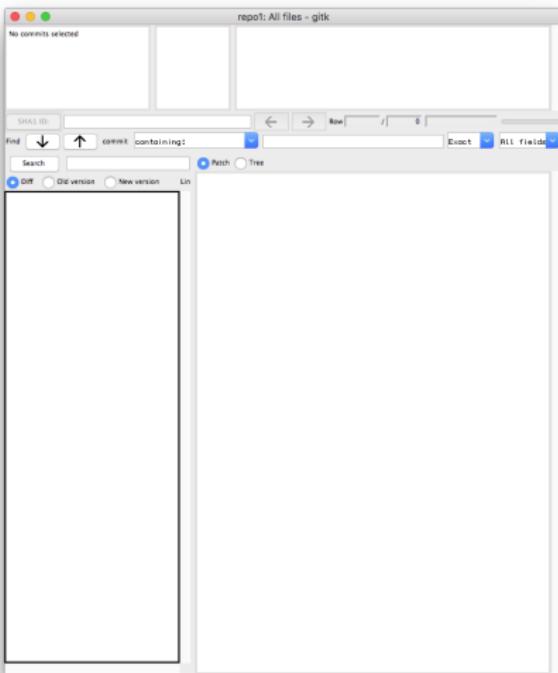
The three states of git



- Define a folder as a repository
- Modify files on the folder (working directory)
- Stage files, as soon as you are done working with them, on the *staging* area. They are not yet saved in this area, just ready to be saved.
- Do a *commit*, taking the files in the staging area and permanently storing a snapshot of the project onto the repository

Local version control with git

- Start version controlling in our computer, to keep track of the work inside a working directory
- We will use git on the command line
- You can also use GUI versions (ex. gitk)



Installing git

- Go to the Git website and follow instructions for your platform
- Use a software package manager, such as chocolatey, homebrew, apt, yum, etc

You have it installed in the computers in the room. Open a terminal (preferably in Linux) and get ready to git...

The git command

```
:> git --version  
git version 2.44.0
```

```
:> git --help  
usage: git [-v | --version] [-h | --help] [-C <path>] [-c <name>=<value>]  
          [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]  
          [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--bare]  
          [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]  
          [--super-prefix=<path>] [--config-env=<name>=<envvar>]  
          <command> [<args>]
```

These are common Git commands used **in** various situations:

start a working area (see also: `git help tutorial`)
`clone` Clone a repository into a new directory
`init` Create an empty Git repository or reinitialize an existing one

work on the current change (see also: `git help everyday`)
`add` Add file contents to the index
`mv` Move or rename a file, a directory, or a symlink
`restore` Restore working tree files
`rm` Remove files from the working tree and from the index

examine the `history` and state (see also: `git help revisions`)
`bisect` Use binary search to find the commit that introduced a bug
`diff` Show changes between commits, commit and working tree, etc
`grep` Print lines matching a pattern
`log` Show commit logs
`show` Show various types of objects
`status` Show the working tree status

Initializing git

First of all, we can provide some initial information about ourselves to git:

```
:> git config --global user.name "R Pastor"  
:> git config --global user.email romualdo.pastor@upc.edu  
:> git config --global core.editor vim
```

git will use the information to tag our commits.

Checking the status

```
:> git config --list  
credential.helper=osxkeychain  
user.name=R Pastor  
user.email=romualdo.pastor@upc.edu  
core.editor=vim
```

Creating a repository

```
:> mkdir project
:> cd project/
:project> git init
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:     git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:     git branch -m <name>
Initialized empty Git repository in $FOLDER/project/.git/

:project> git status
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)
```

You can change the default initial branch name with the command

```
:> git config --global init.defaultBranch main
```

Inside the repository

```
:project> ls -alg
total 0
drwxr-xr-x@ 3 staff    96 Feb 11 12:49 .
drwxr-xr-x@ 3 staff    96 Feb 11 12:48 ..
drwxr-xr-x@ 9 staff   288 Feb 11 12:50 .git
```

Inside the repository

```
:project> ls -alg
total 0
drwxr-xr-x@ 3 staff    96 Feb 11 12:49 .
drwxr-xr-x@ 3 staff    96 Feb 11 12:48 ..
drwxr-xr-x@ 9 staff   288 Feb 11 12:50 .git
```

```
:project> ls -alg .git/
total 24
drwxr-xr-x@ 9 staff   288 Feb 11 12:50 .
drwxr-xr-x@ 3 staff    96 Feb 11 12:49 ..
-rw-r--r--@ 1 staff    23 Feb 11 12:49 HEAD
-rw-r--r--@ 1 staff   137 Feb 11 12:49 config
-rw-r--r--@ 1 staff    73 Feb 11 12:49 description
drwxr-xr-x@ 15 staff   480 Feb 11 12:49 hooks
drwxr-xr-x@ 3 staff    96 Feb 11 12:49 info
drwxr-xr-x@ 4 staff   128 Feb 11 12:49 objects
drwxr-xr-x@ 4 staff   128 Feb 11 12:49 refs
```

Staging files

```
:project> vim hello.f08      # Adding a file
:project> cat hello.f08
program hello
    implicit none

    print *, "Hello World!"
end program hello
```

```
:project> git add hello.f08      # staging a file
```

```
:project> git status      # checking the status
On branch master
```

No commits yet

Changes to be committed:

```
(use "git rm --cached <file>..." to unstage)
    new file:   hello.f08
```

Untracked files:

```
(use "git add <file>..." to include in what will be committed)
    hello.f08~
```

Ignoring files

Text editors usually create back-up files (the `*.*~` file in the present case), which we do not want to keep track of

We can specify files to ignore in a `.gitignore` file

```
:project> vim .gitignore      # Edit .gitignore
:project> cat .gitignore
*~
.*
```

```
:project> git status      # check status again
On branch master
```

No commits yet

Changes to be committed:

(use "`git rm --cached <file>...`" to unstage)
new file: hello.f08

Saving (committing) a snapshot

```
# committing staged files
:project> git commit -m "Initial file"
[master (root-commit) b2d7f48] Initial file
 1 file changed, 5 insertions(+)
 create mode 100644 hello.f08
```

```
# checking
```

```
:project> git status
On branch master
nothing to commit, working tree clean
```

The git commit command

- Saves the snapshot, officially called a “revision”
- Gives that snapshot a unique ID number (a revision hash), here b2d7f48, which we use to identify it
- Names you as the author of the changes
- Allows you, the author, to add a message [-m “Initial file”]

Viewing the history

```
:project> git log  
commit b2d7f4886d917ede42791f8dc6156b4c5ab9d83d (HEAD -> master)  
Author: R Pastor <romualdo.pastor@upc.edu>  
Date:   Sat Feb 11 13:02:54 2023 +0100
```

Initial file

The git log command

- Prints the logged metadata for each commit
- Each commit possesses a unique (hashed) identification number that can be used to refer to that commit (b2d7f48, the first numbers are enough)
- Records the date and time at which the commit occurred.
- The log message for each commit is printed along with that commit.

Making and committing changes

```
# edit the file
:project> vim hello.f08
:project> cat hello.f08
program hello
    implicit none
    character(len=50) :: name

    print *, "Enter your name"
    read *, name

    print *, "Hello ", name, "!"
end program hello

# checking status
:project> git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
            modified:   hello.f08

no changes added to commit (use "git add" and/or "git commit -a")
```

Making and committing changes

```
# stage for the snapshot
:project> git add hello.f08
# commit
:project> git commit -m "Improved program with name"
[master 303135f] Improved program with name
 1 file changed, 4 insertions(+), 2 deletions(-)
```

Tools for git

At some point, we might be lost as to whether we have stuff in the stage area, if we have to make a commit, etc.

Some shells in the terminal can help us at this point

- fish shell with *tide* prompt

```
-/W/T/E/project
● > ~ /Work/Teaching/EIA/project # master > vim hello.f08
● > ~ /Work/Teaching/EIA/project # master !1 > git add hello.f08
● > ~ /Work/Teaching/EIA/project # master +1 > vim hello.py
● > ~ /Work/Teaching/EIA/project # master +1 ?1 > git add hello.py
● > ~ /Work/Teaching/EIA/project # master +2 > git commit -m "Initial files"
[master c405622] Initial files
 2 files changed, 7 insertions(+)
 create mode 100644 hello.py
● > ~ /Work/Teaching/EIA/project # master >
```

What have we done?

```
# check status
:project> git status
On branch master
nothing to commit, working directory clean
:project> git log
commit 303135fb1ab18ac95447812e214be66284c6fc4b (HEAD -> master)
Author: R Pastor <romualdo.pastor@upc.edu>
Date:   Sat Feb 11 14:09:12 2023 +0100
```

Improved program with name

```
commit b2d7f4886d917ede42791f8dc6156b4c5ab9d83d
Author: R Pastor <romualdo.pastor@upc.edu>
Date:   Sat Feb 11 13:02:54 2023 +0100
```

Initial file

Tools for git

lazygit is a useful terminal command to summarize the status of a git repository

The screenshot shows the lazygit interface running in a terminal window. The title bar reads "lazygit ~/W/T/E/T/project". The interface is organized into panels:

- [1]-Status**: Shows "project → main".
- [2]-Files - Worktrees - Submodules**: An empty panel labeled "0 of 0".
- [3]-Local branches - Remotes - Tags**: Shows "* main" and "6d experiment".
- [4]-Commits - Reflog**: A list panel showing a single commit: "ba8ddc5) [MP] o First". It is highlighted with a blue border. Below it, the text "1 of 1" is visible.
- [5]-Stash**: An empty panel labeled "0 of 0".

The right side of the interface displays detailed commit information for the selected commit:

```
Patch
commit ba8ddc53882e0b240ae6491ff975c29da55c9e20 (HEAD → main, experiment)
Author: Romualdo Pastor Satorras <romu@sinera.upc.es>
Date: Mon Feb 24 15:30:45 2025 +0100

First
---
pipa.py | 2 ++
1 file changed, 2 insertions(+)

diff --git a/pipa.py b/pipa.py
new file mode 100644
index 0000000..bbc7ca1
--- /dev/null
+++ b/pipa.py
@@ -0,0 +1,2 @@
+print("Hello world!")
+
```

At the bottom, there is a "Command log" section with the tip: "Random tip: You can page through the items of a panel using ',' and '.'".

At the very bottom, a footer bar contains the following commands: Reward: r | Drop: d | Edit: e | Amend: A | Checkout: <space> | Reset: g | Copy (cherry-pick): C | Keybindings: ? | Donate Ask Question 0.47.2

Making more commits

```
:project> vim hello.f08
:project> cat hello.f08
program hello
    implicit none
character(len=50) :: name, surname

print *, "Enter your name"
read *, name
print *, "Enter your surname"
read *, surname

print *, "Hello ", name, surname, "!"
end program hello
```

```
:project> git add hello.f08
```

```
:project> git commit -m "More improvements"
[master 4da8be2] More improvements
 1 file changed, 5 insertions(+), 6 deletions(-)
```

- Now we have three snapshots, each corresponding to the three commits (hash numbers) made

Looking at the differences

```
# status of the repository, with differences in snapshots
:project> git log -p
commit 4da8be2b7cb696ee1a6154ed09d181e5517139a5 (HEAD -> master)
Author: R Pastor <romualdo.pastor@upc.edu>
Date:   Sat Feb 11 14:11:19 2023 +0100
```

More improvements

```
diff --git a/hello.f08 b/hello.f08
index 51c2cb0..86c700d 100644
--- a/hello.f08
+++ b/hello.f08
@@ -1,7 +1,6 @@
 program hello
- implicit none
- character(len=50) :: name
- print *, "Enter your name"
- read *, name
- print *, "Hello ", name, "!"
-end program hello
+implicit none
+character(len=50) :: name, surname
+print *, "Enter your name" read *, name
+print *, "Enter your surname" read *, surname
+print *, "Hello ", name, surname, "!" end program hello
```

```
commit 303135fb1ab18ac95447812e214be66284c6fc4b
Author: R Pastor <romualdo.pastor@upc.edu>
Date:   Sat Feb 11 14:09:12 2023 +0100
```

Improved program with name

```
diff --git a/hello.f08 b/hello.f08
```



Reverting to a previous snapshot

```
# log in one line format
:project> git log --pretty=oneline
4da8be2b7cb696ee1a6154ed09d181e5517139a5 (HEAD -> master) More improvements
303135fb1ab18ac95447812e214be66284c6fc4b Improved program with name
b2d7f4886d917ede42791f8dc6156b4c5ab9d83d Initial file
```

HEAD -> master indicates the present snapshot

```
# present file
:project> cat hello.f08
program hello
    implicit none
    character(len=50) :: name, surname

    print *, "Enter your name"
    read *, name
    print *, "Enter your surname"
    read *, surname

    print *, "Hello ", name, surname, "!"
end program hello
```

Reverting to a previous snapshot

```
# bring the first version, identified by the first 7 digits of the hash
:project> git reset --hard 8715df
HEAD is now at b2d7f48 Initial file

:project> cat hello.f08
program hello
    implicit none

    print *, "Hello World!"
end program hello
# bring again the last version
:project> git reset --hard 4da8be2b7cb696
HEAD is now at 4da8be2 More improvements
```

You can use more than seven characters of the SHA string

Branches in git

Branches are parallel instances of a repository that can be edited and version controlled in parallel.

They are useful for pursuing various implementations experimentally or maintaining a stable core while developing separate sections of a code base.

List the branches in a repository

```
# Listing branches
```

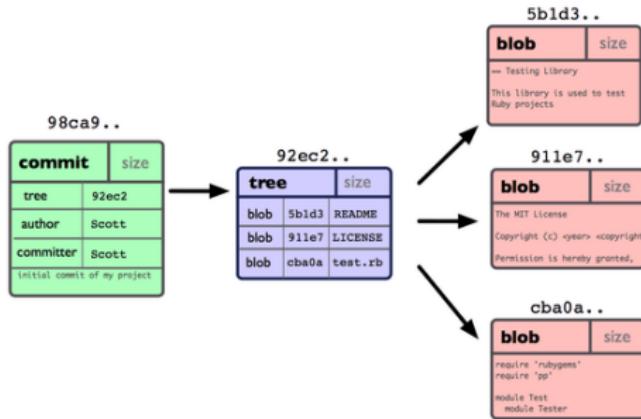
```
:project> git branch  
* master
```

The “master” branch is created when the repository is initialized and the first commit made.

This is the default branch and is conventionally used to store a clean master version of the source code.

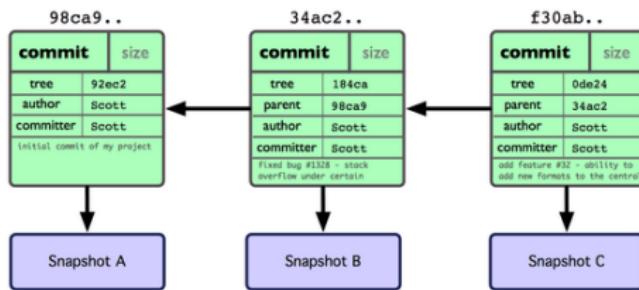
Summary: how git works

A Git repository with an initial commit contains objects: one blob for the contents of each of your files, one tree that lists the contents of the directory and specifies which file names are stored as which blobs, and one commit with a pointer to that root tree and all the commit metadata.



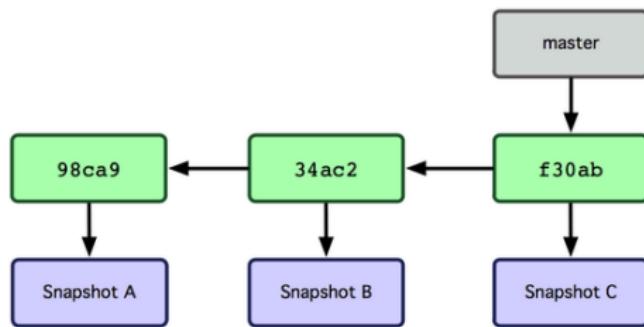
Summary: how git works

If you make some changes and commit again, the next commit stores a pointer to the commit that came immediately before it. After two more commits, your history might look something like this



Summary: how git works

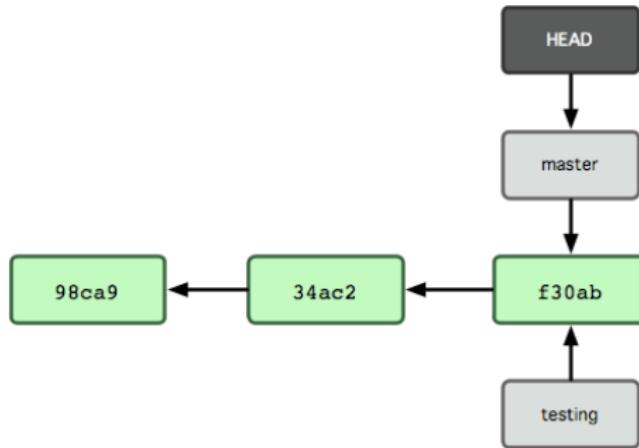
A branch in Git is simply a lightweight movable pointer to one of these commits. The default branch name in Git is `master`. As you initially make commits, you're given a `master` branch that points to the last commit you made. Every time you commit, it moves forward automatically.



Branches in git

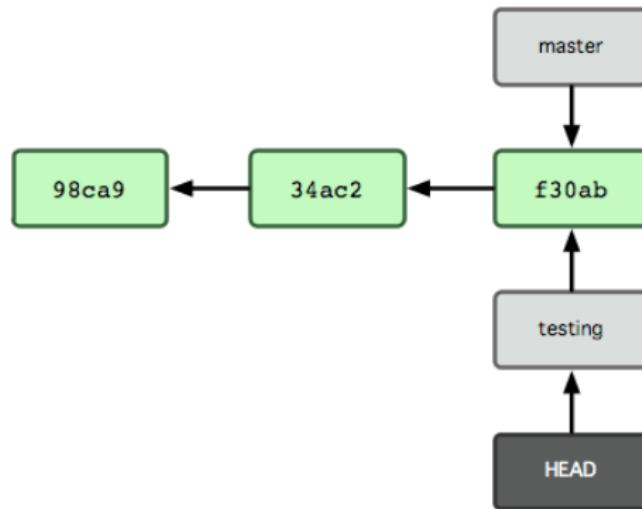
When you create a new branch, a new pointer is created for you to move around.

Git knows in which branch you are by keeping a special pointer called HEAD, pointing to the local branch you're currently on



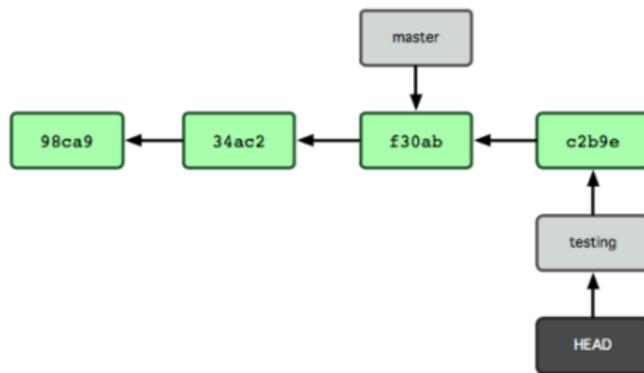
Branches in git

You can move to the newly created branch. HEAD will then point to it.
Now you can start performing changes to your code



Branches in git

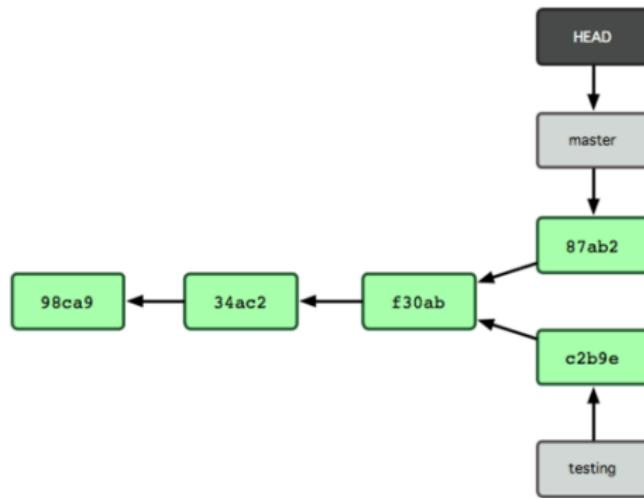
You can start making commits in the new branch. The HEAD in the branch moves forward, but your master branch still points to the commit you were on when you changed to the branch



Branches in git

You can switch back to `master`, perform some more changes and commits. All changes take place in isolated branches, and it is possible to go back and forth among them.

When you are done with the new branch you can delete it, keep it as a separate code base, or merge it with `master`



Creating a branch

```
# Creating a new branch  
:project> git branch experiment  
# Listing branches  
:project> git branch  
    experiment  
* master
```

We have now two branches, master and experiment. The star * indicates the branch in which we are

To switch to a branch

```
# Switch to branch  
:project> git checkout experiment  
Switched to branch 'experiment'  
# list  
:project> git branch  
* experiment  
    master
```

Working on a branch

```
# Add a file to the branch
:project> vim hello.c
:project> cat hello.c
#include <stdlib.h>
#include <stdio.h>

int main()
{
    fprintf(stderr, "%s\n", "Hello world!");

    exit(0);

}
:project> git status
On branch experiment
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    hello.c

nothing added to commit but untracked files present (use "git add" to track)
:project> git add hello.c
:project> git commit -m "Added C version"
[experiment 8b1b746] Added C version
  1 file changed, 4 insertions(+)
create mode 100644 hello.c
```

We can add stuff to the branch, developing some new idea

Switching branches

```
# check where we are
:project> git branch
* experiment
  master

:project> ls -l    # list files in the repository
-rw-r--r--@ 1 romu  staff   65 Feb 11 14:27 hello.c
-rw-r--r--@ 1 romu  staff    0 Feb 11 14:27 hello.c~
-rw-r--r--@ 1 romu  staff  205 Feb 11 14:21 hello.f08
-rw-r--r--@ 1 romu  staff   75 Feb 11 13:00 hello.f08~

# back to master branch
:project> git checkout master
Switched to branch 'master'
:project> git branch
  experiment
* master

:project> ls -l    # list files in the repository
total 16
-rw-r--r--@ 1 romu  staff    0 Feb 11 14:27 hello.c~
-rw-r--r--@ 1 romu  staff  205 Feb 11 14:21 hello.f08
-rw-r--r--@ 1 romu  staff   75 Feb 11 13:00 hello.f08~
```

Back in `master`, observe that the file "hello.c" has disappeared (it's in the `experiment` branch!). Only the back-up file from the text editor remains on the working directory

Merging branches

Once the experiment is finished, if we are happy with it, we can incorporate it to the master branch

```
# go to the master branch
:project> git checkout master
Already on 'master'
# merge the branch experiment to the master
:project> git merge experiment
Updating 4da8be2..8b1b746
Fast-forward
  hello.c | 4 +++
  1 file changed, 4 insertions(+)
  create mode 100644 hello.c
# now "hello.c" is in master
:project> ls -l
-rw-r--r--@ 1 romu  staff   65 Feb 11 14:31 hello.c
-rw-r--r--@ 1 romu  staff    0 Feb 11 14:27 hello.c~
-rw-r--r--@ 1 romu  staff  205 Feb 11 14:21 hello.f08
-rw-r--r--@ 1 romu  staff   75 Feb 11 13:00 hello.f08~
# delete the now useless branch
:project> git branch -d experiment
Deleted branch experiment (was 8b1b746).
```

Must be done inside the branch master

Merging branches

```
# final status
:project> git branch
* master
:project> git log
ommit 8b1b746b0ecf798f7afaa0386a596457206a8d1d (HEAD -> master)
Author: R Pastor <romualdo.pastor@upc.edu>
Date:   Sat Feb 11 14:28:34 2023 +0100
```

Added C version

```
commit 4da8be2b7cb696ee1a6154ed09d181e5517139a5
Author: R Pastor <romualdo.pastor@upc.edu>
Date:   Sat Feb 11 14:11:19 2023 +0100
```

More improvements

```
commit 303135fb1ab18ac95447812e214be66284c6fc4b
Author: R Pastor <romualdo.pastor@upc.edu>
Date:   Sat Feb 11 14:09:12 2023 +0100
```

```
commit 303135fb1ab18ac95447812e214be66284c6fc4b
Author: R Pastor <romualdo.pastor@upc.edu>
Date:   Sat Feb 11 14:09:12 2023 +0100
```

Improved program with name

```
commit b2d7f4886d917ede42791f8dc6156b4c5ab9d83d
Author: R Pastor <romualdo.pastor@upc.edu>
Date:   Sat Feb 11 13:02:54 2023 +0100
```

Initial file

The last commit has been imported from the branch



The power of branches

- ① Branches in git allow to make temporal explorations of some coding project, developing different ideas in parallel to explore their potential
- ② Once we are satisfied with one or several of them, we can merge them into the master code repository
- ③ Branches allow also for team coding: Several aspects of the same codebase can be worked by different people using parallel branches, which are finally merged when the different pieces are ready

Merging conflicts

If the contents of both `master` and the branch have been changes, the merge has conflicts, and it is aborted. Conflicts must be resolved by hand.

Remote version control with git

The full power of git emerges when it is combined with an online repository hosting system, which allows

- ① Back up online automatically your code (thus making a security copy outside your computer)
- ② Managing files in a collaboration
- ③ Forking remote repositories to modify or extend code of other people

Repository hosting

There are many repository hosting services online, that serve to store and access repositories

- Launchpad
- Bitbucket
- Google Code
- SourceForge
- GitHub
- GitLab

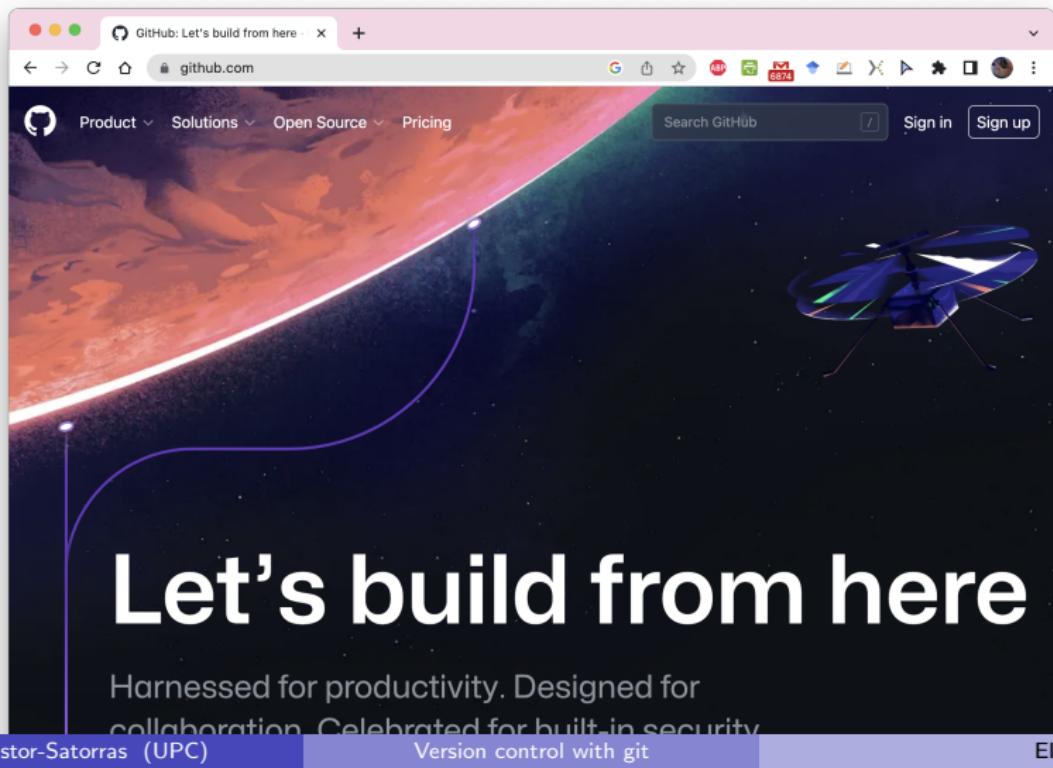
We will use here GitHub, because it is very easy to work in it, providing many tools for browsing, documenting, collaborating, etc

It can be used free, but it has the restriction that all repositories are public (anybody can look at them)

If you want free privacy, go for GitLab

Create a free account on GitHub

Take a moment to create a free account, if you don't have one. You might get email messages at login for confirmation



Creating a repository on GitHub

Go to the main page in your GitHub account and create a new project, clicking in the corresponding big green button. Give it the name "project"

The screenshot shows a web browser window with the URL github.com/new. The title bar says "Create a New Repository". The page has a dark header with the GitHub logo, a search bar, and navigation links for Pull requests, Issues, Codespaces, Marketplace, and Explore. Below the header, the main content area is titled "Create a new repository". It explains that a repository contains all project files, including the revision history, and offers to import an existing one. The "Owner" field is set to "RomuPS" and the "Repository name" field is "project". A note suggests short and memorable names like "jubilant-spork". The "Description (optional)" field contains "A test project repository". Under "Visibility", the "Public" option is selected, with a note that anyone on the internet can see it. The "Private" option is also shown. At the bottom, there's a section for initializing the repository with a README file or an .gitignore file, both of which have checkboxes. The bottom right corner of the browser window shows standard OS X window controls.

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository](#).

Owner * Repository name *

RomuPS / project

Great repository names are short and memorable. Need inspiration? How about [jubilant-spork](#)?

Description (optional)

A test project repository

Public
Anyone on the Internet can see this repository. You choose who can commit.

Private
You choose who can see and commit to this repository.

Initialize this repository with:

Skip this step if you're importing an existing repository.

Add a README file
This is where you can write a long description for your project. [Learn more](#).

Add .gitignore

Creating a repository on GitHub

Watch out: Now GitHub by default does not use the branch name `master`, but `main`. This can create conflicts if you try to synchronize local and remote repositories with different names. Be careful, or choose the default GitHub repository branch name in general Settings, to match the one in your local repository

The screenshot shows a web browser window with the URL `github.com/settings/repositories`. The page is titled "Repositories". On the left, there's a sidebar with account information (RomuPS, Your personal account) and various settings links: Public profile, Account, Appearance, Accessibility, Notifications, Access, Billing and plans, Emails, Password and authentication, and Sessions. The main content area is titled "Repository default branch". It contains a note about choosing a default branch for new repositories, mentioning that it might be "master" or "main". A dropdown menu is set to "master", with an "Update" button next to it. Below this, there's a "Repositories" section showing a list with one item: "EIA-Master". At the bottom right, there are navigation icons for GitHub.

Creating a repository on GitHub

The repository is now empty, and GitHub proposes several ways to fill it.
Do nothing for the moment

The screenshot shows a GitHub repository page for 'RomuPS/project'. The page includes a search bar, navigation links for Pull requests, Issues, Codespaces, Marketplace, and Explore, and buttons for Pin, Unwatch (1), Fork (0), and Star (0). Below the header are links for Code, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. A 'Quick setup' section provides options to 'Set up in Desktop' or use HTTPS or SSH, with the URL <https://github.com/RomuPS/project.git>. It also suggests creating a new file or uploading an existing one, and recommends including a README, LICENSE, and .gitignore. Another section shows command-line instructions for creating a new repository:

```
echo "# project" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/RomuPS/project.git
git push -u origin main
```

At the bottom, there's a section for pushing an existing repository from the command line:

```
git remote add origin https://github.com/RomuPS/project.git
```

Declaring a remote

To synchronize changes between a local repository and a remote repository, the location of the remote must be registered on the local repository

```
# declare remote  
:project> git remote add origin https://github.com/RomuPS/project.git
```

- `remote` makes git register something about a remote repository
- `add` declares the alias of the remote
- `origin` is the conventional alias for the repository
- URL is copied from the GitHub page

```
# check remotes  
:project> git remote -v  
origin      https://github.com/RomuPS/project.git (fetch)  
origin      https://github.com/RomuPS/project.git (push)
```

Accessing github

In order to access github for remotely fetching and pushing from your computer, you must create a personal access token for security reasons (no username-password use is possible).

Steps: Access the general Settings of your account

Signed in as RomuPS

- Your profile
- Your repositories
- Your projects
- Your stars
- Your gists
- Your sponsors

Upgrade

Try Enterprise

Feature preview

Help

Settings

Sign out

Discover interesting projects and people to populate your personal news feed.

Your news feed helps you keep up with recent activity on repositories you [watch](#) or [star](#) and people you [follow](#).

[Explore GitHub](#)

When you take actions across GitHub, we'll provide links to that activity here.

ProTip! The feed shows you events from people you [follow](#) and repositories you [watch](#) or [star](#).

Subscribe to your news feed

Accessing github

Go to Developer settings (at the bottom of Settings)

The screenshot shows the GitHub profile settings page. On the left, there's a sidebar with links like Copilot, Pages, Saved replies, Security, Code security and analysis, Integrations, Applications, Scheduled reminders, Archives, Security log, and Sponsorship log. At the bottom of the sidebar, there's a link to 'Developer settings'. The main content area has sections for Company (with a note about linking to a GitHub organization) and Location. Below these are two checkboxes: 'Display current local time' (unchecked) and 'Make profile private and hide activity' (unchecked). A note explains that enabling the activity hiding option will also hide contributions from social features. There's a green 'Update profile' button. At the bottom, there's a 'Contributions & Activity' section with two more checkboxes: 'Include private contributions on my profile' (unchecked) and another 'Make profile private and hide activity' checkbox (unchecked). A note for this second checkbox states that it will hide contributions and activity from the GitHub profile and social features. There's also a 'Read more' link. At the very bottom, there's a 'Update preferences' button.

Accessing github

Select Personal access tokens, Tokens (classic) and Generate new token (classic)

The screenshot shows a web browser window with the URL github.com/settings/tokens. The page is titled "Personal Access Tokens (Classic)". On the left, there is a sidebar with links: GitHub Apps, OAuth Apps, Personal access tokens (Beta), Fine-grained tokens (Beta), and Tokens (classic). The "Tokens (classic)" link is highlighted with a blue bar. The main content area is titled "Personal access tokens (classic)" and contains the following text:
"Need an API token for scripts or testing? Generate a personal access API."
Below this, it says: "Personal access tokens (classic) function like ordinary OAuth access tokens over HTTPS, or can be used to authenticate to the API over Basic Authentication." To the right of the main content, there are two buttons: "Generate new token" (Beta) and "Generate new token (classic)". The "Generate new token (classic)" button is described as "Fine-grained, repo-scoped". At the bottom of the page, there is a footer with links: GitHub, © 2023 GitHub, Inc., Terms, Privacy, Security, Status, Docs, Contact GitHub, Pricing, API, Training, Blog, About.

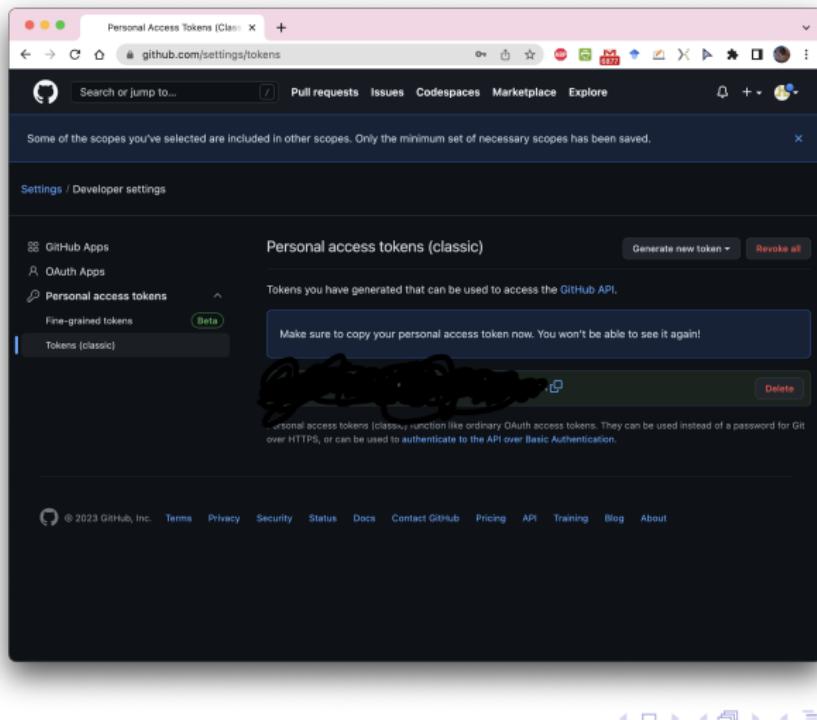
Accessing github

Choose a name for the token, select the expiration of the token, mark the appropriate access options (all if you are not sure), and generate it.

The screenshot shows a web browser window with the URL github.com/settings/tokens/new. The page is titled "New personal access token (classic)". On the left, there's a sidebar with links to GitHub Apps, OAuth Apps, Personal access tokens (selected), Fine-grained tokens (Beta), and Tokens (classic). The main area has a "Note" section with the text "my token" in a text input field. Below it is a "Expiration" section with a dropdown set to "No expiration". A note below says "The token will never expire!". Under "Select scopes", it says "GitHub strongly recommends that you set an expiration date for your token to help keep your information secure. Learn more". The "repo" scope is selected, with a description "Full control of private repositories". Other scopes listed are repo:status, repo_deployment, public_repo, and repo:invite.

Accessing github

In the next screen, copy the token (a long string of characters) somewhere, since you will need it. The token is used instead of the github password



Sending commits (push)

The git push command sends (pushes) commits from a branch of the local repository into the remote repository

```
# push commits
:project> git push origin master
Username for 'https://github.com': RomuPS
Password for 'https://RomuPS@github.com': # Use your token!!
Enumerating objects: 12, done.
Counting objects: 100% (12/12), done.
Delta compression using up to 8 threads
Compressing objects: 100% (9/9), done.
Writing objects: 100% (12/12), 1.22 KiB | 1.22 MiB/s, done.
Total 12 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/RomuPS/project.git
 * [new branch]      master -> master
```

- Push "master" into "origin"
- Requires GitHub username and token

Sending commits (push)

Now our GitHub project is populated with the files from the local repository

The screenshot shows a GitHub repository named "RomuPS/project". The repository is public and has 4 commits. The latest commit was made by "EIA-Master" and added a C version. The repository has 0 stars, 1 watcher, and 0 forks. It also has 0 releases published.

Commits:

- EIA-Master Added C version (8b1b746, 5 hours ago)
- hello.c (Added C version, 5 hours ago)
- hello.f08 (More improvements, 6 hours ago)

About:

A test project repository

Statistics:

- 0 stars
- 1 watching
- 0 forks

Help:

Help people interested in this repository understand your project by adding a README.

Add a README

Releases:

No releases published
Create a new release

Packages:

No packages published
Publish your first package

Sending commits (push)

Now we can make more commits in the local repository and push them on the remote using the same steps

```
# Add a new file
:project> vim hello.py
:project> cat hello.py
print("Hello World!")
:project> git add hello.py
# commit the changes
:project> git commit -m "Added python version"
[master 19a5b0e] Added python version
 1 file changed, 2 insertions(+)
 create mode 100644 hello.py
# push the changes
:project> git push origin master # no password/token needed
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 327 bytes | 327.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/RomuPS/project.git
 8b1b746..19a5b0e master -> master
```

The changes have been pushed to the remote GitHub (check!)

Pushing branches

```
:project> git branch experiment
:project> git branch
  experiment
* master
:project> git checkout experiment
Switched to branch 'experiment'
:project> vim hello.jl
:project> git add hello.jl
:project> git commit -m "Added Julia version"
:project> git push origin master
Everything up-to-date
# oops! We must push the branch explicitly
:project> git push origin experiment
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 354 bytes | 354.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'experiment' on GitHub by visiting:
remote:     https://github.com/RomuPS/project/pull/new/experiment
remote:
To https://github.com/RomuPS/project.git
 * [new branch]      experiment -> experiment
```

We can check in the GitHub page that now two branches appear, "master" and "experiment"

Cloning a Github repository

Github allows to *clone* an entire repository of another person into our local machine (that is, if the repository is public)

We will have the complete repository, with all its contents, at our disposition, to edit and play with it

Example: Let us clone a nice repository of emacs configurations for people trying to escape from vim: **Emacs for the stubborn martian vimmer**

<https://github.com/hlissner/doom-emacs>,

```
:> git clone https://github.com/hlissner/doom-emacs ./emacs.d
Cloning into './emacs.d'...
remote: Enumerating objects: 118647, done.
remote: Counting objects: 100% (20/20), done.
remote: Compressing objects: 100% (17/17), done.
remote: Total 118647 (delta 4), reused 6 (delta 3), pack-reused 118627
Receiving objects: 100% (118647/118647), 29.34 MiB | 21.90 MiB/s, done.
Resolving deltas: 100% (82909/82909), done.
```

The contents is in our emacs configuration folder `.emacs.d`

We can now start messing and playing with it.

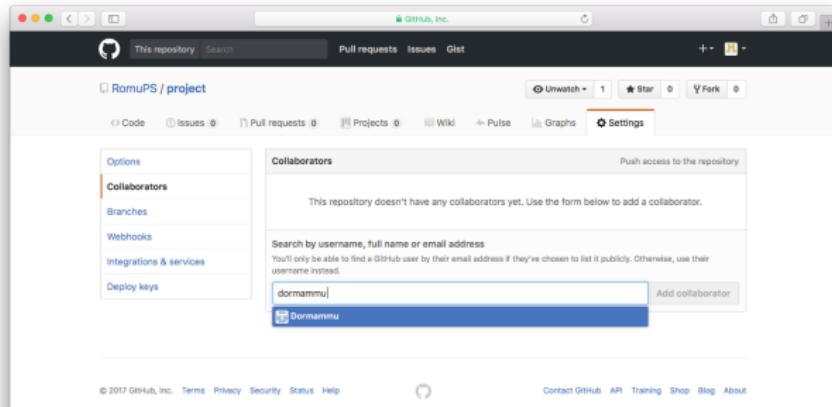
But... we don't have permissions to push anything into the github repository

Collaborating with GitHub

GitHub allows the collaboration of different users on the same repository

There are two possibilities for collaboration:

- In the tab Settings, define a collaborator, by searching his/her username or email address. This gives the collaborator(s) full access to push data to the remote repository
- While this is simple, it can lead to problems
 - ▶ Two people trying to push the same file, with different versions
 - ▶ Can lead to conflicts, which are difficult to observe and fix, since everybody has the same privileges



Cloning and forking

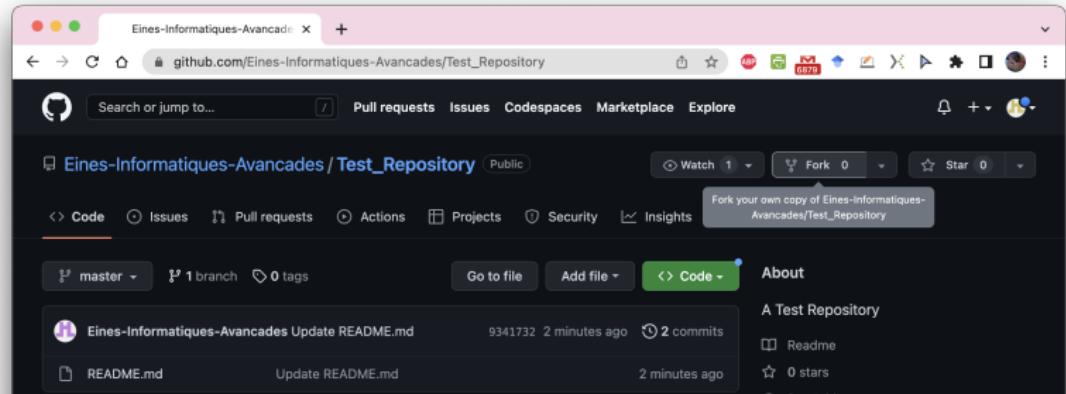
Another approach to collaborating on GitHub consists in **forking** an already existing project

To start the collaboration, some manager creates a main repository, where the main version of the code base will be maintained

Collaborators, on their GitHub page, locate the page of the manager, and the main repository.

On this page, collaborators click on the button Fork

This creates in your own GitHub account a full copy of the main repository



Cloning a remote repository

To have access to the forked repository we just created, we have to clone it, from our GitHub space, into our local machine (use your own github username!!)

```
:> git clone https://github.com/RomuPS/Test_Repository
Cloning into 'Test_Repository'...
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 6 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (6/6), done.
:> cd Test_Repository/
:Test_Repository> ls -l
total 8
-rw-r--r--@ 1 romu  staff  66 Feb 11 20:50 README.md
:Test_Repository> git remote -v
origin      https://github.com/RomuPS/Test_Repository (fetch)
origin      https://github.com/RomuPS/Test_Repository (push)
```

The cloning operation directly links our local to our remote copy of the main repository, with the alias "origin"

In this way, we can start making changes and updating our own version of the main repository

Updating a cloned remote repository

But in the mean time, the main repository may have been changed.

To update the changes in the main repository into our local version, we have to link it to the main, for which we choose the alias "main"

```
:Test_Repository> git remote add main https://github.com/Eines-Informatiques-Avancades/Test_Repository
:Test_Repository> git remote -v
main      https://github.com/Eines-Informatiques-Avancades/Test_Repository (fetch)
main      https://github.com/Eines-Informatiques-Avancades/Test_Repository (push)
origin    https://github.com/RomuPS/Test_Repository (fetch)
origin    https://github.com/RomuPS/Test_Repository (push)
```

Now we can fetch changes in main

```
:Test_Repository> git fetch main
From https://github.com/Eines-Informatiques-Avancades/Test_Repository
 * [new branch]      master      -> main/master

:Test_Repository> git pull main master
From https://github.com/Eines-Informatiques-Avancades/Test_Repository
 * branch            master      -> FETCH_HEAD
Already up to date.
```

Working in a cloned remote repository

We now work on our cloned repository, adding a file

```
:Test_Repository> vim stuff.txt
:Test_Repository> cat stuff.txt
This is a file with really unimportant stuff
```

```
:Test_Repository> git add stuff.txt
:Test_Repository> git commit -m "File with stuff"
[master f3c9935] File with stuff
 1 file changed, 2 insertions(+)
 create mode 100644 stuff.txt
# push to our cloned remote "origin"
:Test_Repository> git push origin master
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 328 bytes | 328.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/RomuPS/Test_Repository
 9341732..f3c9935  master -> master
```

But since we do not have access to the main repository “main”, we cannot directly push there our changes

```
:Test_Repository> git push main master
remote: Permission to Eines-Informatiques-Avancades/Test_Repository.git denied to RomuPS.
fatal: unable to access 'https://github.com/Eines-Informatiques-Avancades/Test_Repository/': The requested URL
```

Pulling to the main remote repository

To pull changes to the main, we have to open a "New pull request" in our GitHub web page, in the Pull requests section

The screenshot shows a GitHub repository page for 'RomuPS/Test_Repository'. The 'Pull requests' tab is selected. The repository is public and forked from 'Eines-Informatiques-Avancades/Test_Repository'. The 'Code' tab is active, showing the 'master' branch (1 commit ahead), 1 branch, and 0 tags. A commit by 'EIA-Master File with stuff' (f3c9935) was made 3 minutes ago, containing 3 commits. The commit details show changes to 'README.md' and 'stuff.txt'. The 'About' section indicates it's a test repository with 0 stars, 0 watching, and 1 fork. The 'Releases' section shows no releases published, and the 'Packages' section shows no packages published.

RomuPS / Test_Repository Public
forked from Eines-Informatiques-Avancades/Test_Repository

<> Code Pull requests Actions Projects Wiki Security Insights Settings

master 1 branch 0 tags Go to file Add file <> Code

This branch is 1 commit ahead of Eines-Informatiques-Avancades:master.

EIA-Master File with stuff f3c9935 3 minutes ago 3 commits

README.md Update README.md 17 minutes ago

stuff.txt File with stuff 3 minutes ago

About

A Test Repository

Readme 0 stars 0 watching 1 fork

Releases

No releases published Create a new release

Packages

No packages published Publish your first package

Test_Repository

This is the README file of the Test Repository

Pulling to the main remote repository

We have to choose the base fork (the main) and our fork, choose the branches we want to pull, add a comment, and submit

The screenshot shows a GitHub pull request comparison interface. At the top, the URL is `github.com/Eines-Informatiques-Avancades/Test_Repository/comp...`. The page title is "Comparing Eines-Informatiques / Test_Repository". Below the title, there are buttons for "Pull requests", "Issues", "Codespaces", "Marketplace", and "Explore". On the right, there are buttons for "Watch 1", "Fork 1", and "Star 0". The main navigation bar includes "Code", "Issues", "Pull requests" (which is highlighted), "Actions", "Projects", "Security", and "Insights".

The main content area is titled "Comparing changes". It says "Choose two branches to see what's changed or to start a new pull request. If you need to, you can also compare across forks." Below this, there are dropdown menus for "base repository: Eines-Informatiques-Avanca...", "base: master", "head repository: RomuPS/Test_Repository", and "compare: master". A green checkmark indicates "Able to merge. These branches can be automatically merged."

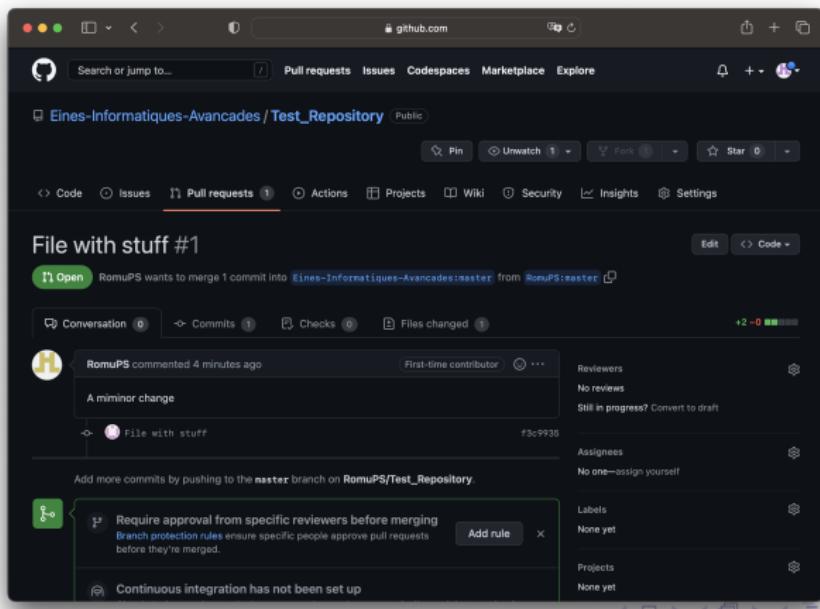
A blue button at the bottom of this section says "Create pull request". Below this, there are summary statistics: "-o 1 commit", "1 file changed", and "1 contributor".

Under the commit summary, it says "-o- Commits on Feb 11, 2023". A single commit is listed: "File with stuff" by "EIA-Master committed 5 minutes ago". There are icons for copy, delete, and refresh next to the commit details.

Pulling to the main remote repository

In the GitHub page of the owner of main, appears a pull request, in the corresponding tab. The owner of main can check it, see if there are no conflicts, and accept (merge) it.

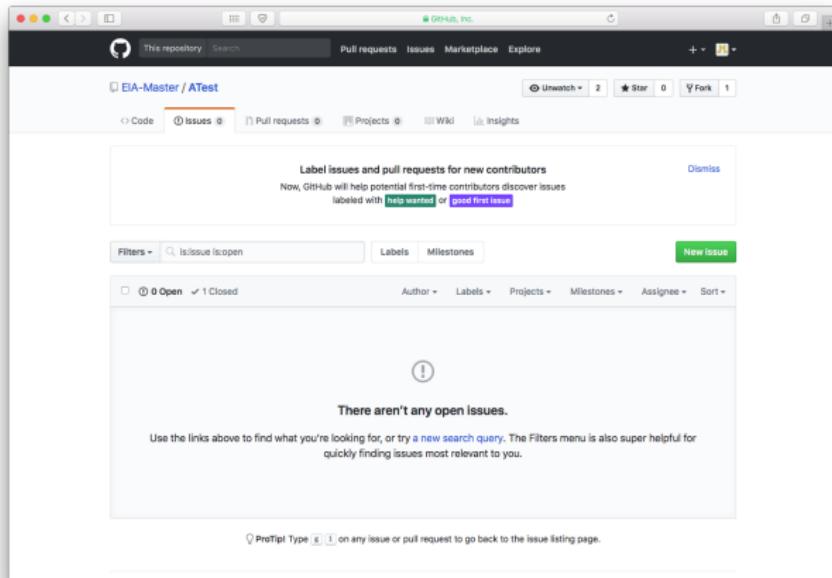
Both remotes (origin and main) will be now synchronized



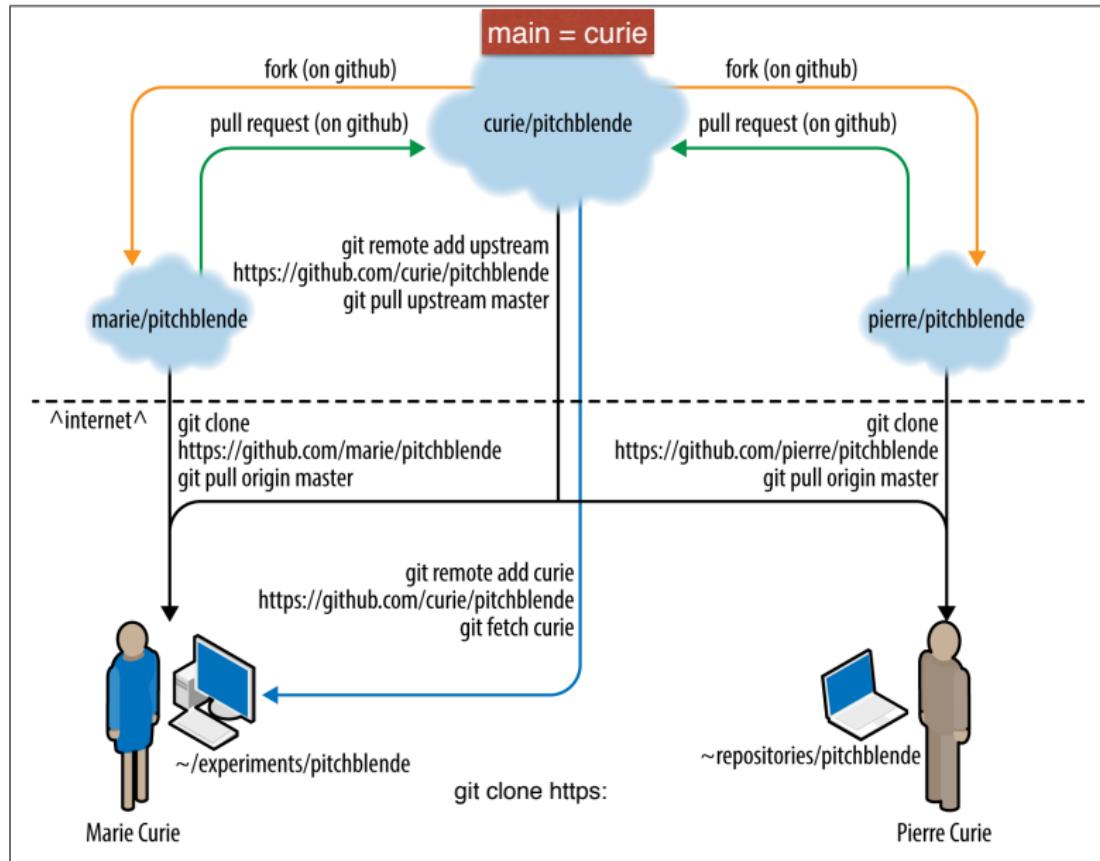
Managing collaboration

Apart from adding new pieces of code, GitHub allows discussion on problems (issues) in the codebase. To start a discussion, we have to open a new issue in the Issues tab of the main repository.

The discussion can be continued by the members of the collaboration, and closed when the problem raised has been solved.



Forking and cloning sum up



Final Remark: README best practices

Objective of the README is to provide information about:

- What is the purpose of the project?
- What are the prerequisites?
- How to install it?
- How to run it?
- Contributors

Best practices:

- Make it to the point
- Make it clear
- Provide examples of installation and running

You can find examples of good README files in the course repository

<https://github.com/Eines-Informatiques-Avancades/awesome-readme>

Final Remark: README best practices

You can write the README using the markdown markup language to organize the structure of the document in sections, add enumerations, figures, references, equations, emphasize, insert snippets of code, etc

● ● ● Quadratic formula

```
1 ## Quadratic formula
2
3 In [elementary algebra][r1], the
**quadratic formula** is the solution of
the [quadratic equation][r2]. With the
above parameterization, the quadratic
formula is:
4
5 $$ x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}
6
7 _Math is hard, let's write some code:_
8
9 ```swift
10 lazy var propertyName: NSDateFormatter =
{
11     let formatter = NSDateFormatter()
12     formatter.dateStyle = .long
13     return formatter
14 }()
15 ```
16
17 ###### inspirational quote
```



Quadratic formula

In [elementary algebra](#), the **quadratic formula** is the solution of the [quadratic equation](#). With the above parameterization, the quadratic formula is:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Math is hard, let's write some code:

```
lazy var propertyName: NSDateFormatter =
{
    let formatter = NSDateFormatter()
    formatter.dateStyle = .long
    return formatter
}()
```

Inspirational Quote

Those who dare to fail miserably can achieve



Collaboration exercise

We are going to develop a fake project in which I will play the role of boss.

- Go to the course GitHub page (Eines-Informatiques-Avancades user) and fork and clone the repository Test_Repository
- Link your own copy to repository folder as main
- Open a branch of the project in your own local folder, add some source code for something simple and merge it into your master
- Make a pull request
- Let us see what happens ...