CINECA

Git

version control

Mattia Mencagli
m.mencagli@cineca.it

22/04/2024

git

Version control vs. GitHub or GitLab

Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later.
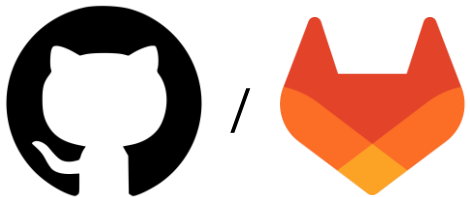
GitHub or **GitLab** are developer platform that allows developers to create, store, manage and share their code through **git** software.

3

git
git clone

You want to get a copy of an **existing** remote git repository?

```
> git clone <remote_url> <local_dir>
```
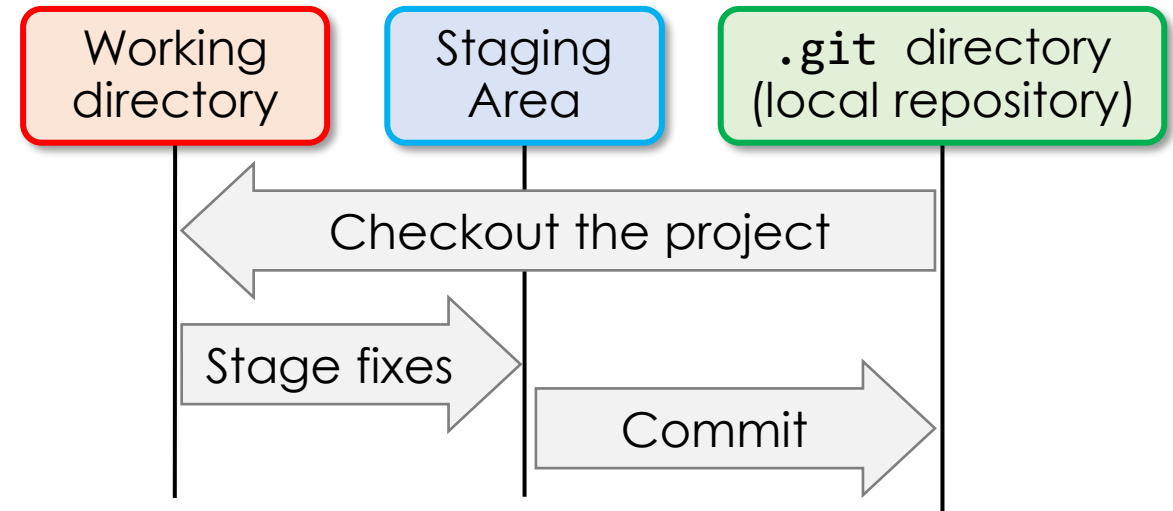
DO NOT TOUCH the `.git` directory.

That creates a directory named `<local_dir>`,

initializes a `.git/` directory inside it (that is your local repository),

and pulls down all the data for that repository.

`<remote_url>`:

HTTP protocol: https://github.com/mattiamencagli/git_introduction.git

SSH protocol: git@github.com:mattiamencagli/git_introduction.git
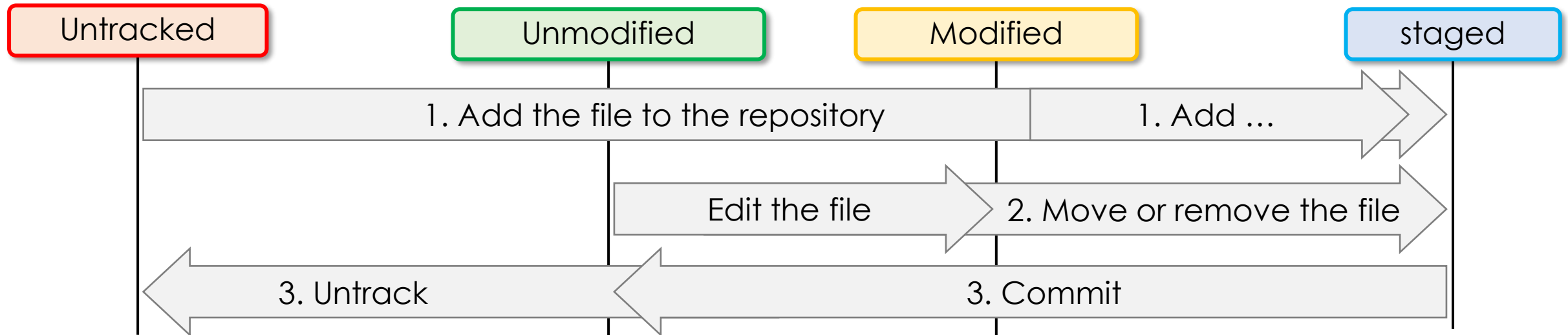
git
git states

- **Modified** means that you have changed the file but have **not committed** yet it to your local repository yet.

- **Staged** means that you have marked a modified file in its current version to go into your next **commit snapshot**.

- **Committed** means that the data is safely stored in your local repository.

Working directory	Staging Area	`.git` directory (local repository)

Checkout the project

Stage fixes

Commit

git thinks about its data like a **stream of snapshots**, each commit point to a snapshot.

git
Files life-cycle

Untracked		Unmodified		Modified		staged

1. Add the file to the repository → 1. Add …

Edit the file → 2. Move or remove the file

← 3. Untrack ← 3. Commit

1.
```
> git add <files>
> git add –A
```

2.
```
> git rm --cached <files>
> git mv <old_f> <new_f>
```

3.
```
> git commit -m "your commit text"
```

git

file status & ignored files

Viewing your staged and unstaged changes:

```
> git status
```

To see what you've changed but not yet staged:

```
> git diff <file>
```

Check the history of your commits:

```
> git log
> git log --all --oneline --decorate --graph
```

A file listing patterns to match the names of files that will be ignored by git actions.

```
> vim .gitignore
```

.gitignore example:

```
*.o
*.x
*.dat
!*ini.dat
build*/
.vscode/
```

git
Undoing things

Improve the previous commit:

```
> git commit --amend
```

Un-stage a staged file:

```
> git restore --staged <file>
> git reset HEAD <file>
```

Un-modify a modified file:

```
> git restore <file>
> git checkout <file>
```

Undo the last local commit,
or the last 3 :

```
> git reset HEAD~1
> git reset HEAD~3
```
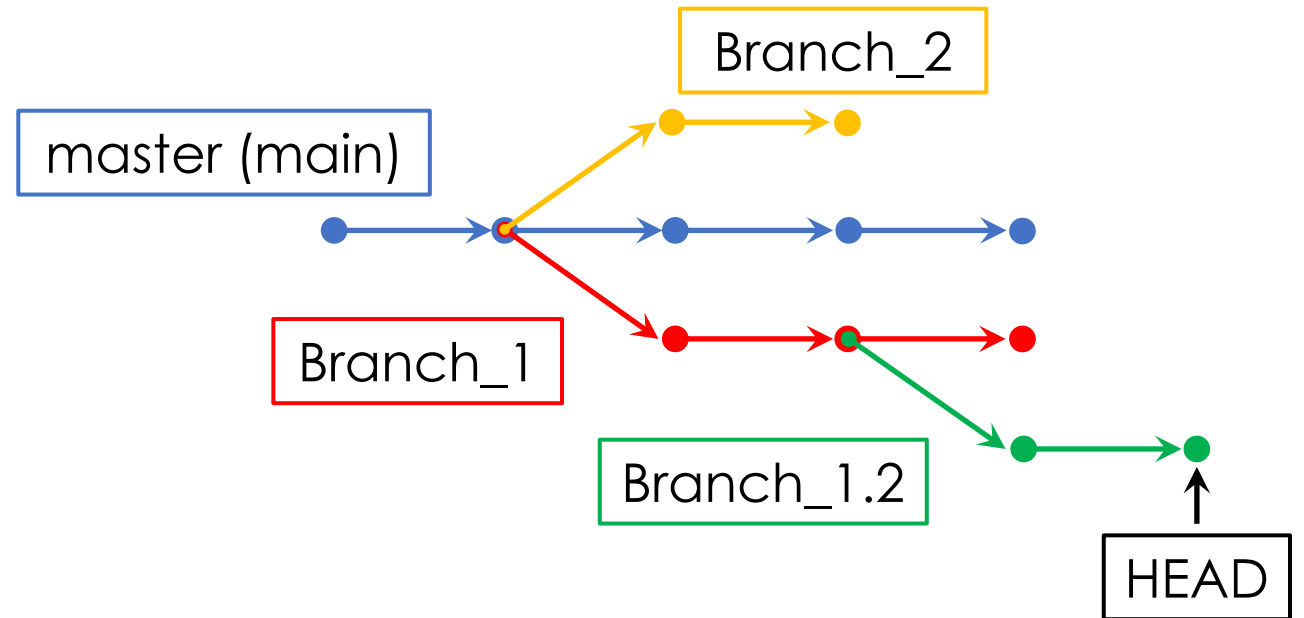
git
Branches

A branch in git is a movable pointer to one of the commits.

Create a new branch:

```
> git branch <branch_name>
```

Switch to an existing branch:

```
> git checkout <branch_name>
```

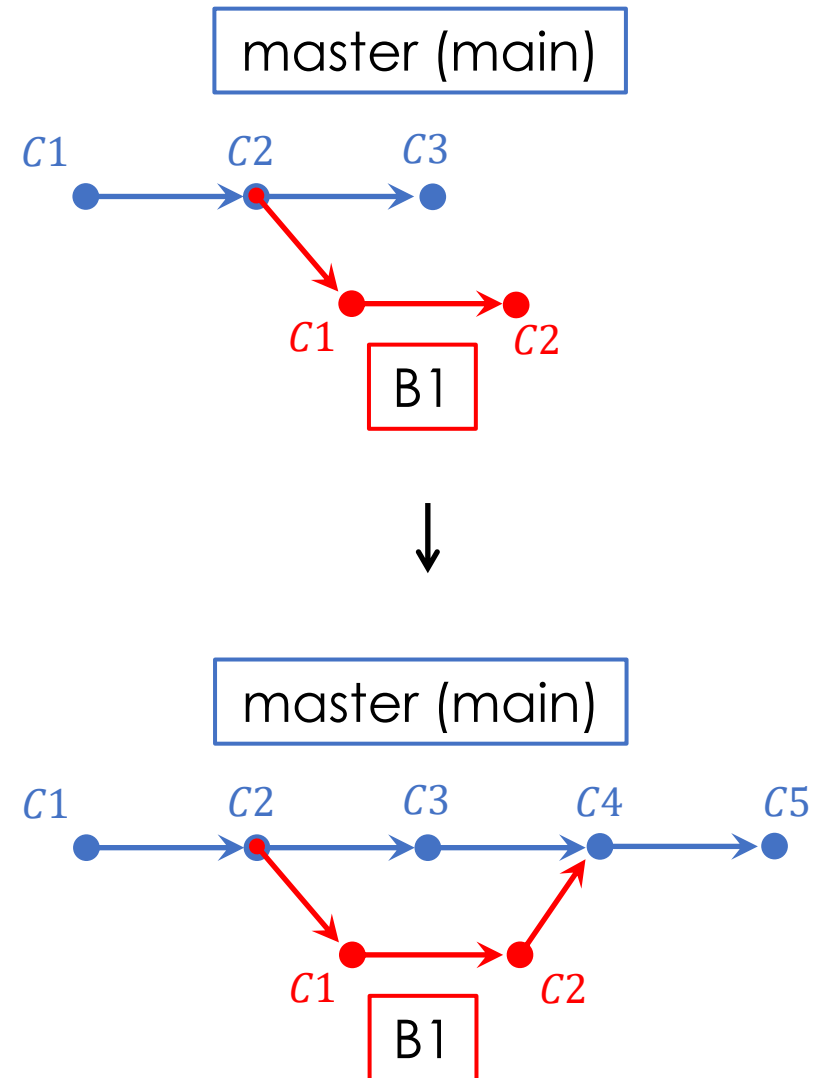HEAD is a particular pointer. It always points to the local branch you're currently on.

Branch_2

master (main)

Branch_1

Branch_1.2

HEAD

git

Merge branches

Merge `<branch>` onto the current branches:

```
> git merge <branch>
```

Merge commit (B1): git creates a **new** commit (snapshot) that results from this three-way merge.
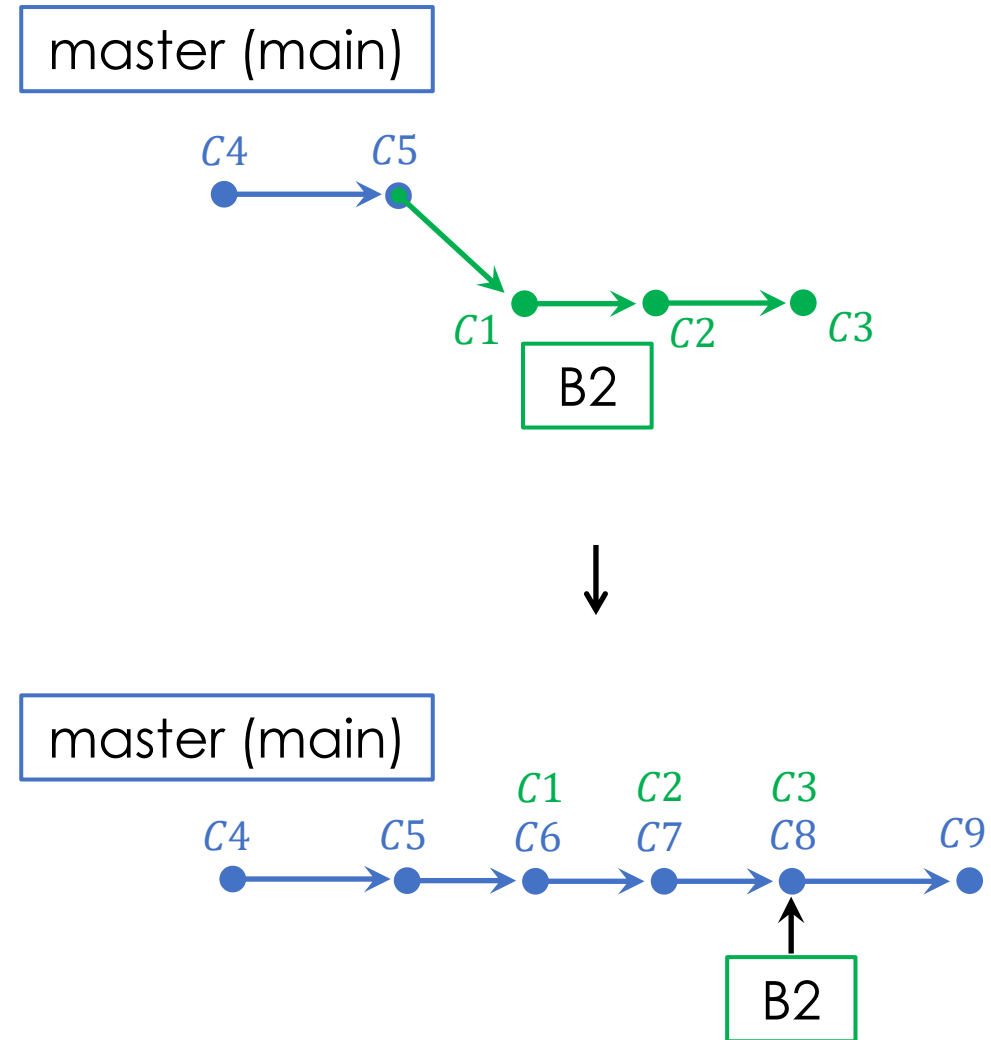[C4 is a new master commit, B1 remain at C2]

git
Merge branches

Merge `<branch>` onto the current branches:

```
> git merge <branch>
```

Fast-forward merge (B2): is a special case of merge that happens if there is **not** a divergent history: git just moves the pointer forward.
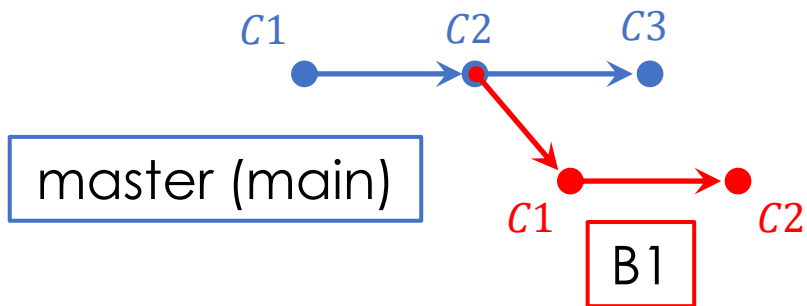*[the master advance to the commits of branch B2]*

master (main)

$C4$ $C5$

$C1$ $C2$ $C3$

B2

master (main)

$C1$ $C2$ $C3$

$C4$ $C5$ $C6$ $C7$ $C8$ $C9$
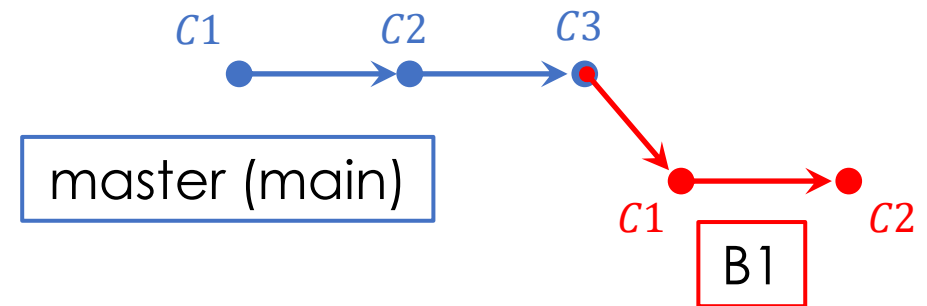
B2

git
Rebase branches

Take all the changes committed in the current branch and reapply them on top of `<branch>`:

```
> git rebase <branch>
```

Before rebase:

After rebase:

Useful to create fast-forward merges instead of merge commits.

git
Remote repository

Remote repositories are versions of your project that are hosted on the Internet or network (e.g., GitHub or GitLab). Very useful to **collaborate** with others, **share** your work, or just having a backup of your code.

If you have **cloned** the remote repository, you can check it:

```
> git remote -v
```

Otherwise, you can **initialize** your new local repository from scratch, and then **connect** it to a new remote repository:

```
> git init
> git remote add <repo_name> <repo_url>
```

git
git config

Configure your most used account to use your git repositories with SSH-key protocol:

```
> git config --global user.email youremail@domain.com
> git config --global user.name "yourname"
```

Configure a local account for a specific repository:

```
> git config --local user.email youremail@domain.com
> git config --local user.name "yourname"
```

Check your configuration:

```
> git config --list
```

Useful option:

```
> git config --global core.editor "vim"
```
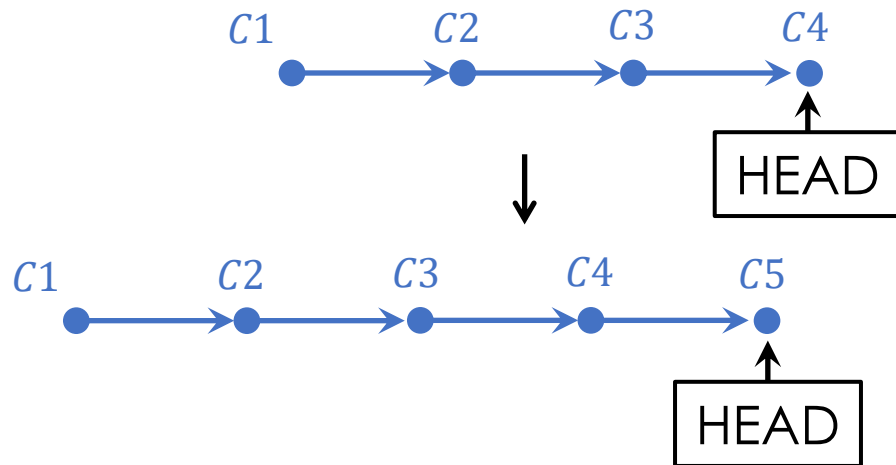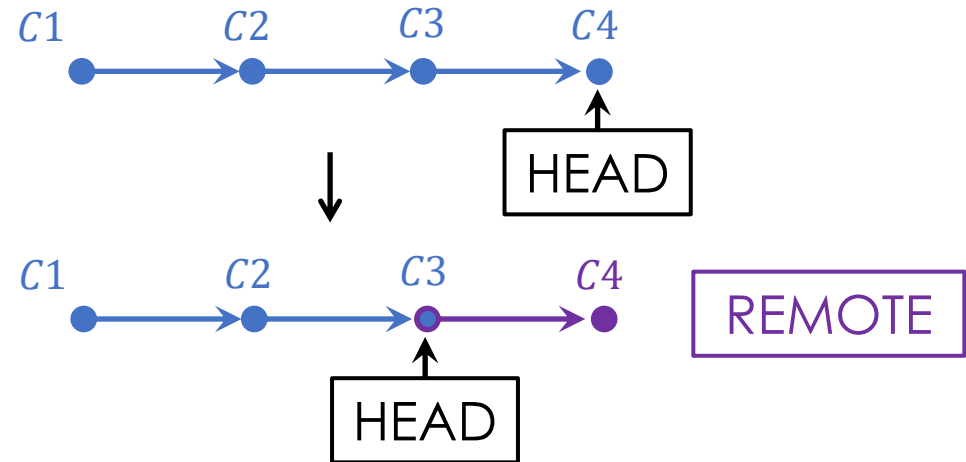
git
Undoing things (again)

Undo the last **local** commit, or the last 3 :

```
> git reset HEAD~1
> git reset HEAD~3
```

Undo the last **remote** commit, or the last 3 :

```
> git revert HEAD
> git revert HEAD~3..HEAD
```

$C1$ $C2$ $C3$ $C4$

HEAD

↓

$C1$ $C2$ $C3$ $C4$ REMOTE

HEAD

$C1$ $C2$ $C3$ $C4$

HEAD

↓

$C1$ $C2$ $C3$ $C4$ $C5$

HEAD

$C5$ revert the changes made in $C4$, so we end up in the same state as we ere in $C3$.

git

git fetch and git pull

```
> git fetch <remote>
```

Pulls down all the data from the remote project. You will have references to all the remote branches, which **you can merge** with your local branch in or inspect at any time.

```
> git merge <branch>
```

+

```
> git pull <remote> <branch> --ff --ff-only --rebase
```

Automatically **fetch and then merge** the remote branch into your current local branch.
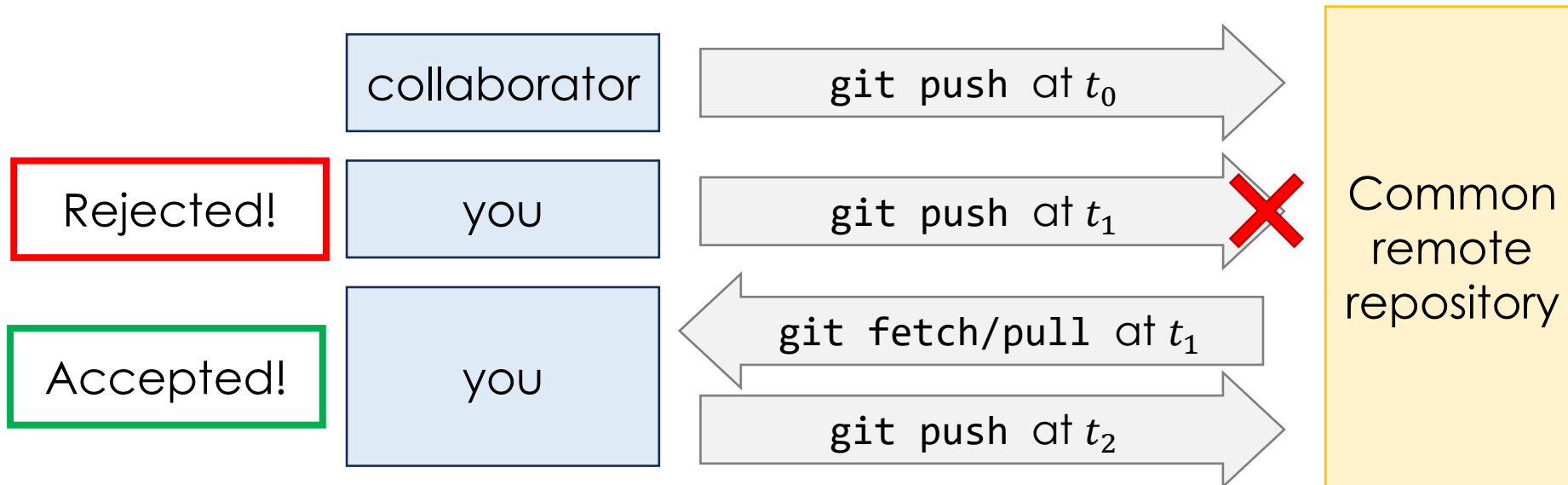
git

git push

Sends missing objects to the remote repository and updates the remote branch.
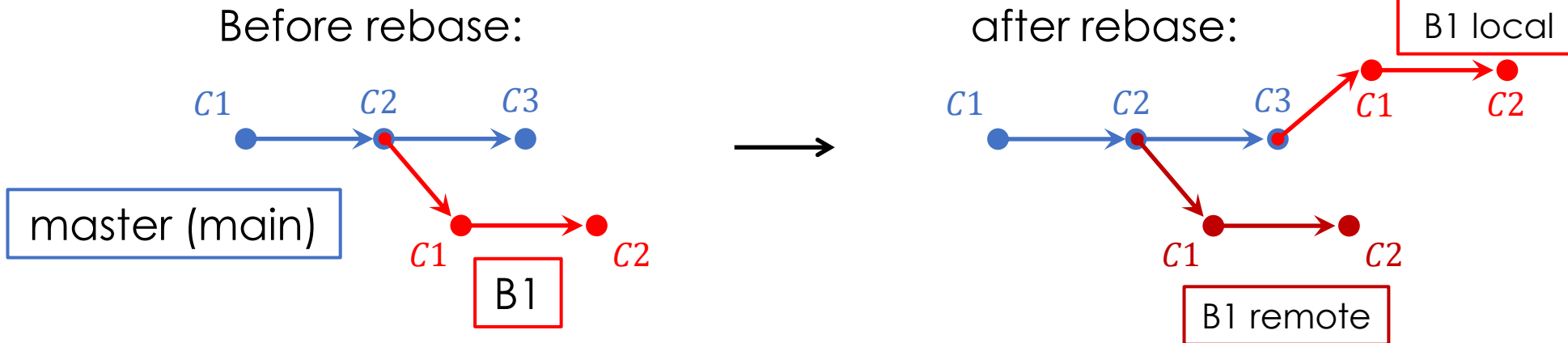
```
> git push <remote> <branch>
```

This command works only if you clone from a server to which you have **write access** and if nobody has pushed in the meantime.

	collaborator	git push at t_0	Common remote repository
Rejected!	you	git push at t_1 ✗	
Accepted!	you	git fetch/pull at t_1	
		git push at t_2	

git
Push after a rebase

```
> git checkout B1
> git rebase master
```

Before rebase:

after rebase:

master (main)

B1 local

B1 remote

B1

After the rebase B1 has become a **divergent branch**!
So, `git push` will be **rejected**! You will need a `push force`.

```
> git push origin B1 --force-with-lease --force
```

git
git init

Create a **local** repository by initializing a `.git/` directory inside `<local_dir>` (that is your local repository) :

```
> git init <local_dir>
```

```
> git add -A
> git commit -m "my first commit"
```

Create the **remote** repository on your favorite platform (e.g. GitHub, GitLab), **connect** it to your local repository, and push it:

```
> git remote add origin <repo_url>
> git push -u origin master --all
```

Local branches   → master, hpc
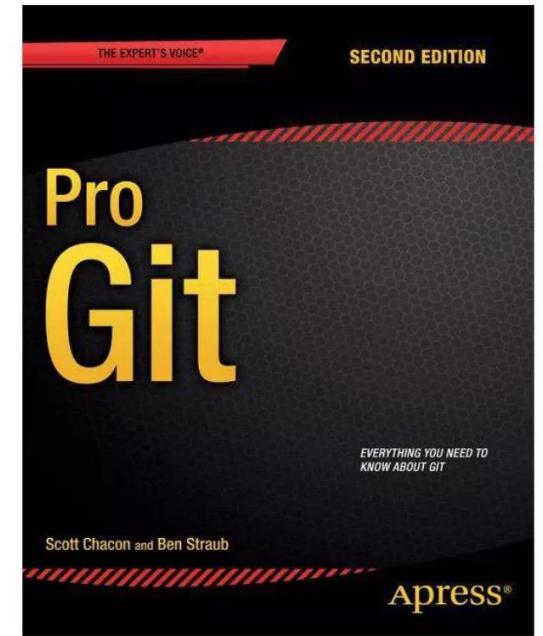Remote branches → origin/master, origin/hpc

git
References

Course repository: https://github.com/mattiamencagli/git_introduction

git book: https://git-scm.com/book/en/v2

GitHub documentation: https://docs.github.com/

GitLab documentation: https://docs.gitlab.com/

End of presentation

Mattia Mencagli
m.mencagli@cineca.it

Now the live session

git: EXTRA 2

SSH-key protocol

If you don't have one, create a ssh key:

```
> ssh-keygen
```

If you don't choose a custom name, directory or password, your private and public keys will be in the directory `${HOME}/.ssh/` as: `id_rsa` and `id_rsa.pub`

Add the content of your PUBLIC key in GitHub (https://github.com/settings/keys) or GitLab (https://gitlab.com/-/profile/keys), or your favorite platform.

Remember: NEVER share your PRIVATE key outside of your local machine.

git: EXTRA 3

Custom line prompt

Inside `${HOME}/.bashrc`, following the example below, export the git variables and add in the `PS1` variable a string as "git_string":

```
export GIT_PS1_SHOWCOLORHINTS=1
export GIT_PS1_SHOWDIRTYSTATE=1
export GIT_PS1_SHOWUNTRACKEDFILES=1
export GIT_PS1_SHOWUPSTREAM=1

user='\[\e[1;31m\]\u'
host='\[\e[0;35m\]@\h'
work_dir='\[\e[1;36m\]\w'
dollar='\[\e[1;32m\]\$'
input='\[\e[0;37m\] '

git_string='\[\e[1;93m\]$(__git_ps1 "(%s)")'

PS1=${user}${host}${work_dir}${git_string}${dollar}${input}
```

You can choose different colors changing the piece of string in the front: "`\[\e[1;93m\]`".
More informations about `PS1` and its colors here: link1, link2.

```
mmencagl@NMMENCAGL206897~/programming/gpluto_cpp(ot11 *%=)$ echo "hello world"
hello world
```

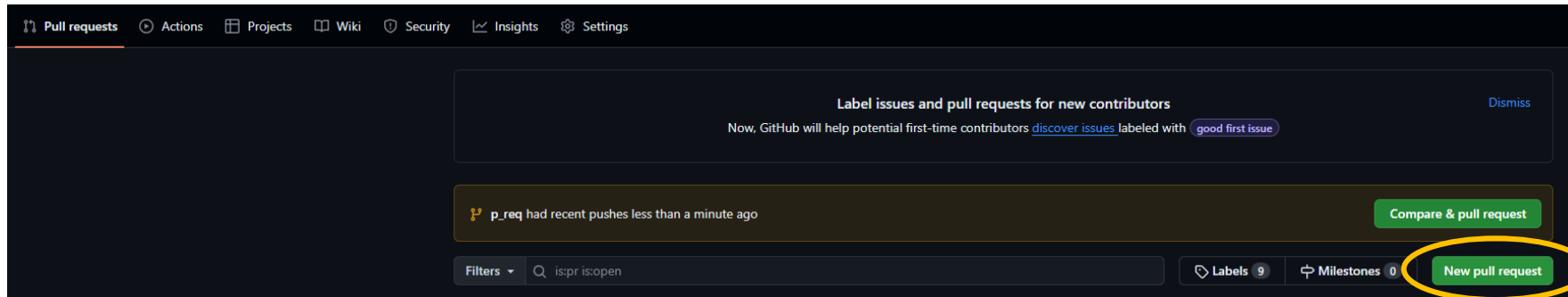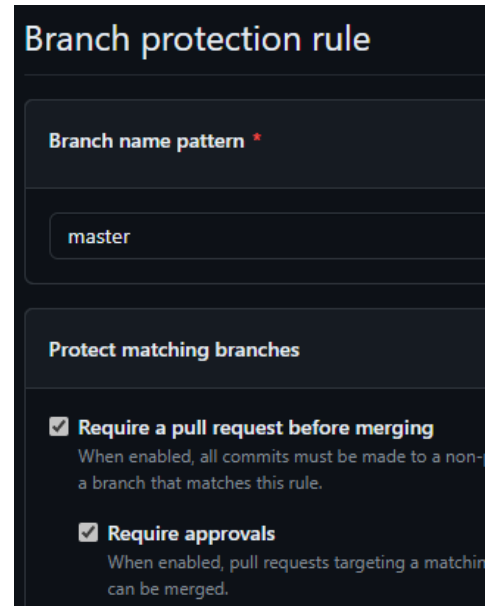| current branch | modified files | untracked files | upstream state |

git: EXTRA 4

Pull requests (GitHub example)

If the `master` is protected ([here](#) you can set your master as protected), you won't be able to normally push to it, you will have to push other branches and then ask for a "`pull request`" (that will merge your branch onto the `master`).
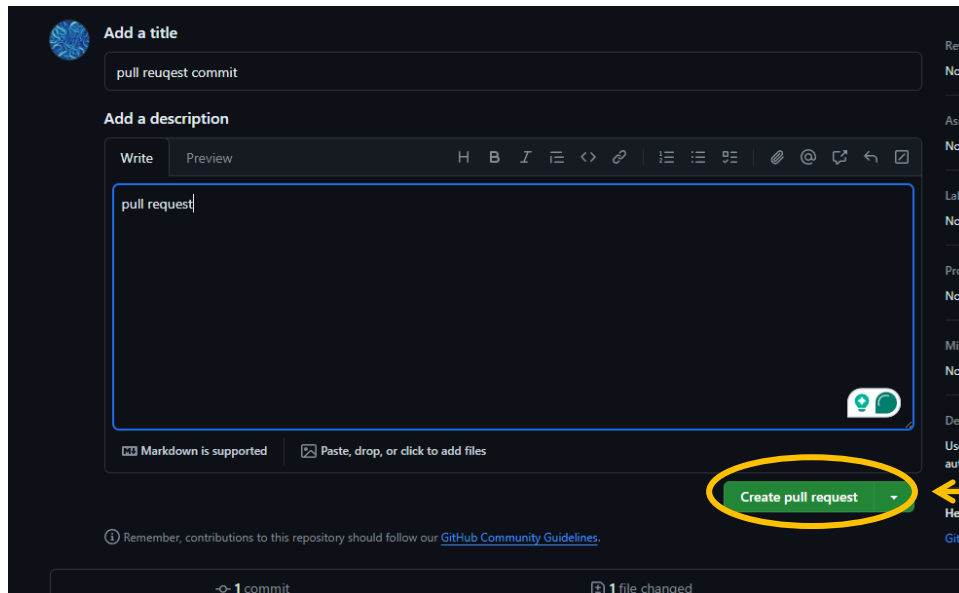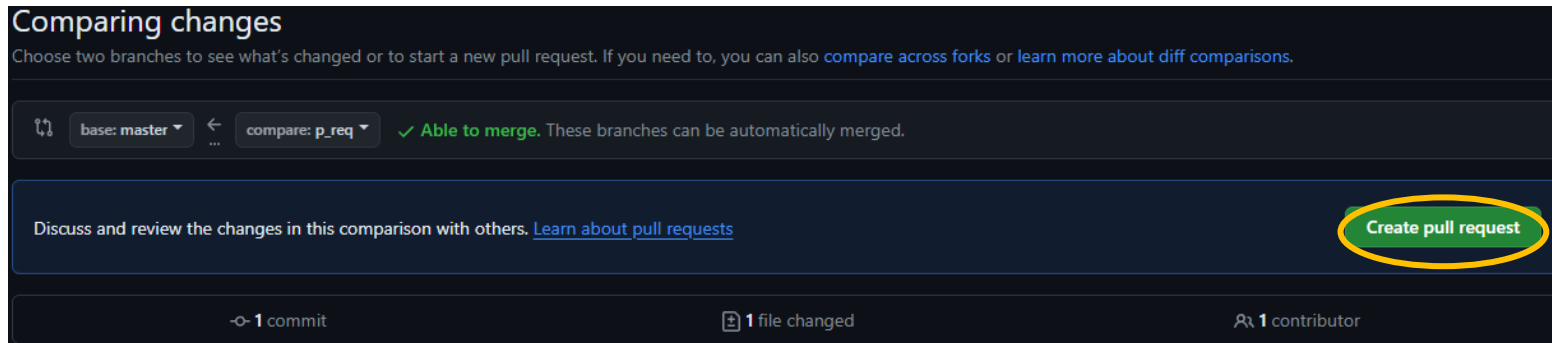
Create a `pull request` from [here](#).

Choose the branch that you want to merge onto the `master`.

git: EXTRA 4

Pull requests (GitHub example)

Add a description of your merge and create the `pull request`.

End

Mattia Mencagli
m.mencagli@cineca.it

Online seminar about **Continuous Integration (CI)** with **GitLab** on Wednesday the 24th of April from 09:30 to 11:30 CEST, by Geray Karademir & Kristian Kadlubiak (IT4i).

hpc-portal.eu link.