CINECA

Git

version control

Mattia Mencagli
m.mencagli@cineca.it

22/04/2024

git

Distributed version control system created by Linus Torvalds.

Slides: https://github.com/mattiamencagli/git_introduction

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.

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.

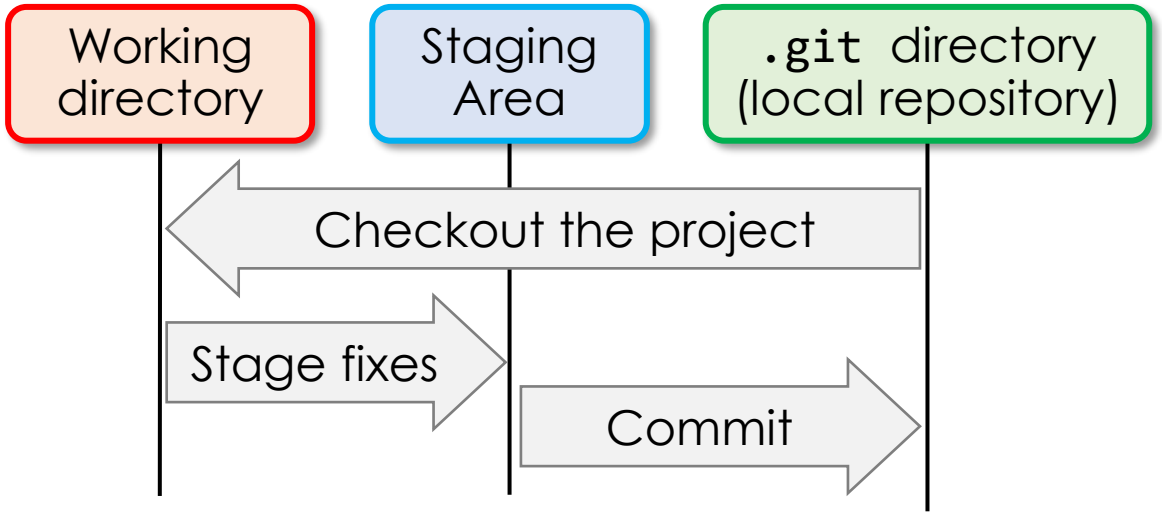
That creates a directory named `<local_dir>`,
initializes a `.git/` directory inside it (that is your local repository),
and pulls down all the data for that repository.

`<remote_url>`:
- HTTP protocol: https://github.com/mattiamencagli/git_introduction.git
- SSH protocol: git@github.com:mattiamencagli/git_introduction.git

git

git states

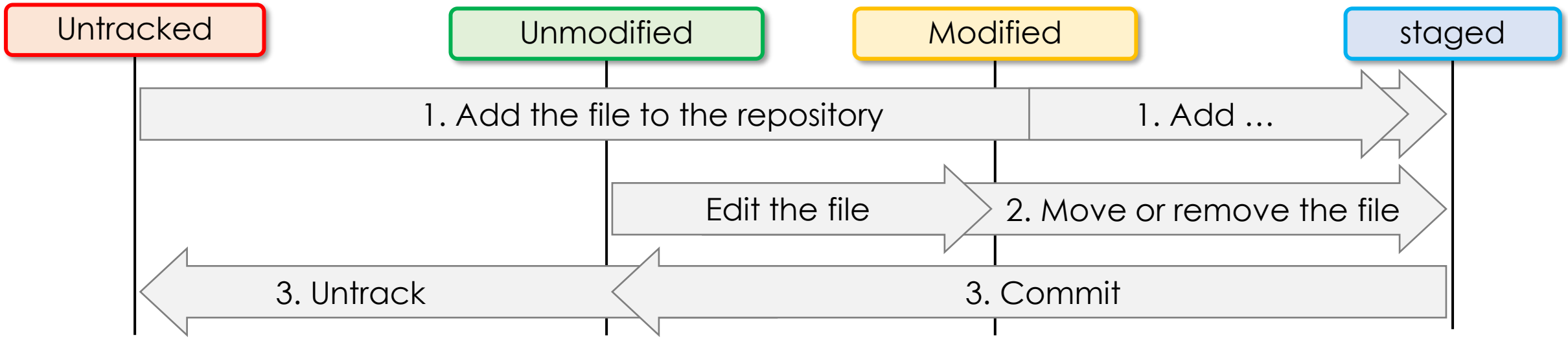
- **Modified** means that you have changed the file but have **not committed** yet it to your local repository yet.
- **Staged** means that you have marked a modified file in its current version to go into your next **commit snapshot**.
- **Committed** means that the data is safely stored in your local repository.



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

git

Files life-cycle



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 thing

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

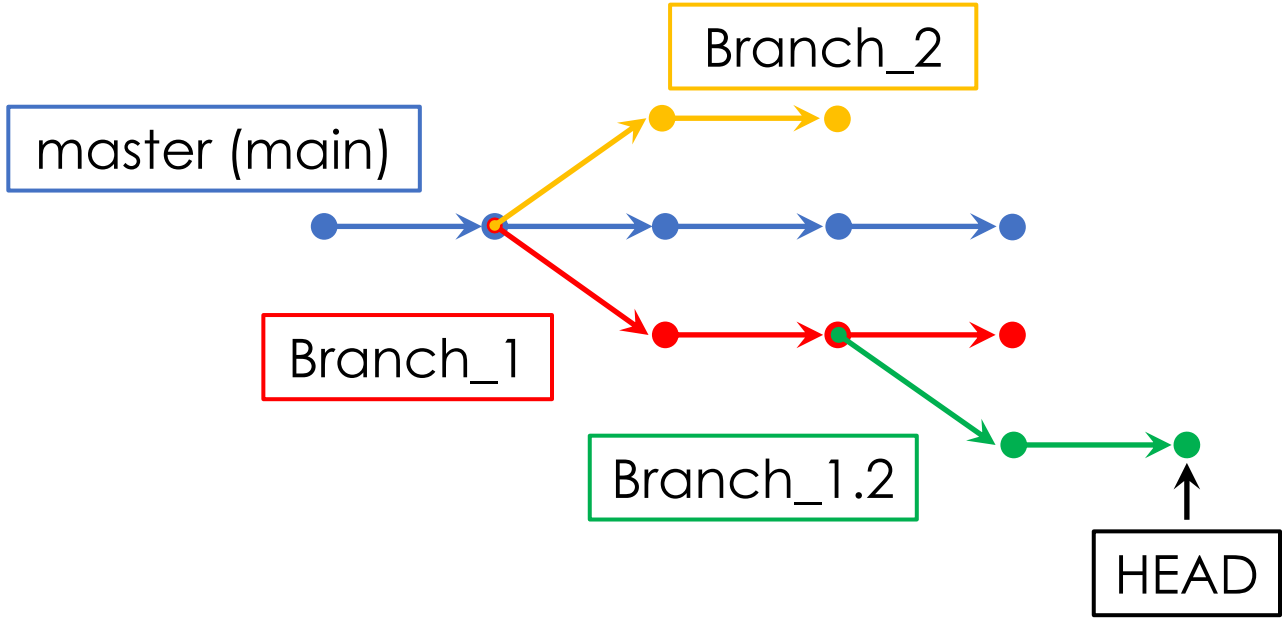
A branch in git is a movable pointer to one of the commits.

Create a new branch:

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

Switch to an existing branch:

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



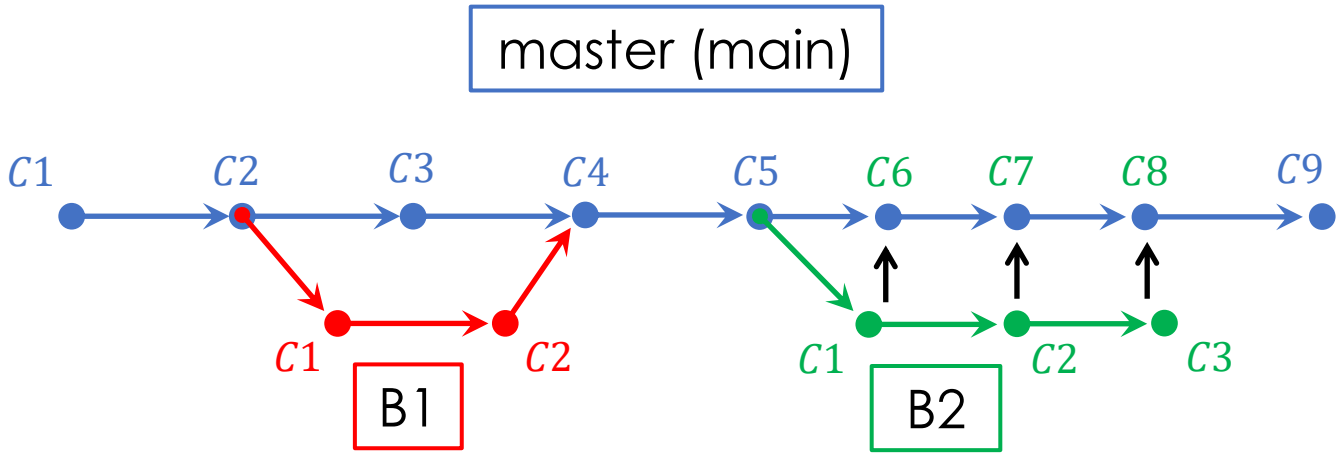
HEAD is a particular pointer. It always points to the local branch you're currently on.

git

Merge branches

Merge two branches:

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



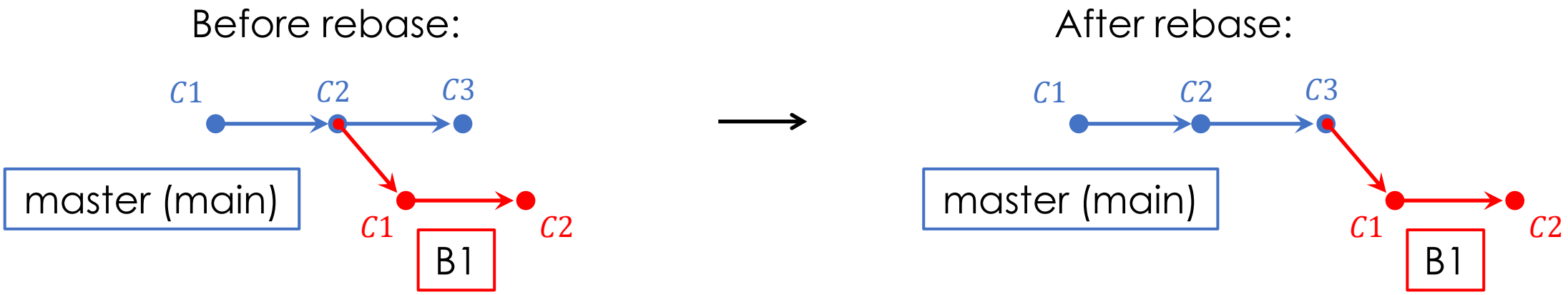
- **Merge commit** (B1): git creates a new commit (snapshot) that results from this three-way merge.
  [C4 is a new commit]
- **Fast-forward merge** (B2): is a special case of merge that happens if there is not a divergent history: git just moves the pointer forward.
  [the master points to the commits of the branch B2]

git

Rebase branches

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

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



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 or **share** your work.

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 thing (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
```

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>
```

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

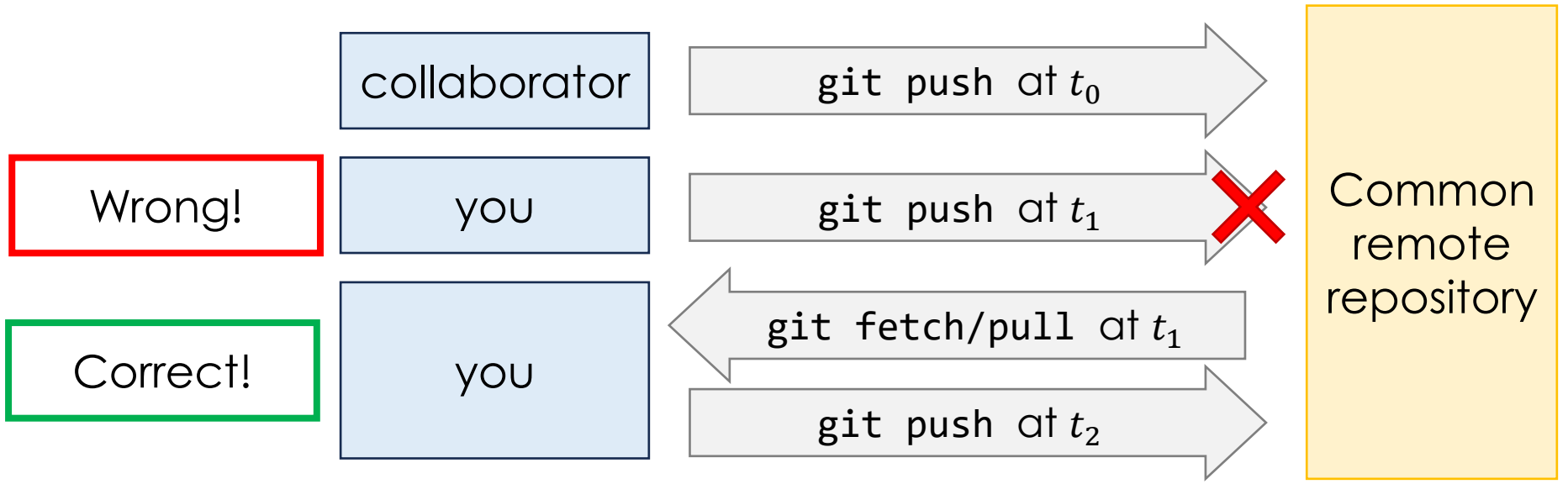
git

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.



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
> git push –u origin --all
```

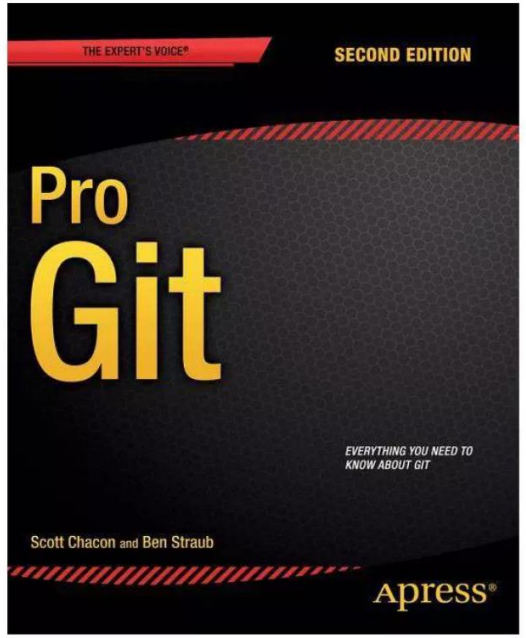
Local branches → `master`, `hpc`
Remote branches → `origin/master`, `origin/hpc`

git

References

Slides: 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

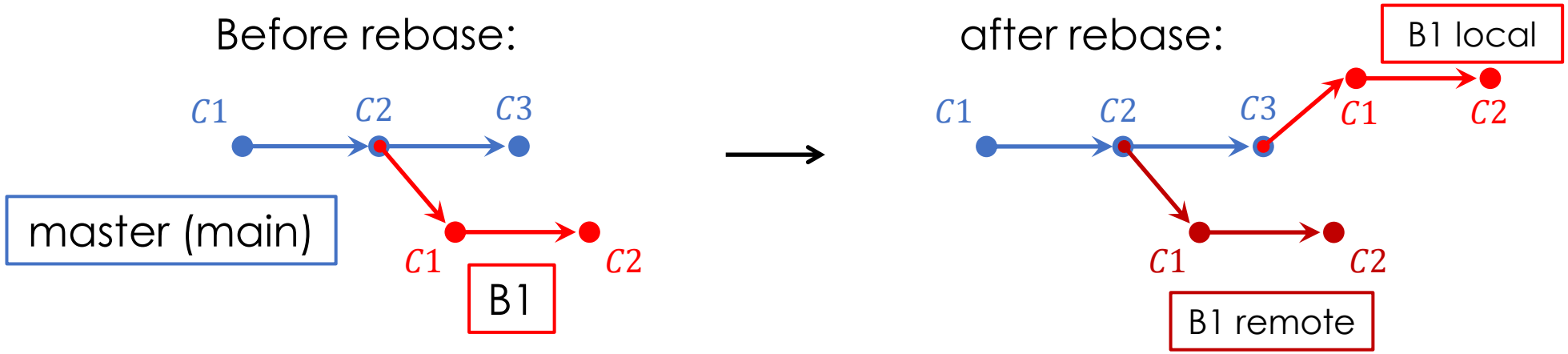
Mattia Mencagli
m.mencagli@cineca.it

Now the live session

git: EXTRA 1

Push rebase example

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



After the rebase B1 has become a **divergent branch**! So, `git push` will not work anymore, you will need a `push force.`

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

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.