GIT in 2 hours

Dott. Ing. **Marco Rabozzi**
Politecnico di Milano
Email: **marco.rabozzi@polimi.it**

Credits: Jeffrey Leek - Johns Bloomberg School of Public Health; http://git-scm.com/book/en/v2/

Outline

1) Introduction and setup

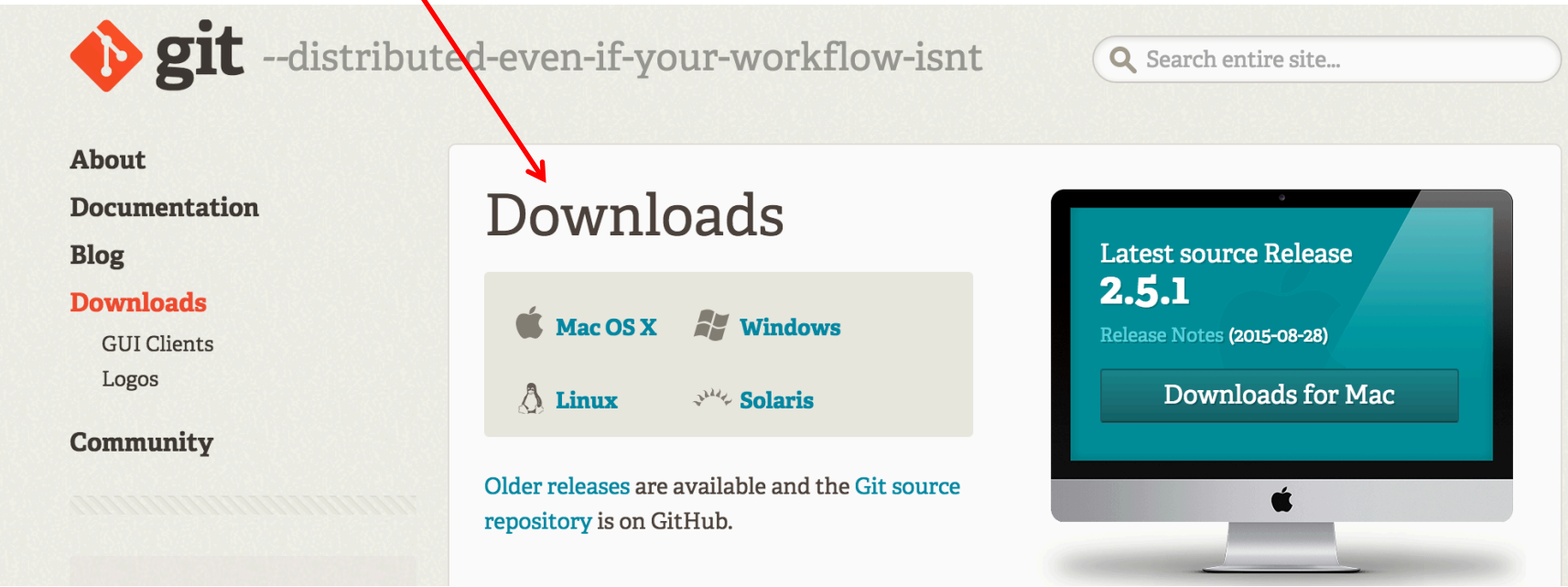
2) Using GIT locally

3) Work with a remote repository

1. INTRODUCTION AND SETUP

First of all…

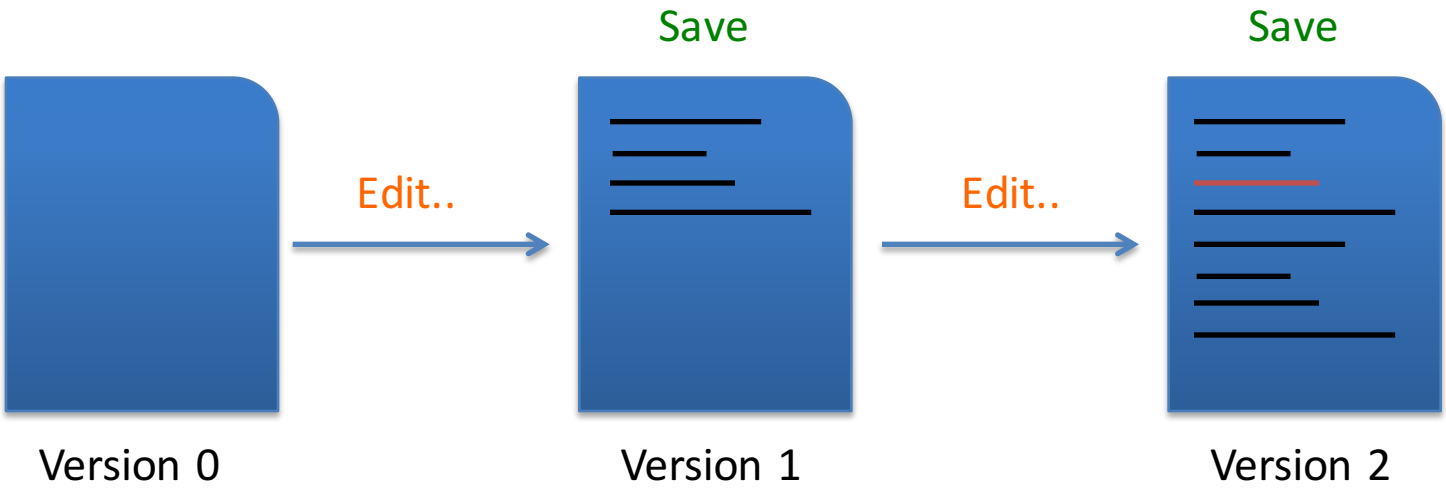
- Download git from: http://git-scm.com/downloads



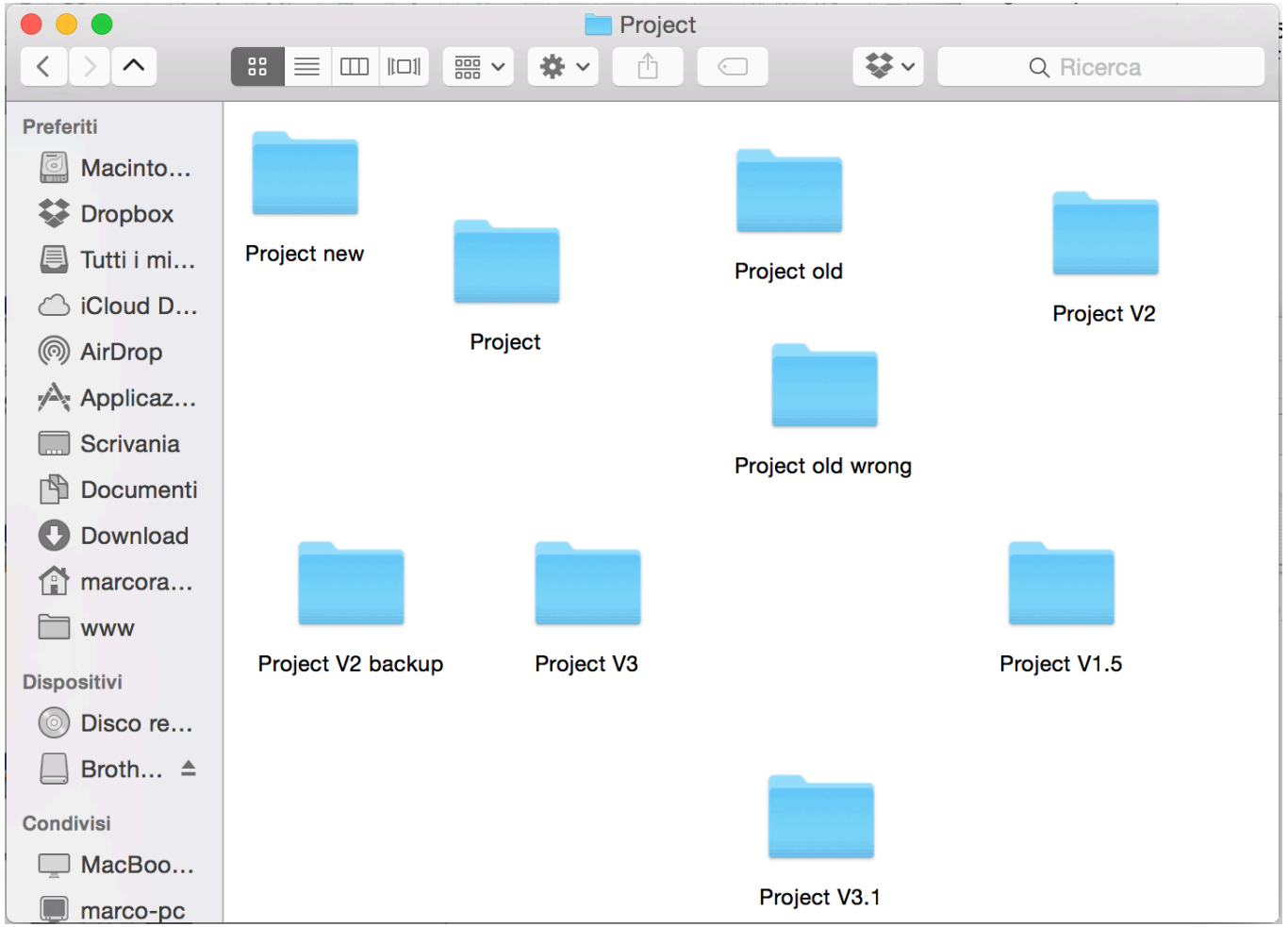
- These slides are available at: https://github.com/marcorabo/brief-git-course.git Under “short-version” folder

What is a Version Control System?

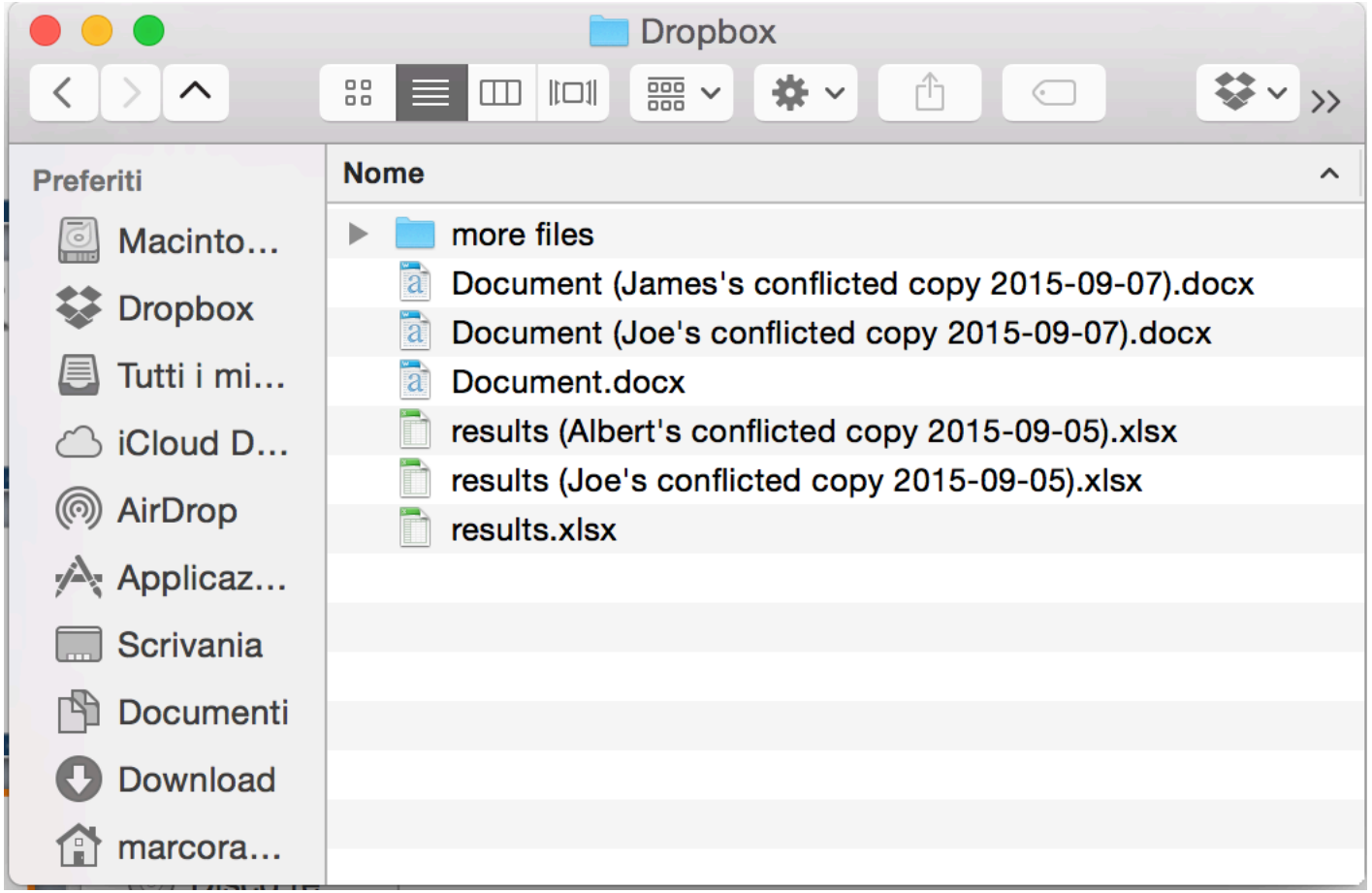
- A Version Control System (VCS) is a software that **records changes** to a set of **files** over time so that you can recall specific versions later



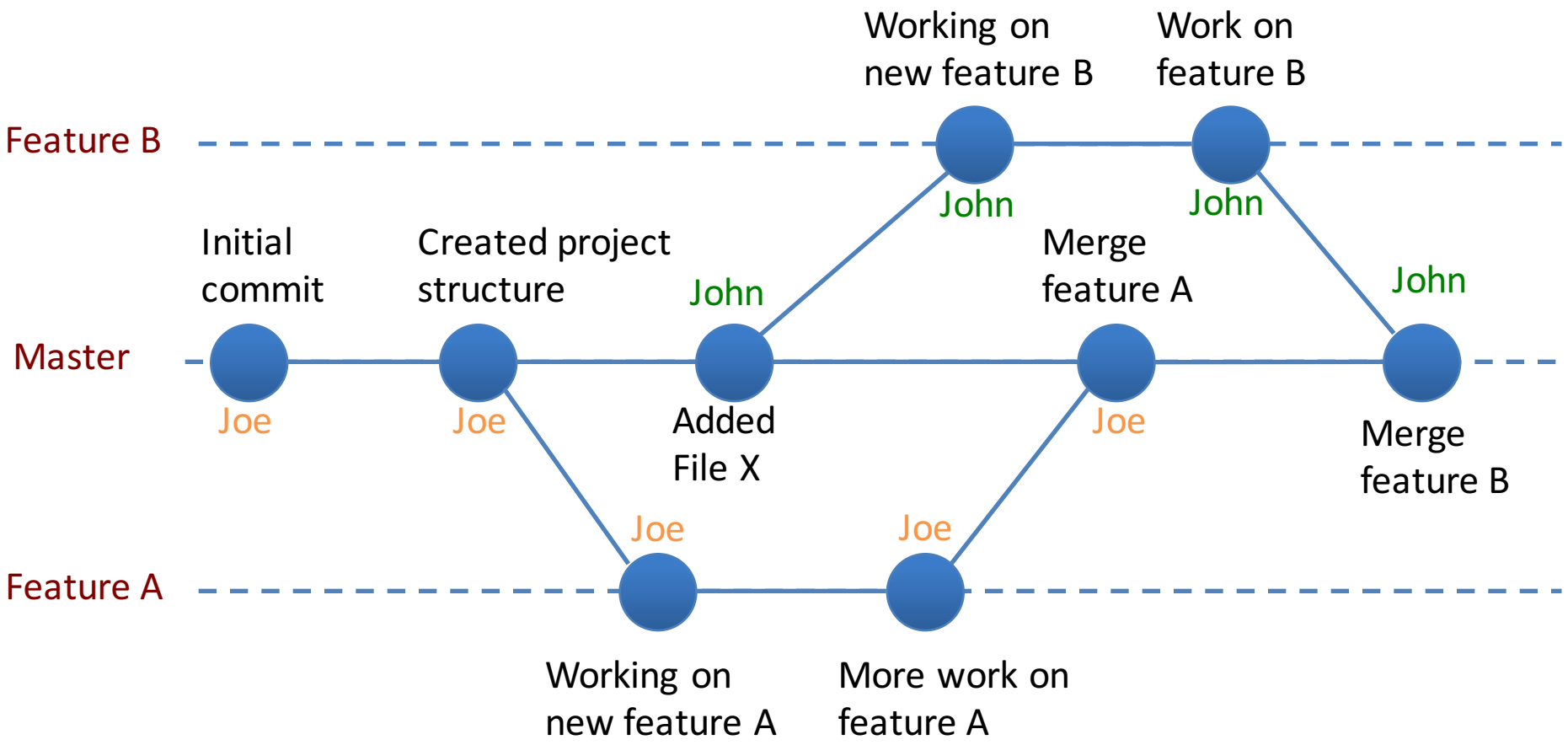
Why you need a VCS?



Why you need a VCS?



Version Control System



Version Control System

- Allows multiple users to collaborate on a project
- Keeps track of changes made by different users over time
- Example of VCS:
  - GIT
  - SVN
  - Mercurial

What is GIT?

http://git-scm.com

"Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency"

When to use GIT

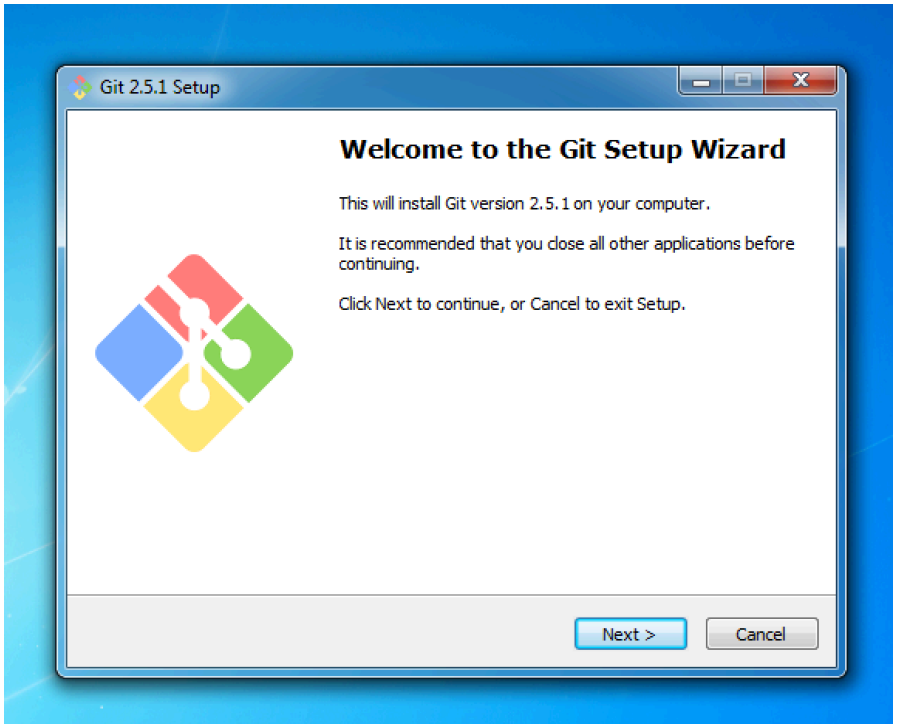
- GIT is powerful at versioning text files:
  - C
  - C++
  - MATLAB
  - Latex
  - HTML
  - …
- Usage scenarios:
  - **Single user**: keep track of the file changes for your own project
  - **Multiple users**: allow a team to share its work for a project
  - **Open source**: allow the world to propose fixes and new features to a project

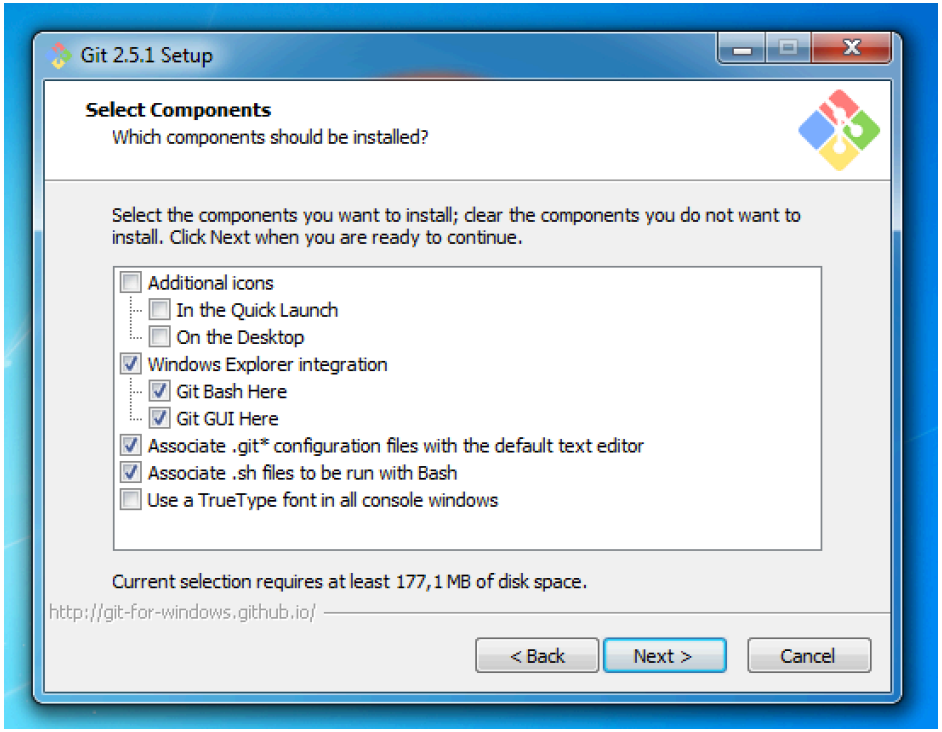
Getting Started!

- Download git from: http://git-scm.com/downloads

Setup GIT

Leave all the parameters to their default values



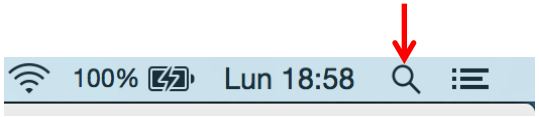


(Note: Screenshots for Windows users)

Verify your setup

- Open the Command Line Interface (CLI)
  - Windows: search “Git Bash” from the Start Menu

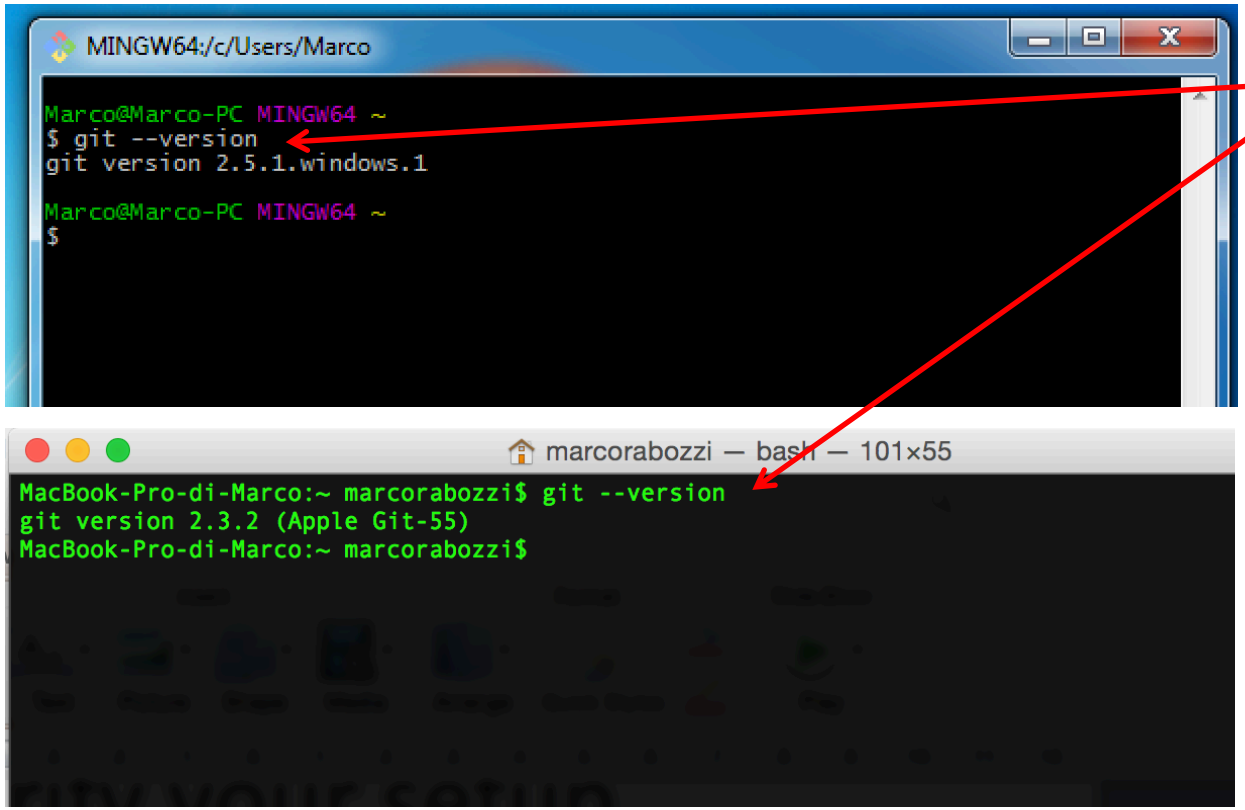
  - Mac OS X: search “terminal” from spotlight search



  - Linux (Ubuntu): search “terminal” from dash

Verify your setup

- Check from the CLI that GIT is installed



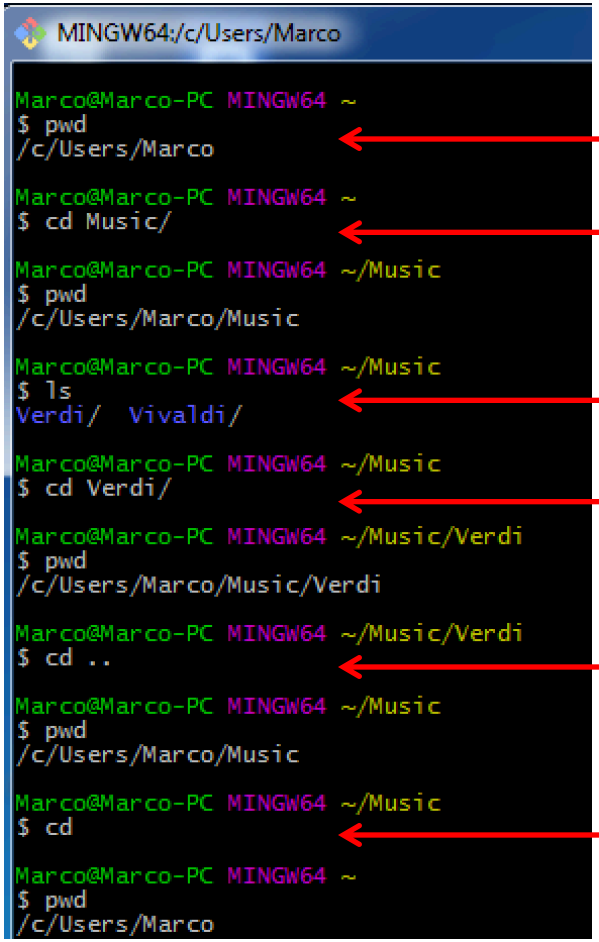
Run the command:
`git --version`

The installed version of GIT should be printed

Command line interface (CLI)

- Useful commands
  - `pwd`
  - `cd`
  - `ls`
  - `touch`
  - `mkdir`
  - `cp`
  - `rm`
  - `mv`
- Useful tools:
  - `vim`

CLI – navigate folders



Print current directory

Move to folder “Music”

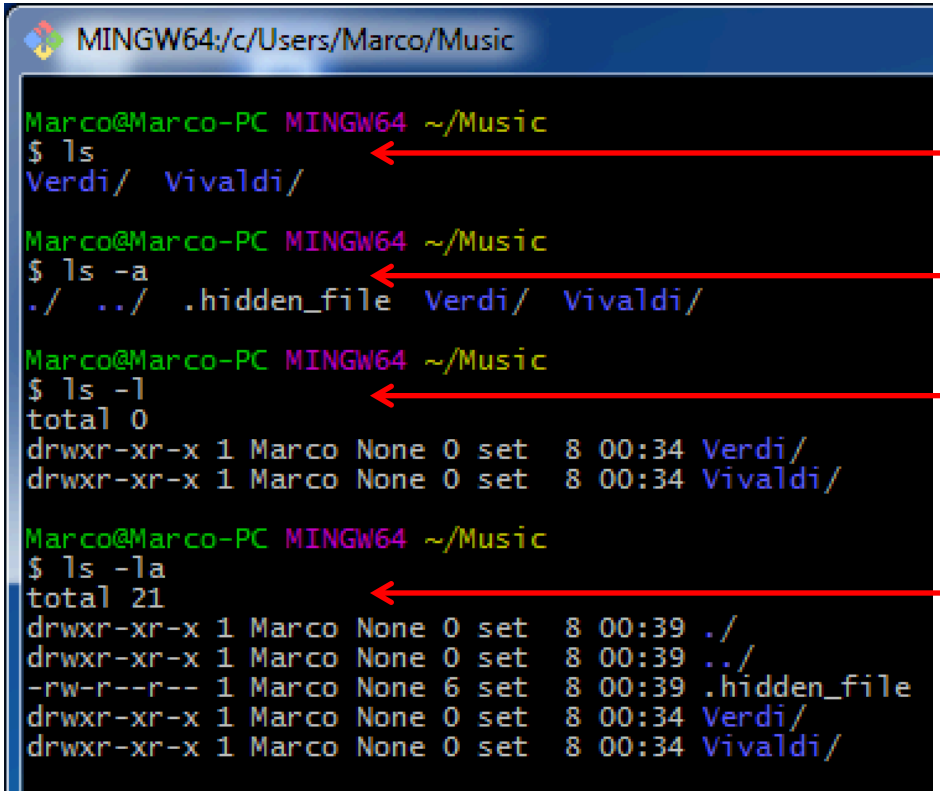
List files/folder in current directory

Move to folder “Verdi”

Go UP one level (Move to parent folder)

Go to home directory

CLI – list files



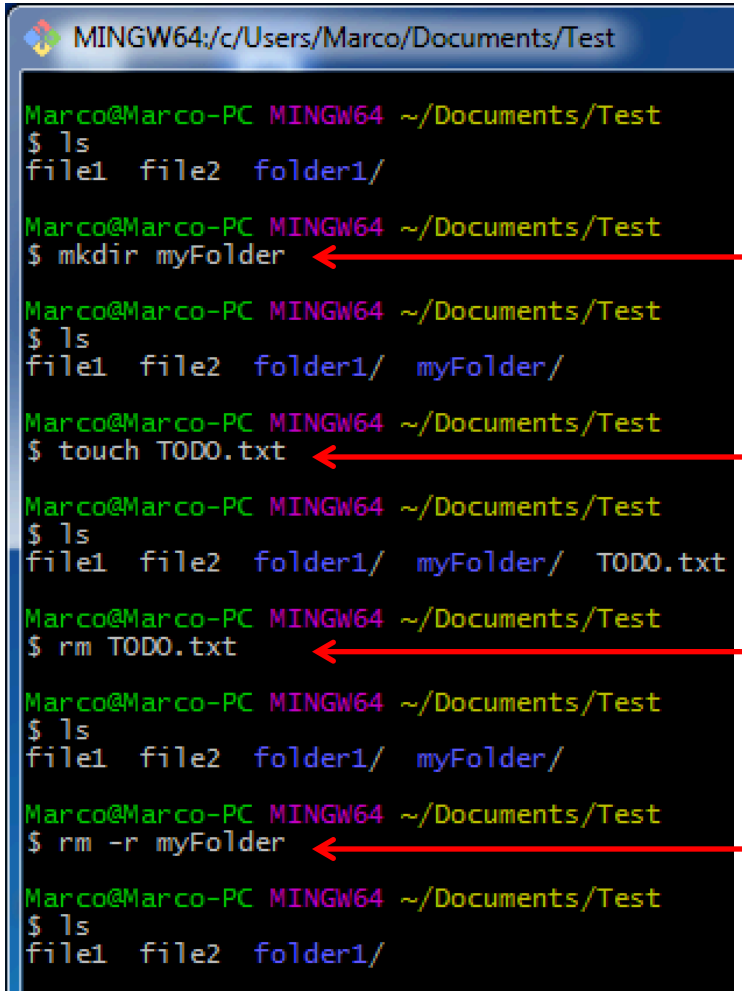
List files/folders in current directory

List all files/folders including hidden files

List files/folders providing more details

List all files/folder with details including hidden files
NOTE: `ls -la` == `ls -al`

CLI – create/delete files and folders



Create folder "myFolder" in current directory

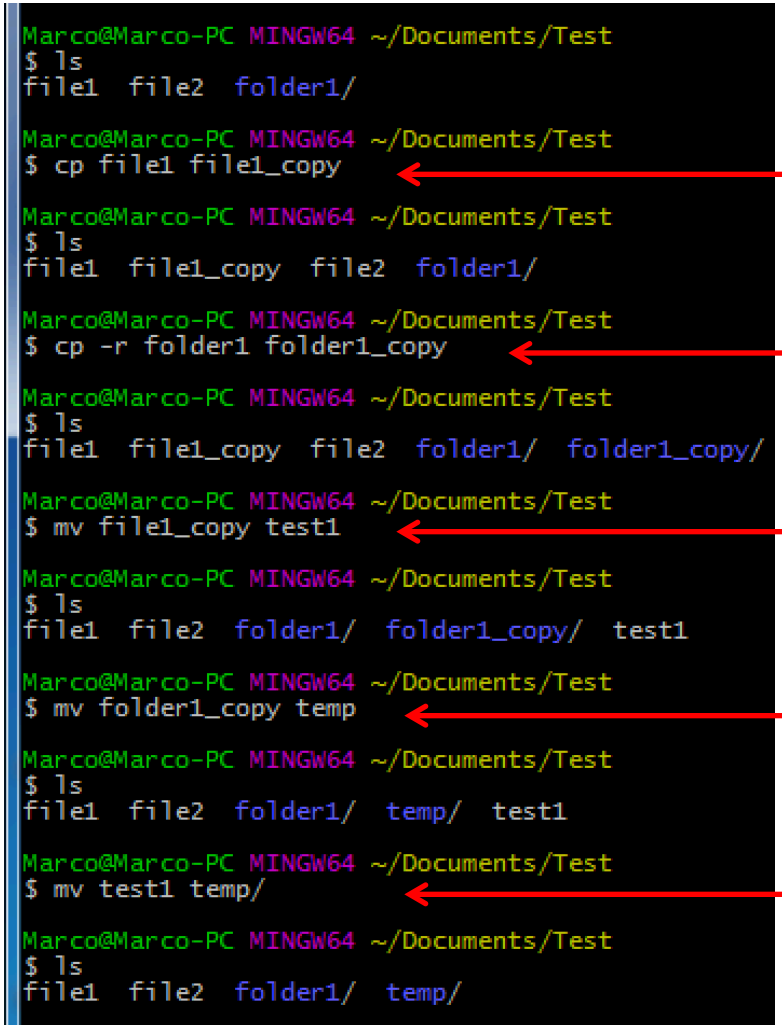
Create file "TODO.txt" in current directory

Remove file "TODO.txt"

Remove folder "myFolder"

BE CAREFUL There is no undo for `rm`!

CLI – copy/move files and folders



Make a copy of “file1” called “file1_copy”

Make a copy of “folder1” called “folder1_copy”

Rename “file1_copy” to “test1”

Rename “folder1_copy” to “temp”

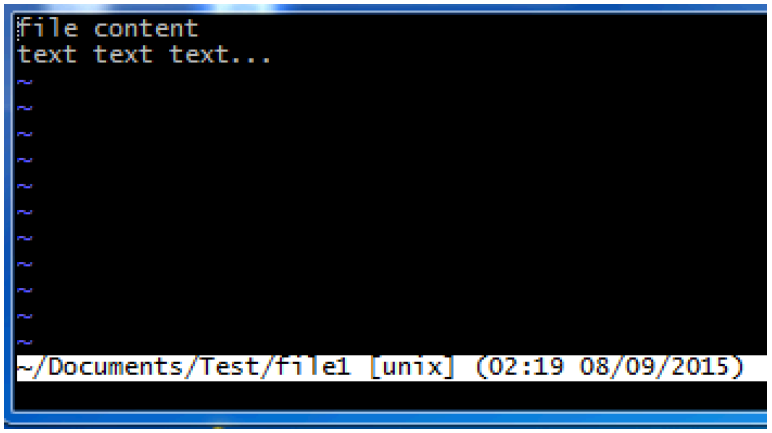
Move file “test1” in folder “temp”

VIM text editor

```
Marco@Marco-PC MINGW64 ~/Documents/Test
$ ls
file1  file2  folder1/

Marco@Marco-PC MINGW64 ~/Documents/Test
$ vim file1
```

`vim <file>`
(enter vim)

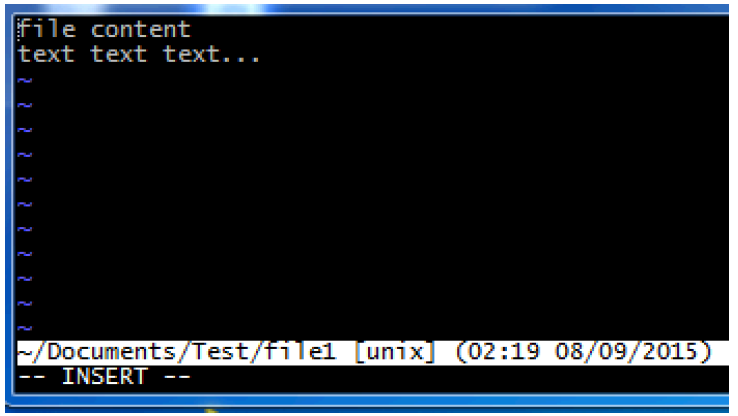
"`:wq`" (quit and save changes)
"`:q!`" (quit without saving)



Normal mode

"`i`"

ESC



Insert mode

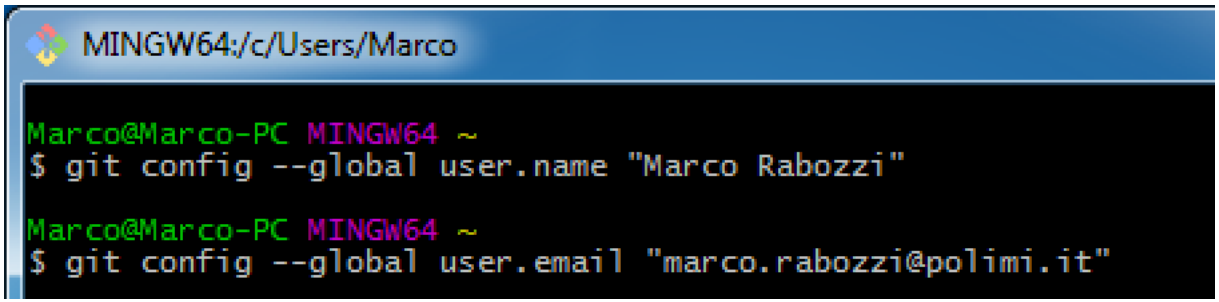
CLI - recap

- Useful commands
  - `pwd` Print current working directory
  - `cd` Change directory
  - `ls` List files and folders
  - `touch` Create a file
  - `mkdir` Create a folder
  - `cp` Copy file/folder
  - `rm` Remove file/folder
  - `mv` Move file/folder (e.g.: cat and paste)
- Useful tools:
  - `vim` Open a text editor

GIT configuration

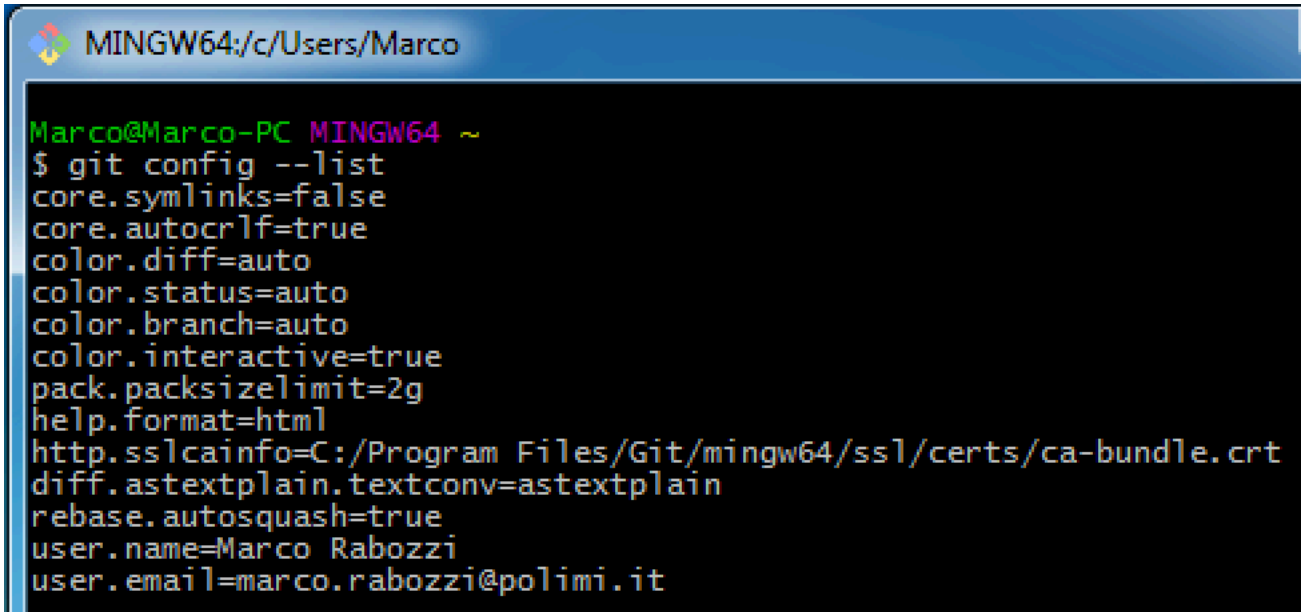
- Each commit to a Git repository will be “tagged” with the name of the person who made the commit

- The following commands set your name and email:



GIT configuration

- Run the command `git config --list` to see your current configuration:



2. USING GIT LOCALLY

Create a project

- In order to start using GIT, we first need a folder to store our project:

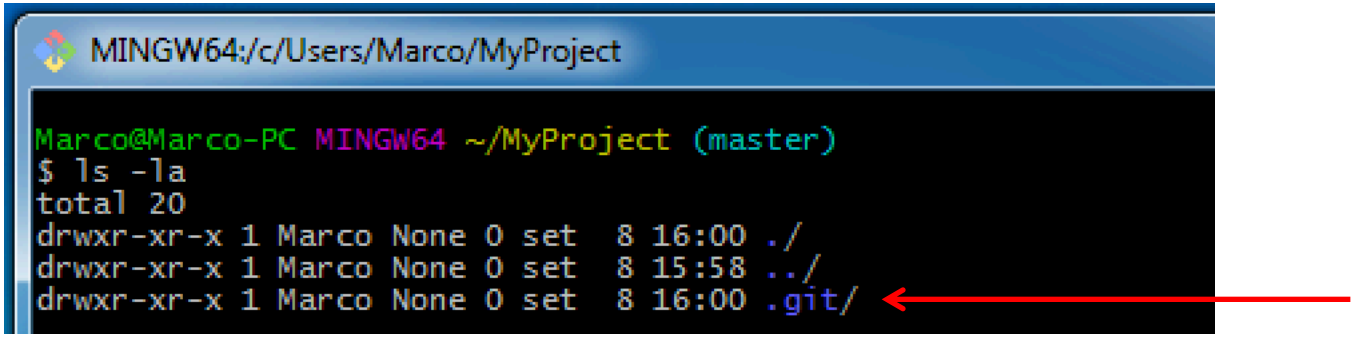
```
Marco@Marco-PC MINGW64 ~
$ mkdir MyProject
```

- To initialize a GIT ***repository*** for your project, enter the project folder and run `git init`

```
Marco@Marco-PC MINGW64 ~
$ cd MyProject/

Marco@Marco-PC MINGW64 ~/MyProject
$ git init
Initialized empty Git repository in C:/Users/Marco/MyProject/.git/

Marco@Marco-PC MINGW64 ~/MyProject (master)
$
```

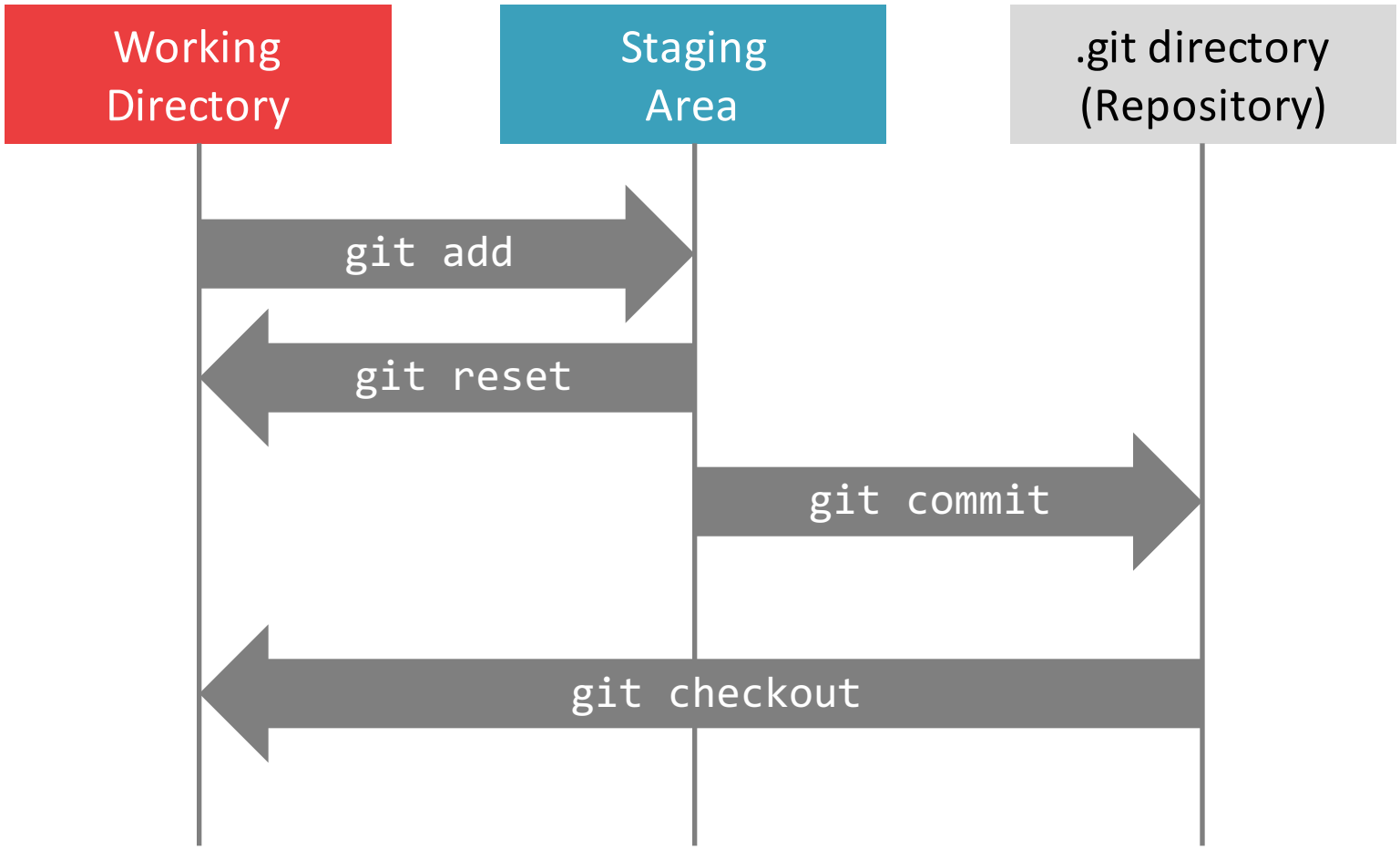
Create a project

- The repository is contained within the hidden folder ".git" in your project directory:



- This folder is used by GIT to manage your repo, it should not be modified manually

How GIT works (locally)



Towards the first commit

- Run: `git status`
  to show the current status of your repo
  (i.e.: what changed in your project since the last commit)

```
Marco@Marco-PC MINGW64 ~/MyProject (master)
$ git status
On branch master

Initial commit

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

Nothing has changed

Towards the first commit

- Create a file in your project, you can create/edit the file with the editor of your choice (VIM, notepad, sublime, …)

```
Marco@Marco-PC MINGW64 ~/MyProject (master)
$ touch contributors.txt

Marco@Marco-PC MINGW64 ~/MyProject (master)
$ ls
contributors.txt
```

- Check the status of your repo:

```
Marco@Marco-PC MINGW64 ~/MyProject (master)
$ git status
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        contributors.txt

nothing added to commit but untracked files present (use "git add" to track)
```

Adding files

- Use the command `git add <file>` to add the desired files to the staging area

```
Marco@Marco-PC MINGW64 ~/MyProject (master)
$ git add contributors.txt
```

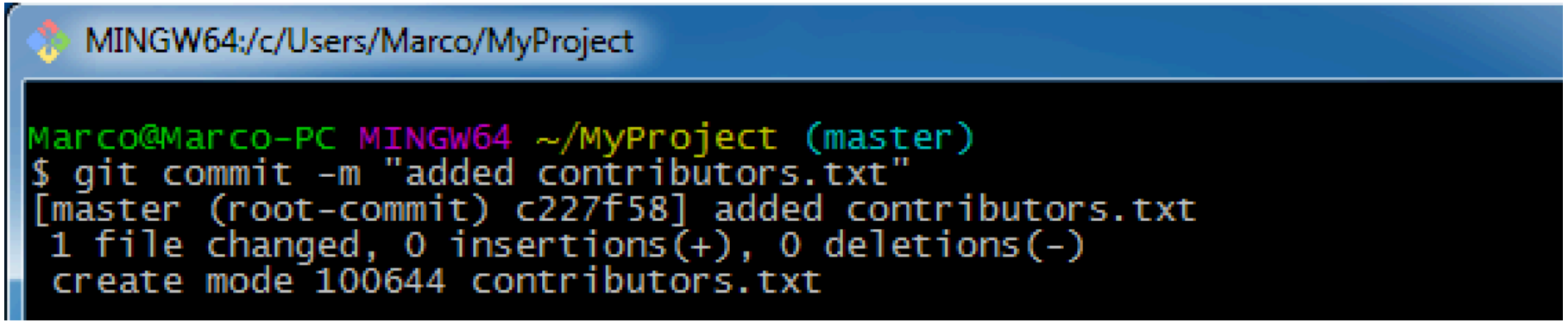
- Check the status of your repo:

```
Marco@Marco-PC MINGW64 ~/MyProject (master)
$ git status
On branch master

Initial commit

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

        new file:   contributors.txt
```

- NOTE: `git add` can be undone using `git reset`

The first commit

- The file “contributors.txt” is now in the staging area but has not been committed yet to the repo
- To commit the files in the staging area use:
  - `git commit -m “your commit message”`



Verify the status of your repo

- `git status`

```
Marco@Marco-PC MINGW64 ~/MyProject (master)
$ git status
On branch master
nothing to commit, working directory clean
```

- `git log` (shows the commit history of your repo)

```
Marco@Marco-PC MINGW64 ~/MyProject (master)
$ git log
commit c227f58eda5e1e2be19405b5ccb0ddcef0ca2e0e
Author: Marco Rabozzi <marco.rabozzi@polimi.it>
Date:   Tue Sep 8 17:15:12 2015 +0200

    added contributors.txt
```

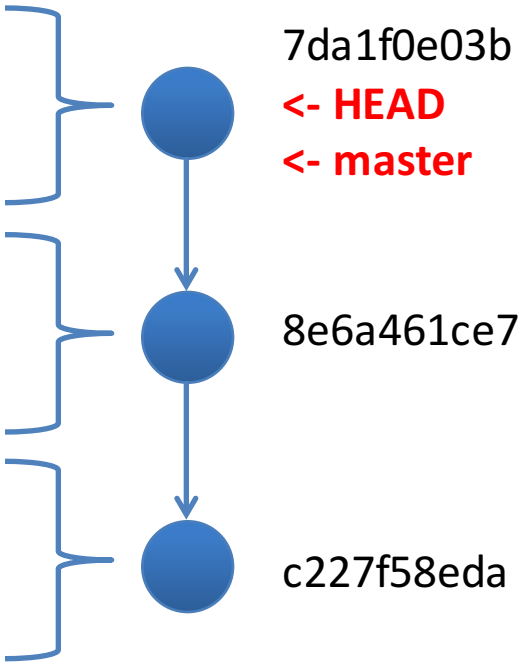
Commit ID

Message assigned to the commit

GIT history

– Assume that after a few commits `git log --graph` returns:

```
Marco@Marco-PC MINGW64 ~/MyProject (master)
$ git log --graph
* commit 7da1f0e03bf68794db6f4716d7acc6c2c843fe8d
| Author: Marco Rabozzi <marco.rabozzi@polimi.it>
| Date:   Tue Sep 8 19:14:34 2015 +0200
|
|     Added more contributors
|
* commit 8e6a461ce749bcfad38e2c95c43000ad5dcd4c56
| Author: Marco Rabozzi <marco.rabozzi@polimi.it>
| Date:   Tue Sep 8 19:13:21 2015 +0200
|
|     added Marco Rabozzi to contributors
|
* commit c227f58eda5e1e2be19405b5ccb0ddcef0ca2e0e
  Author: Marco Rabozzi <marco.rabozzi@polimi.it>
  Date:   Tue Sep 8 17:15:12 2015 +0200

      added contributors.txt
```



– Commits can be identified by the first 10 characters of their ID

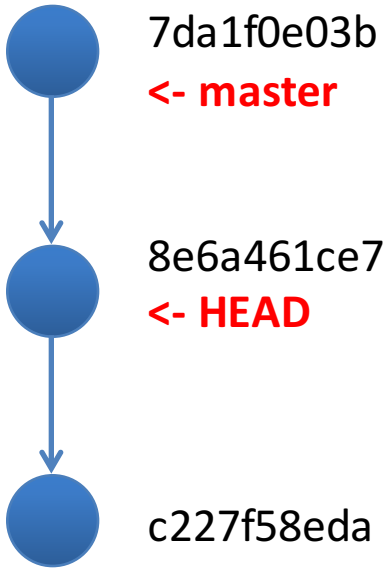
– **HEAD** is a pointer to the current commit

– **master** is a pointer to the last commit of the branch “master”

Checkout a previous commit

- If your working directory is clean you can revert using `git checkout <commit_id>`:

```
Marco@Marco-PC MINGW64 ~/MyProject (master)
$ git checkout 8e6a461ce7
Note: checking out '8e6a461ce7'.

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 performing another checkout.

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

  git checkout -b <new-branch-name>

HEAD is now at 8e6a461... added Marco Rabozzi to contributors
```

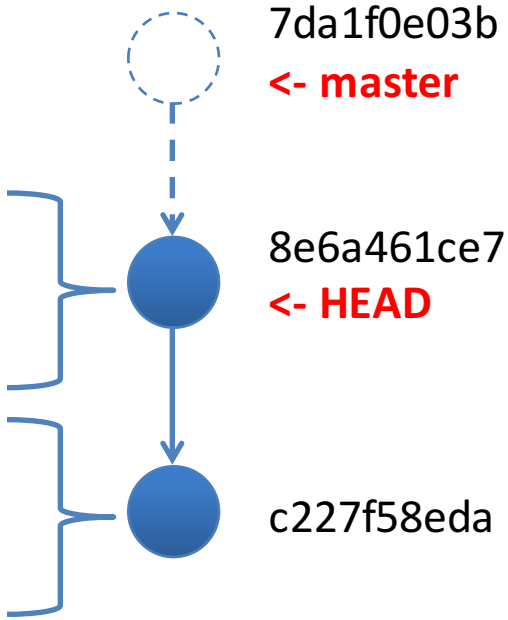


- The working directory will show the content of the files as they were at the specified commit

Checkout a previous commit

- Running `git log --graph` now shows:

```
Marco@Marco-PC MINGW64 ~/MyProject ((8e6a461...))
$ git log --graph
* commit 8e6a461ce749bcfad38e2c95c43000ad5dcd4c56
| Author: Marco Rabozzi <marco.rabozzi@polimi.it>
| Date:   Tue Sep 8 19:13:21 2015 +0200
|
|     added Marco Rabozzi to contributors
|
* commit c227f58eda5e1e2be19405b5ccb0ddcef0ca2e0e
  Author: Marco Rabozzi <marco.rabozzi@polimi.it>
  Date:   Tue Sep 8 17:15:12 2015 +0200

      added contributors.txt
```

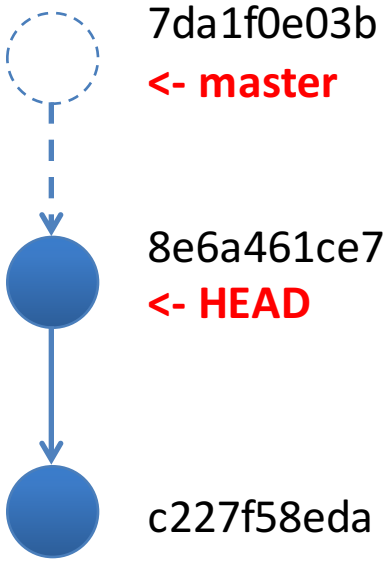


Checkout a previous commit

- Running `git status` shows:

```
MINGW64:/c/Users/Marco/MyProject

Marco@Marco-PC MINGW64 ~/MyProject ((8e6a461...))
$ git status
HEAD detached at 8e6a461
nothing to commit, working directory clean
```

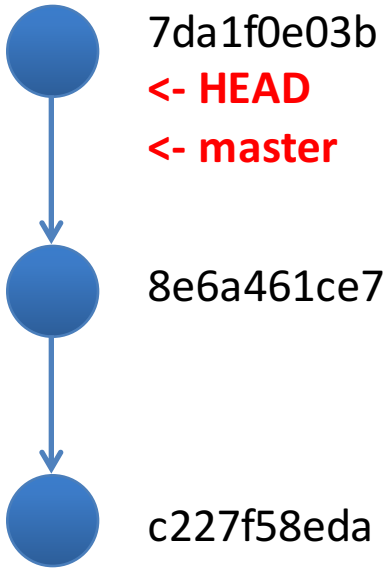
HEAD is not pointing to any referenced commit (e.g. “master”)



Return to the last commit

- To go to the last commit of a branch run:
  `git checkout <branch_name>`

```
Marco@Marco-PC MINGW64 ~/MyProject ((8e6a461...))
$ git checkout master
Previous HEAD position was 8e6a461... added Marco Rabozzi to
contributors
Switched to branch 'master'
```



How to check last changes

- To show what are the changes not committed yet run:
  - `git diff HEAD`

- To show what are the changes not committed yet for a specific file run:
  - `git diff HEAD -- <file>`

How to undo things

- To remove a file form the staging (i.e. undo a git add <file>) run:
  - `git reset <file>`

- If you want to discard the changes made to a file since the last commit (or since the last add) run:
  - `git checkout -- <file>`

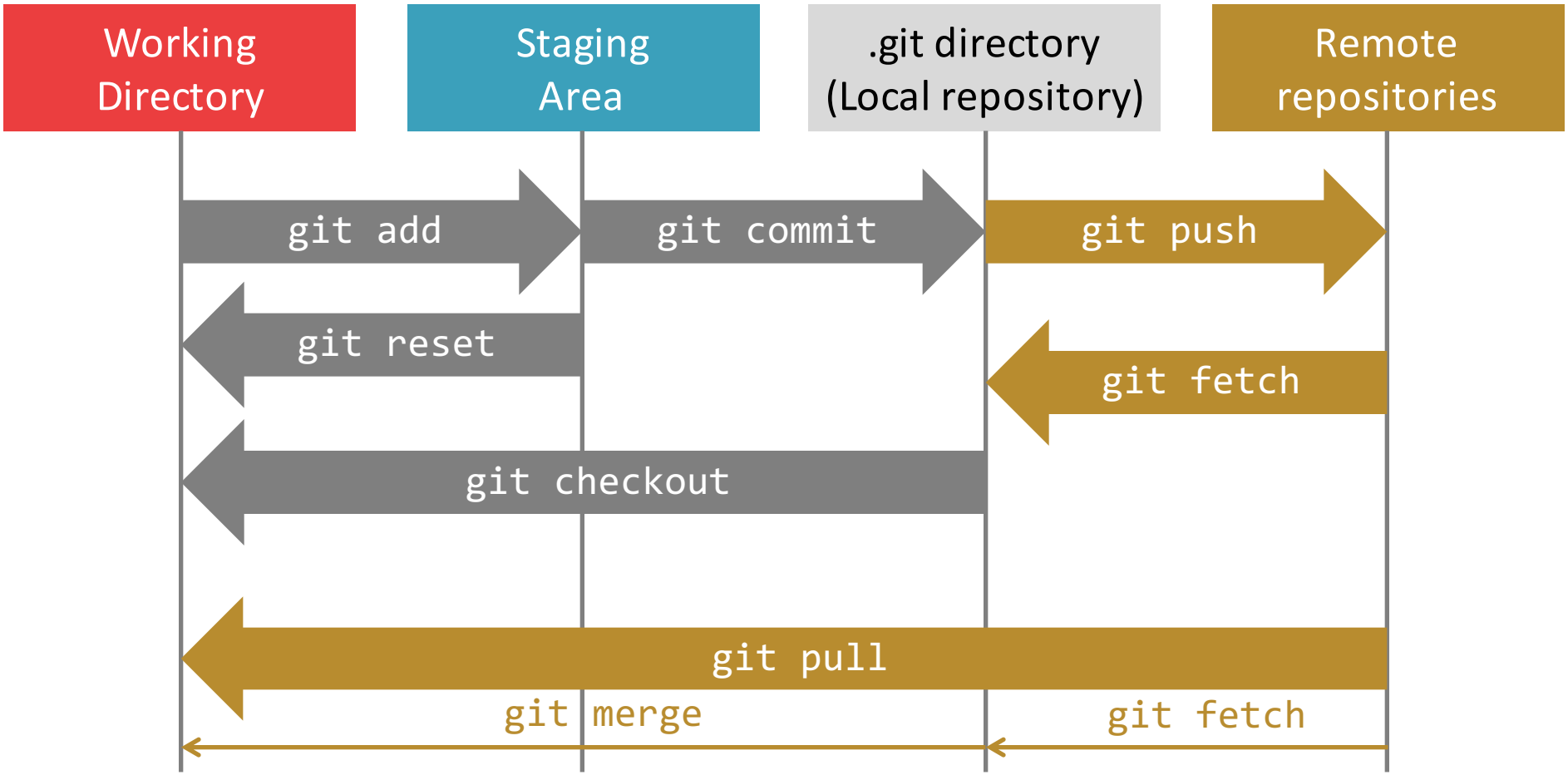
WARNING! If undo changes that are not committed yet, these are lost and cannot be recovered

3. WORK WITH A REMOTE REPOSITORY

Remote repositories

- Allow to keep a copy of your repository to other computers in case of failures
- Allow to share the work with other people:
  - Public repository (e.g. the repo where these slides are stored)
    - Everyone has read access
    - Only specific users have write access
  - Private repository
    - Only specific users have read access
    - Only specific users have write access

How GIT works (local & remote)



How to obtain a remote repository

GitHub (https://github.com)

- Public repositories for free
- Private repositories not available with the free plan
- Unlimited number of users can share your private repos

Bitbucket (https://bitbucket.com)

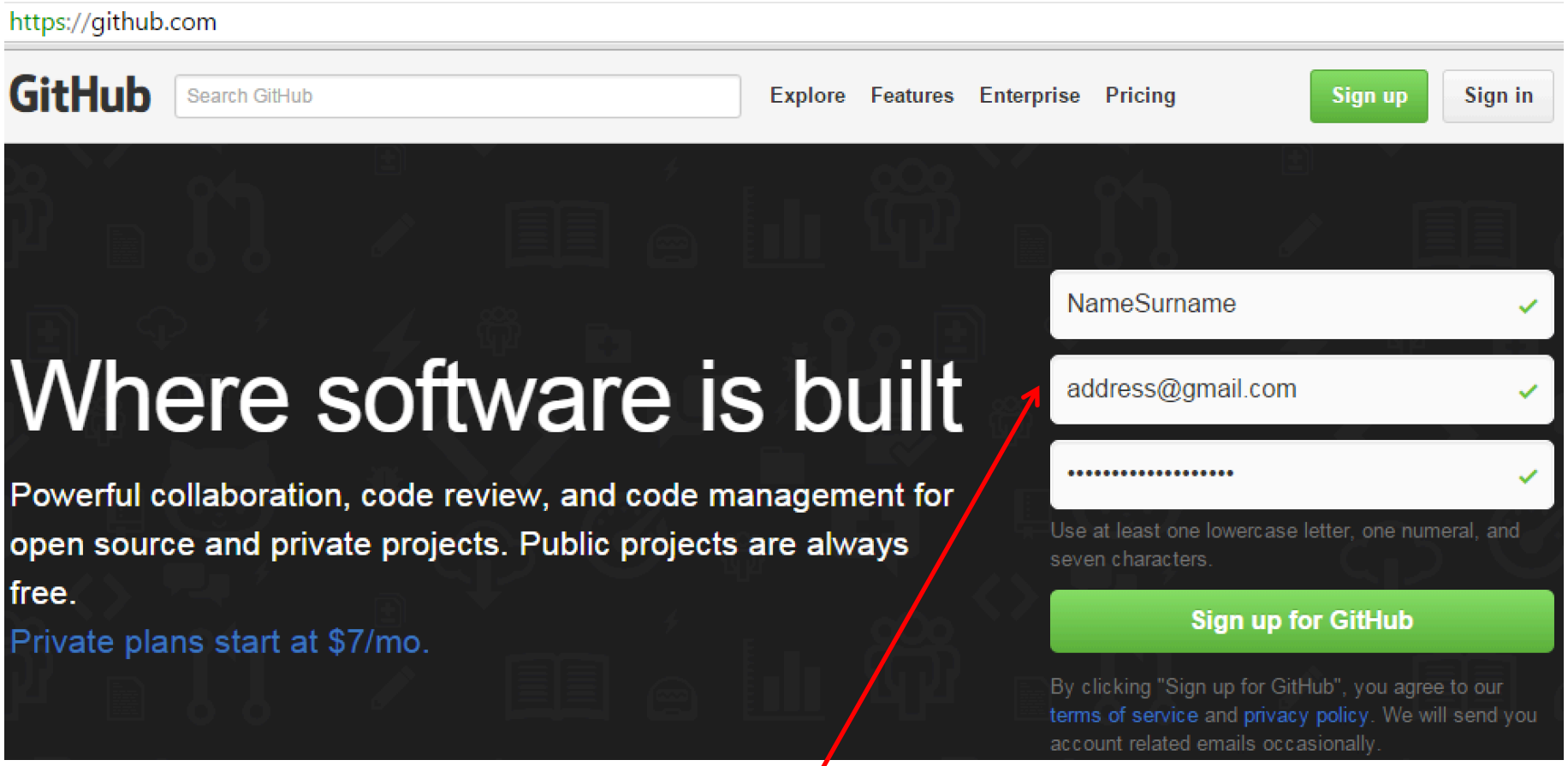
- Public repositories for free
- Private repositories for free
- Number of users sharing private repositories limited to 5, pay for extra users

- Both have a soft limits of 1 Gb storage for each repo

Info update at 27/04/2016

Create a GitHub account

Go to: https://github.com



If possible use the same email address used for “git config –global --user-email”

Create a GitHub account

Welcome to GitHub

You've taken your first step into a larger world, **@marcorabozzi**.

Completed	Step 2:	Step 3:
Set up a personal account	Choose your plan	Go to your dashboard

Choose your personal plan

Plan	Cost (view in EUR)	Private repositories	
Large	$50/month	50	Choose
Medium	$22/month	20	Choose
Small	$12/month	10	Choose
Micro	$7/month	5	Choose
Free	$0/month	0	Chosen

Each plan includes:

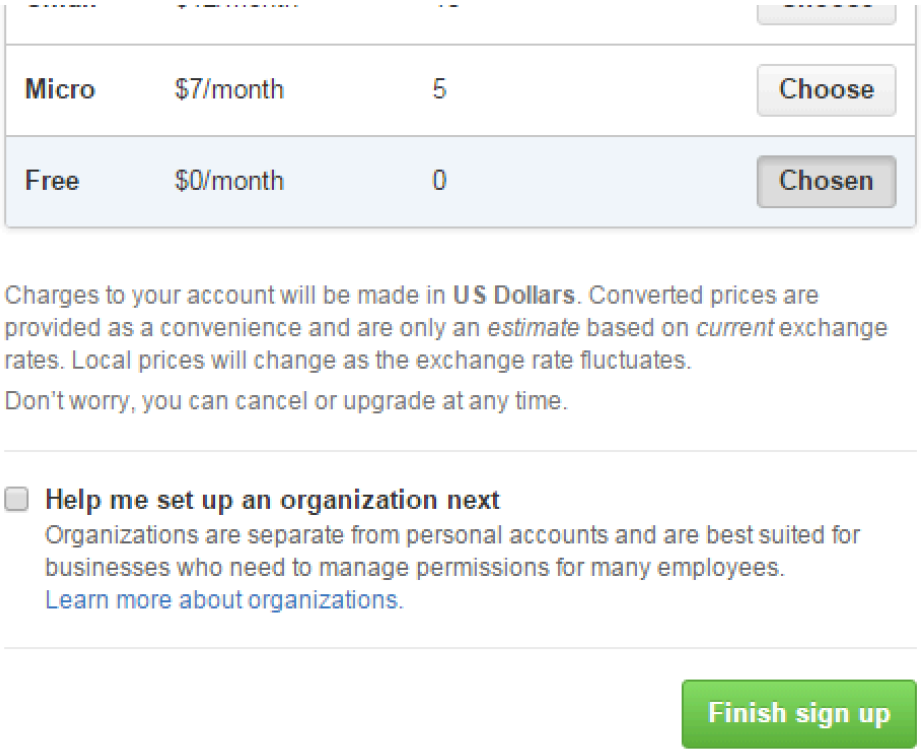
Unlimited collaborators

Unlimited public repositories

- Free setup
- HTTPS Protection
- Email support
- Wikis, Issues, Pages, & more

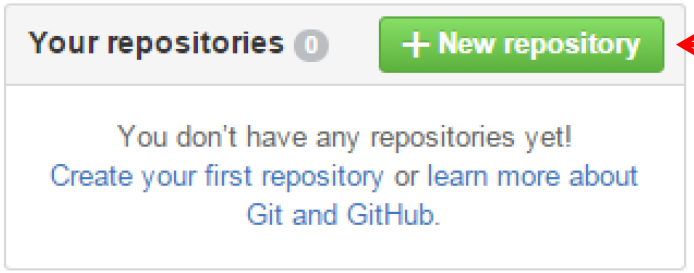
Select the free plan

Create a GitHub account

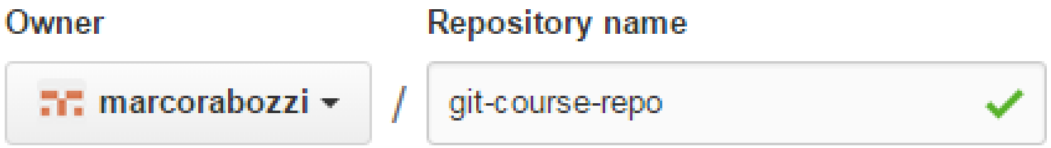


Confirm account creation

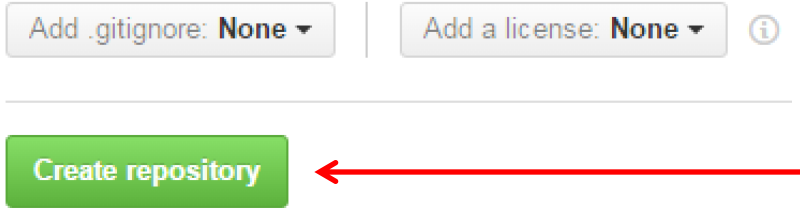
Create a remote repository



Create a new repo



Name the repo as: git-course-repo



Use default options and press “create repository”

Create a remote repository

Initialize remote repo from local repo

- First of all, add the reference to the remote repo
  - `git remote add <alias> <remote_url>`
  - <alias> is how you name your remote repo
  - <remote_url> is the reference to the remote repo

```
Marco@Marco-PC MINGW64 ~/project (master)
$ git remote add origin https://github.com/marcorabo/git-course-repo.git
```

- Check that your remote has been added using command:
  - `git remote -v`

```
Marco@Marco-PC MINGW64 ~/project (master)
$ git remote -v
origin  https://github.com/marcorabo/git-course-repo.git (fetch)
origin  https://github.com/marcorabo/git-course-repo.git (push)
```

Push your work to the remote repo

- If your remote repo is empty, no remote branches are available
- To push the commits of your local branch to a remote branch, use the following:
  - `git push <remote_alias> <branch_to_push>`

```
Marco@Marco-PC MINGW64 ~/project (master)
$ git push origin master
Username for 'https://github.com': marcorabo
Password for 'https://marcorabo@github.com':
Counting objects: 27, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (24/24), done.
Writing objects: 100% (27/27), 2.26 KiB | 0 bytes/s, done.
Total 27 (delta 11), reused 0 (delta 0)
To https://github.com/marcorabo/git-course-repo.git
 * [new branch]      master -> master
```

Insert username and password

Push your work to the remote repo

- It is possible to bind your local branch to a specific remote branch using the “upstream” option “-u”:
  - `git push –u origin master`
- After upstream is set, you can simply run:
  - `git push`
- If you do not want to type your password every time you push:
  - On Windows:
    - `git config --global credential.helper wincred`
  - On Linux / Mac OS X
    - `git config --global credential.helper cache`

Pull updates from the remote repo

- To update your repository and get the changes **pushed** by others on the remote repo, we can use the pull command:
  - `git pull origin master`
- If upstream was previously set (with push -u), you can simply run:
  - `git pull`

```
Marco@Marco-PC MINGW64 ~/Desktop/Project/repo2 (master)
$ git pull
remote: Counting objects: 6, done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 6 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (6/6), done.
From https://github.com/marcorabo/git-course-repo
   e43ceff..faa7ae6  master     -> origin/master
Updating e43ceff..faa7ae6
Fast-forward
 index.html | 4 ++++
 1 file changed, 4 insertions(+)
```

Initialize local repo from remote repo

- If someone else has already created a repo for you and you want a copy to work on locally use:
  - `git clone <remote_url>`

```
Marco@Marco-PC MINGW64 ~/slides
$ git clone https://github.com/marcorabo/brief-git-course.git
Cloning into 'brief-git-course'...
remote: Counting objects: 7, done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 7 (delta 1), reused 7 (delta 1), pack-reused 0
Unpacking objects: 100% (7/7), done.
Checking connectivity... done.
```

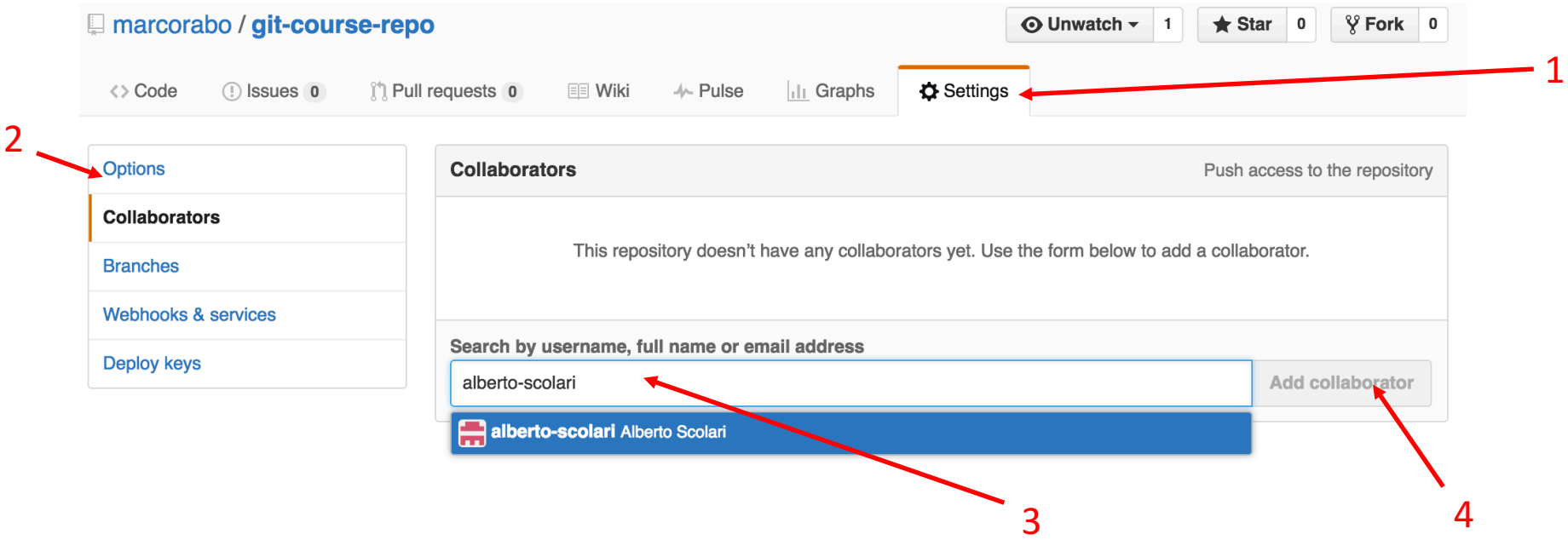
Git clone

- Creates a copy of a remote repository into a folder in your current working directory
- Creates a local branch “master” and sets upstream for master -> origin/master

```
Marco@Marco-PC MINGW64 ~/slides/brief-git-course (master)
$ git branch -vv
* master f2faca6 [origin/master] removed date from slides
```

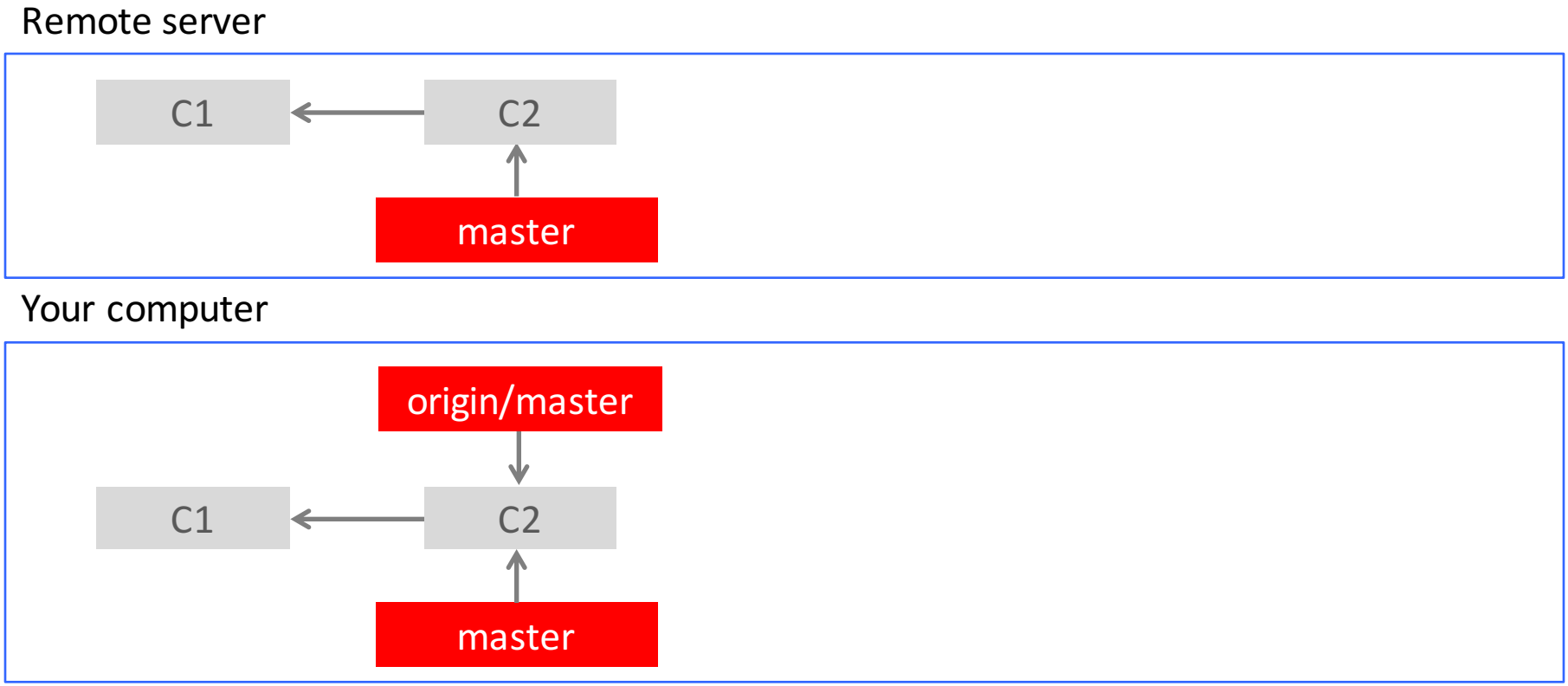
Add collaborators to a GitHub repo

- Only users with write access can push content to a remote repository
- Add a collaborator to give write access to your remote repo:



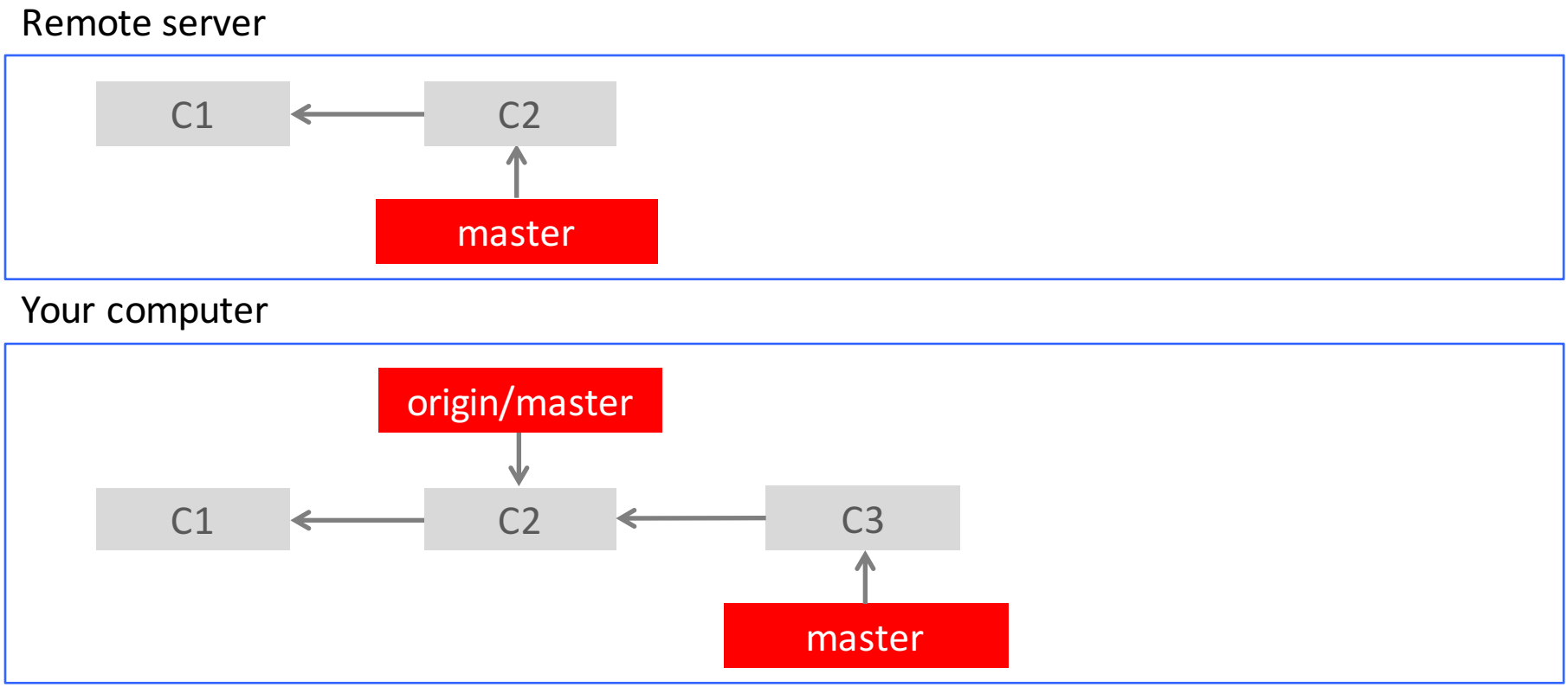
Working with remotes

- Assume the following situation



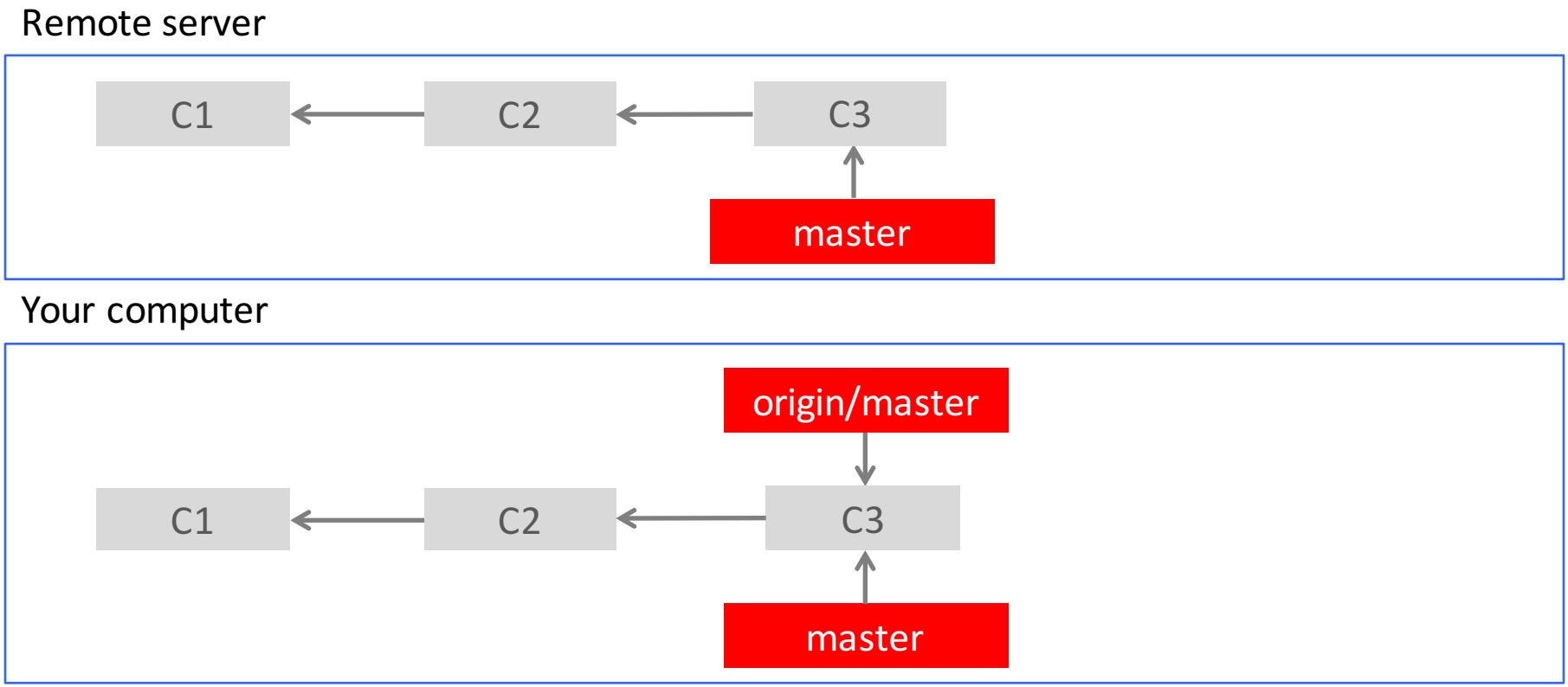
Working with remotes

- Do a commit on your computer



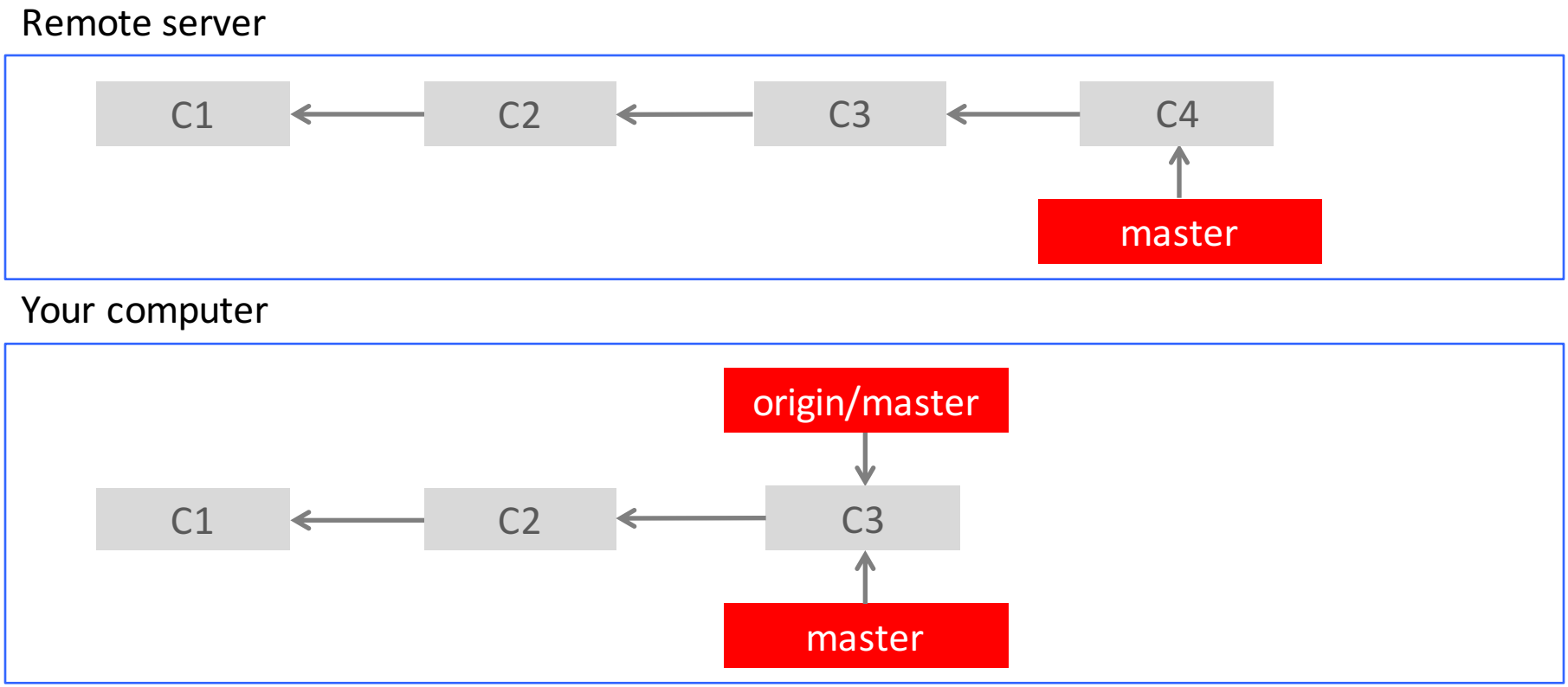
Working with remotes

- Push changes to remote: `git push origin master`



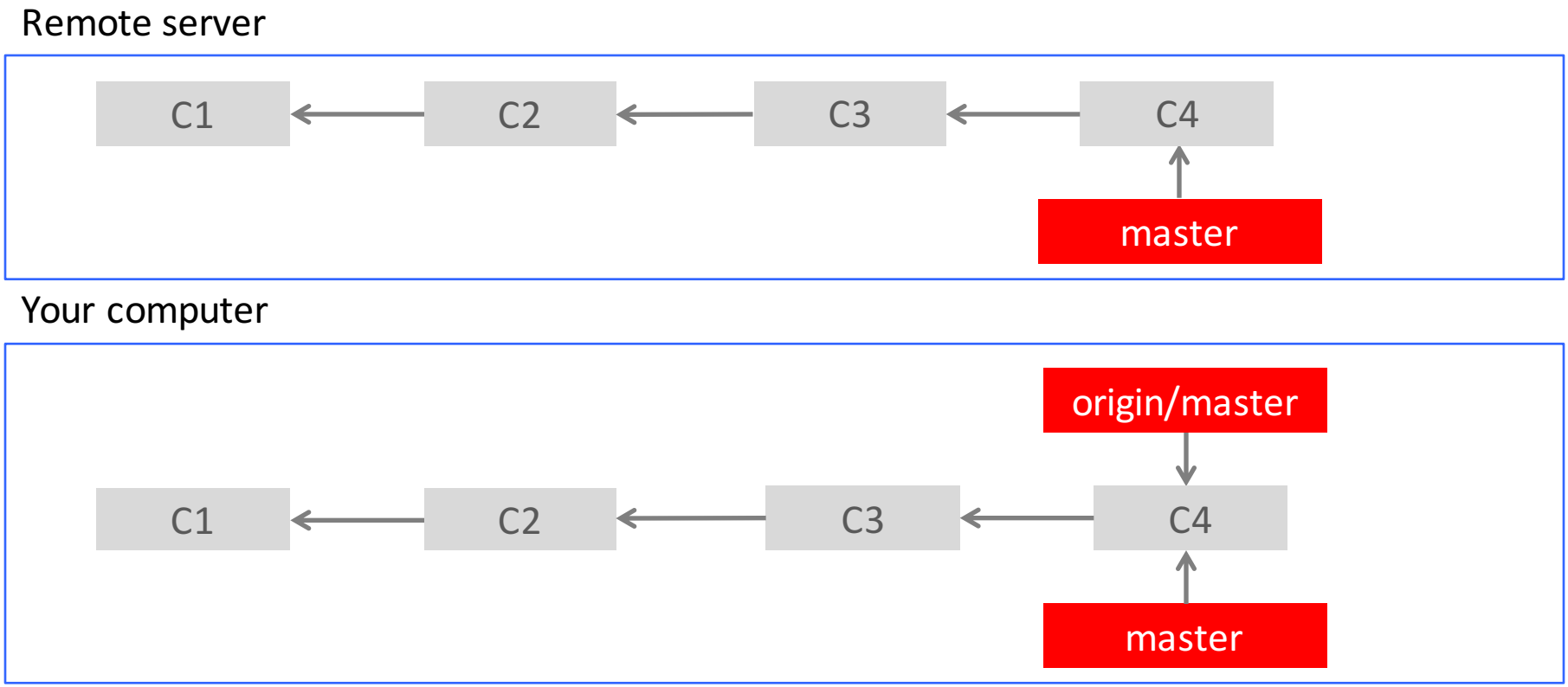
Working with remotes

- Someone else on your team pushes a commit



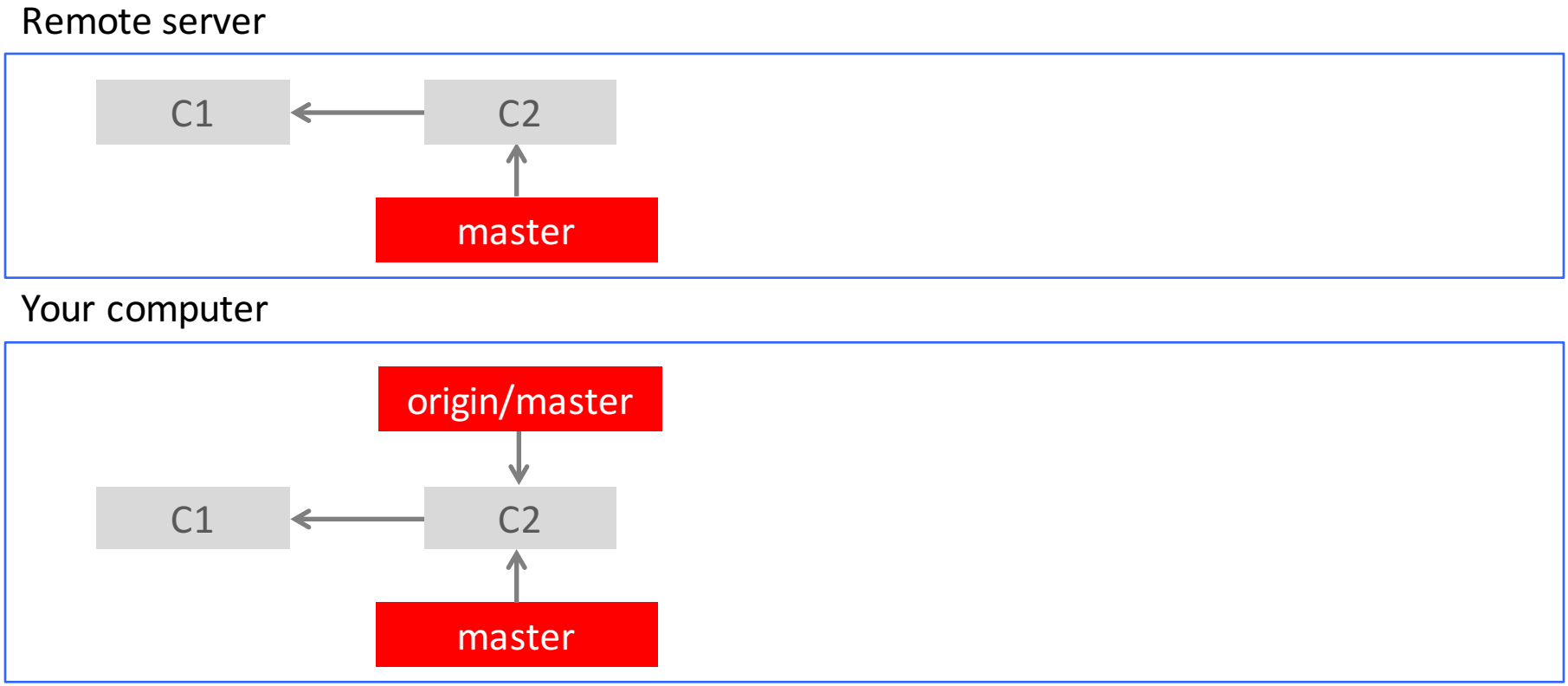
Working with remotes

- To synchronize your local repo: `git pull origin master`



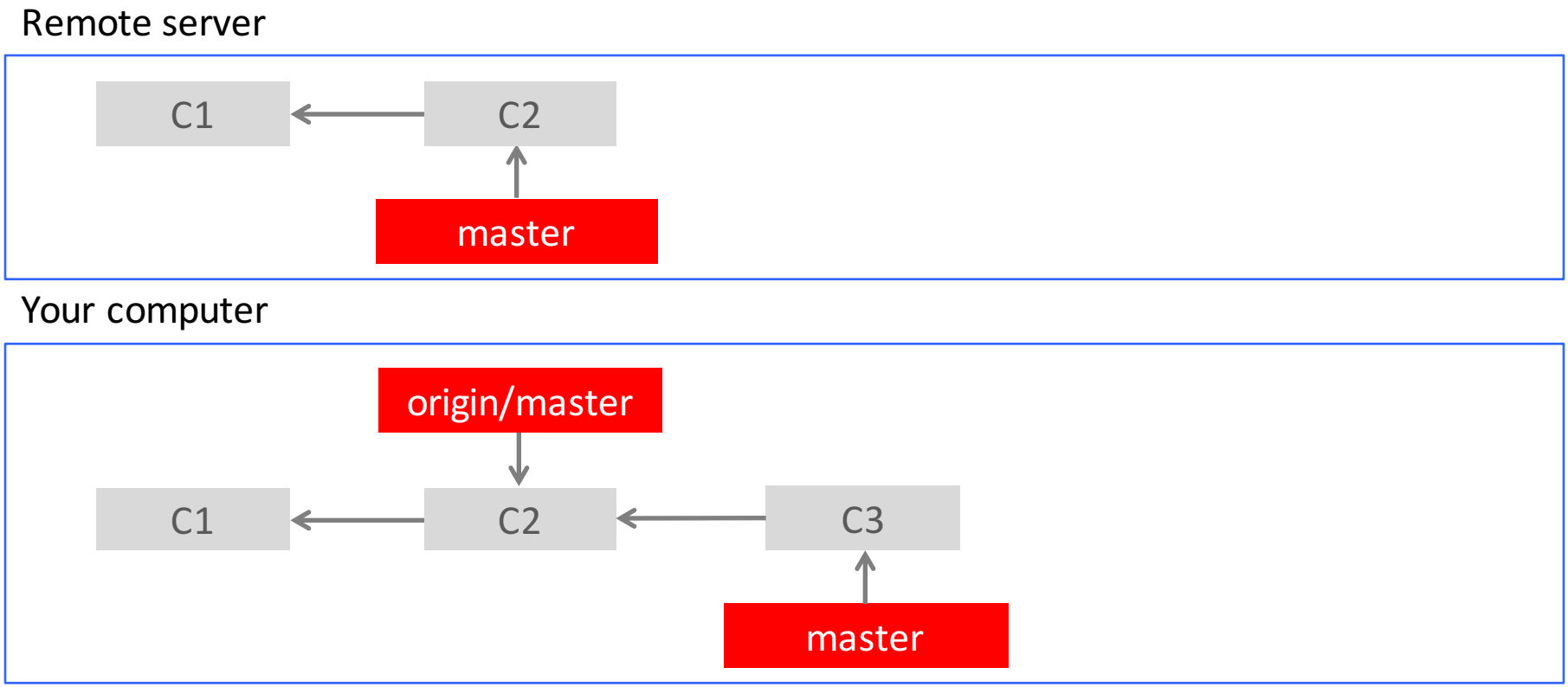
Working with remotes (divergent work)

- Assume the following situation



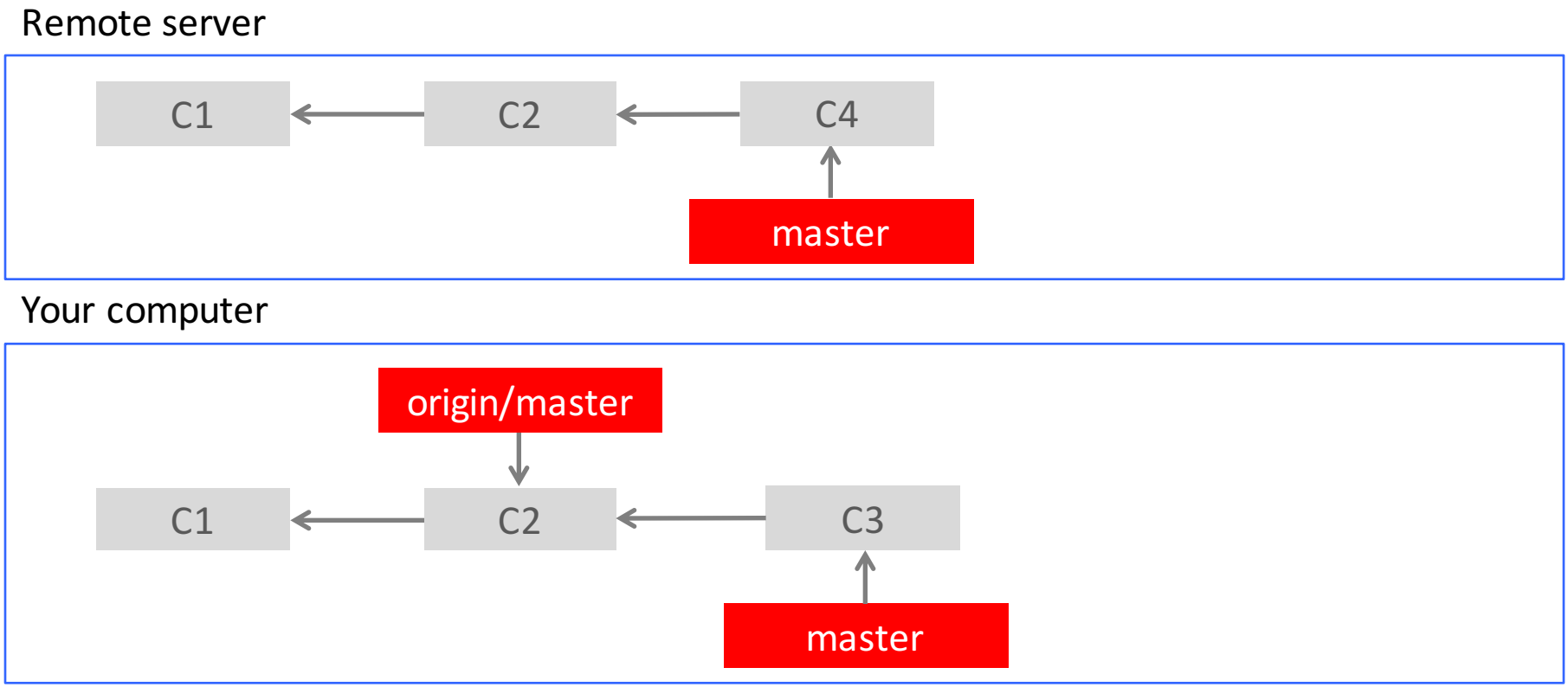
Working with remotes (divergent work)

- Do a commit on your computer



Working with remotes (divergent work)

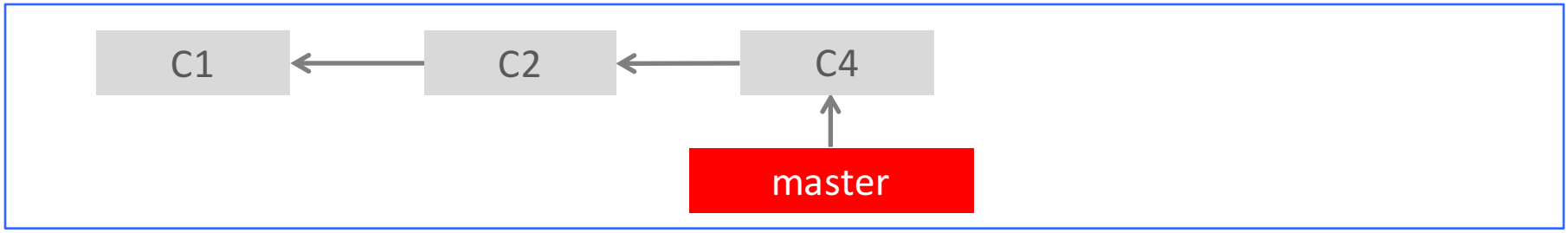
- Someone else on your team pushes a commit



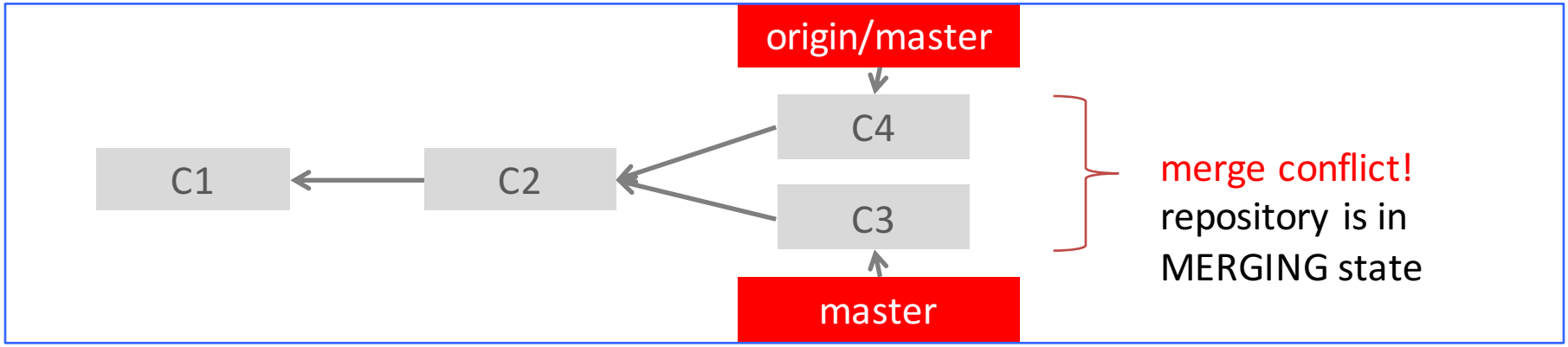
Working with remotes (divergent work)

- You try to **push** but the command fails because remote repo has changed!
- To sync with the remote repo you run **pull** and discover that the same files were modified by you and your colleague -> merge conflict

Remote server



Your computer



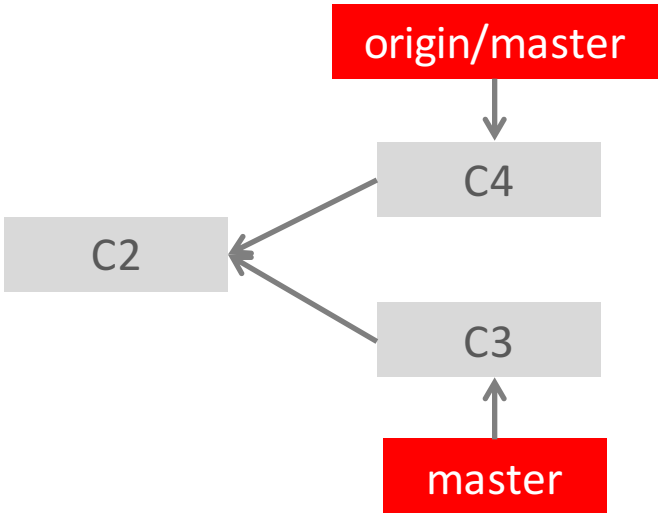
Conflict example

page.html (C4 origin/master)

```
<html>
<head>
</head>
<body>
        <h1>My title</h1>
</body>
</html>
```

page.html (C3 master)

```
<html>
<head>
</head>
<body>
        <h1>New title</h1>
</body>
</html>
```



- What happen when we **pull**?

Conflict example

```
Marco@Marco-PC MINGW64 ~/Desktop/Project/git-course-repo
(master)
$ git pull
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/marcorabo/git-course-repo
   1211c78..e43ceff  master     -> origin/master
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
Automatic merge failed; fix conflicts and then commit the result.

Marco@Marco-PC MINGW64 ~/Desktop/Project/git-course-repo (master|MERGING)
$ git status
On branch master
Your branch and 'origin/master' have diverged,
and have 1 and 1 different commit each, respectively.
  (use "git pull" to merge the remote branch into yours)
You have unmerged paths.
  (fix conflicts and run "git commit")

Unmerged paths:
  (use "git add <file>..." to mark resolution)

        both modified:   index.html

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

Resolve the conflict

page.html

```
<html>
<head>
</head>
<body>
<<<<<<< HEAD
        <h1>New title</h1>
=======
        <h1>My title</h1>
>>>>>>> e43ceff697364f5a959e5
</body>
</html>
```

Our version (HEAD)

Version on the remote repo

```
<html>
</head>
<head>
<body>
        <h1>correct title</h1>
</body>
</html>
```

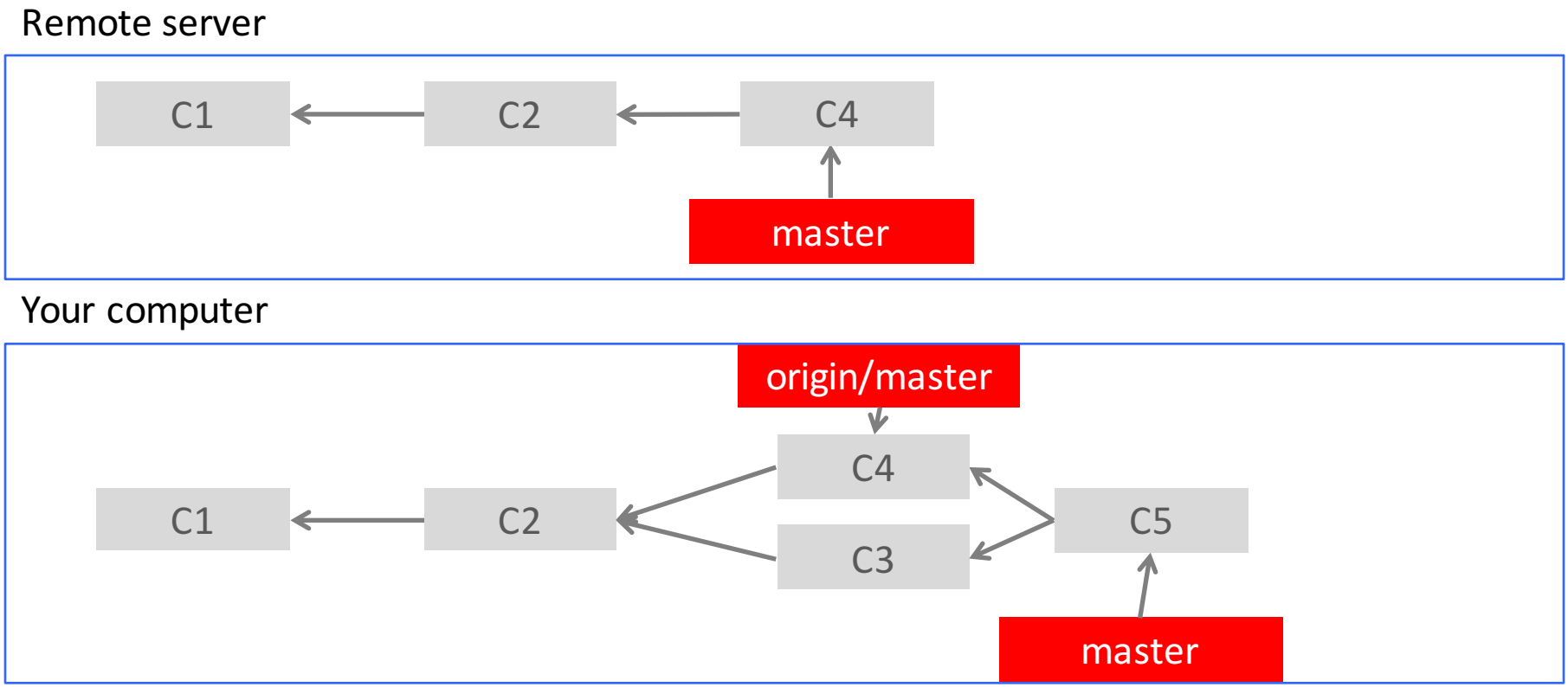
File fixed manually

Resolve the conflict

```
Marco@Marco-PC MINGW64 ~/Desktop/Project/git-course-repo (master|MERGING)
$ git add index.html

Marco@Marco-PC MINGW64 ~/Desktop/Project/git-course-repo (master|MERGING)
$ git status
On branch master
Your branch and 'origin/master' have diverged,
and have 1 and 1 different commit each, respectively.
  (use "git pull" to merge the remote branch into yours)
All conflicts fixed but you are still merging.
  (use "git commit" to conclude merge)

Changes to be committed:

        modified:   index.html


Marco@Marco-PC MINGW64 ~/Desktop/Project/git-course-repo (master|MERGING)
$ git commit -m"fixed index title"
[master faa7ae6] fixed index title

Marco@Marco-PC MINGW64 ~/Desktop/Project/git-course-repo (master)
$
```

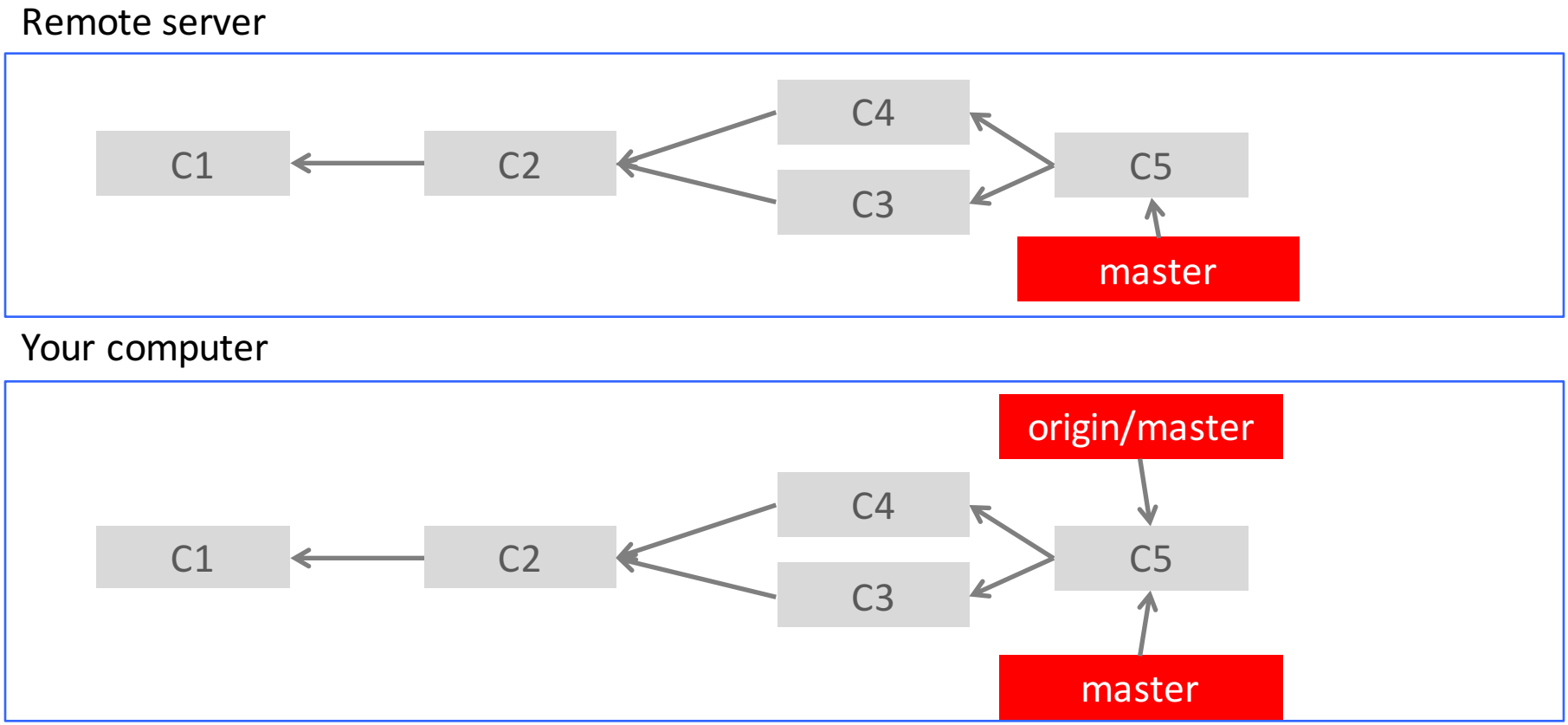
Working with remotes (divergent work)

- Now you need to commit your merge fix



Working with remotes (divergent work)

- Finally you push your work: `git push origin master`



Working with remotes simple flow

1. Pull last changes from remote
2. Work on your project
3. Commit changes
4. Push changes
5. If push fails:
    1. Pull last updates
    2. Commit resolution of conflicts (if needed)
    3. Push changes (your work + solved conflicts)

GIT commands recap

- git --version
- git config –-global user.name
- git config –-global user.email
- git config –-list
- git init
- git status
- git add <file>
- git reset <file>
- git commit -m “<message>”
- git log
- git log --graph
- git checkout <commit_id>
- git checkout <branch_name>
- git checkout -- <file>
- git diff HEAD
- git diff HEAD -- <file>

GIT commands recap

- `git checkout <branch name>`
- `git remote add <alias> <remote url>`
- `git remote`
- `git remote -v`
- `git clone <remote url>`
- `git push <remote alias> <branch to push>`
- `git push -u <remote alias> <branch to push>`
- `git push`
- `git pull <remote alias> <branch to pull>`
- `git pull`

Useful links

- Git Book:
  - http://git-scm.com/doc
- Git Tutorial:
  - https://www.atlassian.com/git/tutorials/
- Video lectures (check course Week 2):
  - https://class.coursera.org/datascitoolbox-032/lecture