Branches and remotes

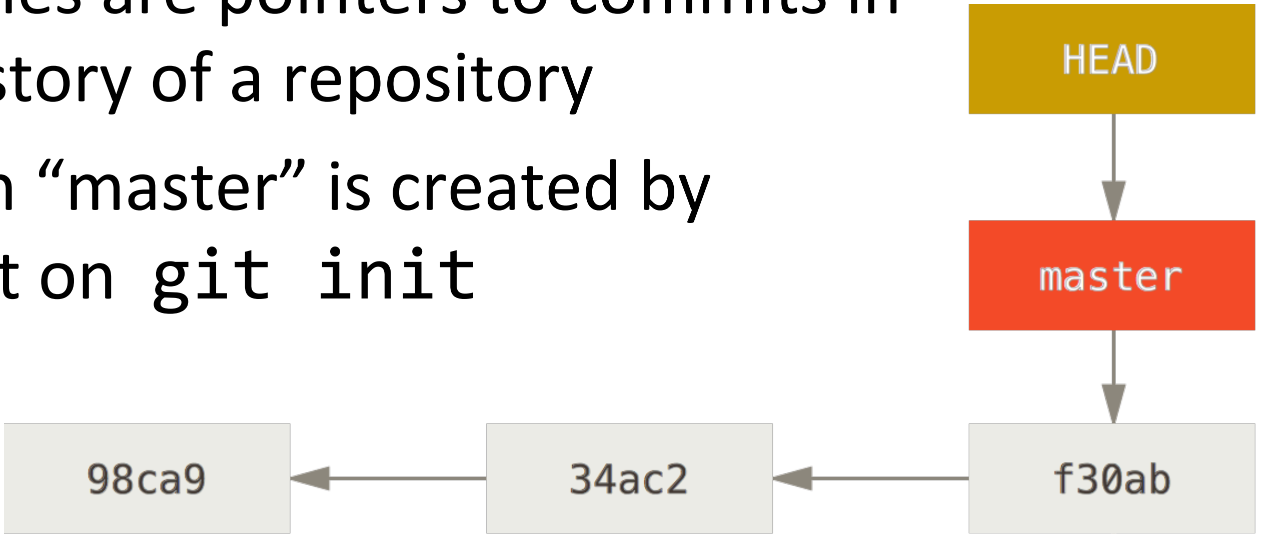
Dott. Ing. **Marco Rabozzi**
Politecnico di Milano
Email: **marco.rabozzi@polimi.it**

Credits: http://git-scm.com/book/en/v2/

POLITECNICO
MILANO 1863

What are branches in GIT?

- Branches are pointers to commits in the history of a repository
- Branch “master” is created by default on `git init`



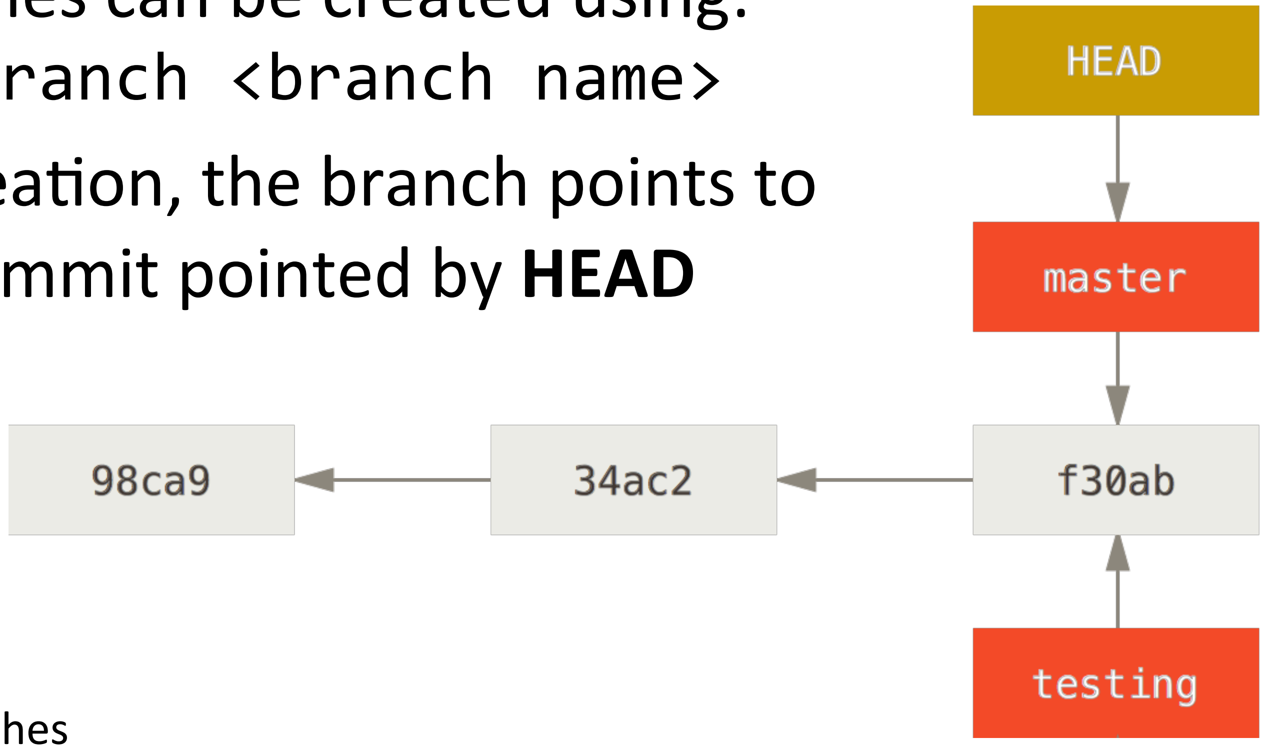
🟥 branches

⬜ commits

https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell

Create a new branch

- Branches can be created using: `git branch <branch name>`
- On creation, the branch points to the commit pointed by **HEAD**



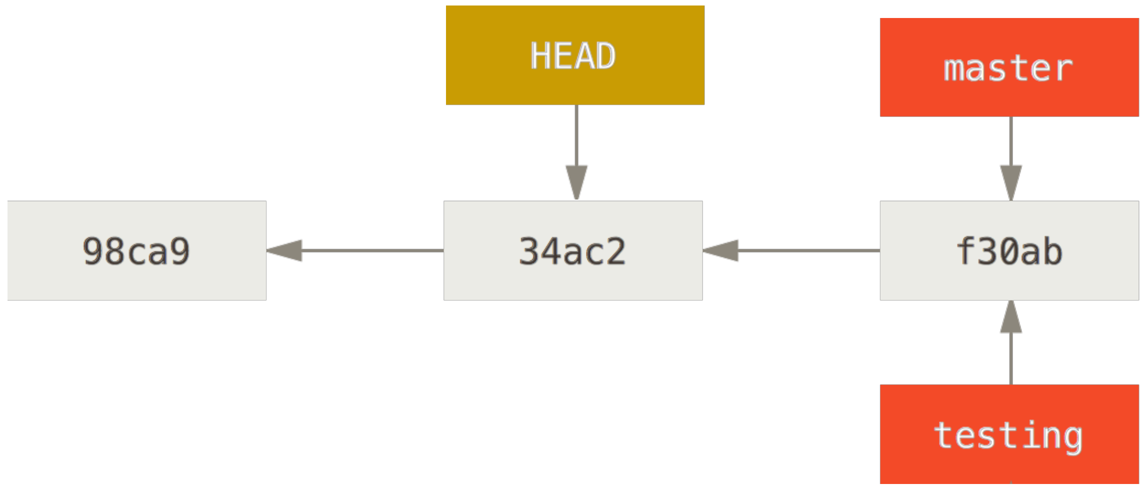
🟥 branches

⬜ commits

```
Marco@Marco-PC MINGW64 ~/project (master)
$ git branch testing
```

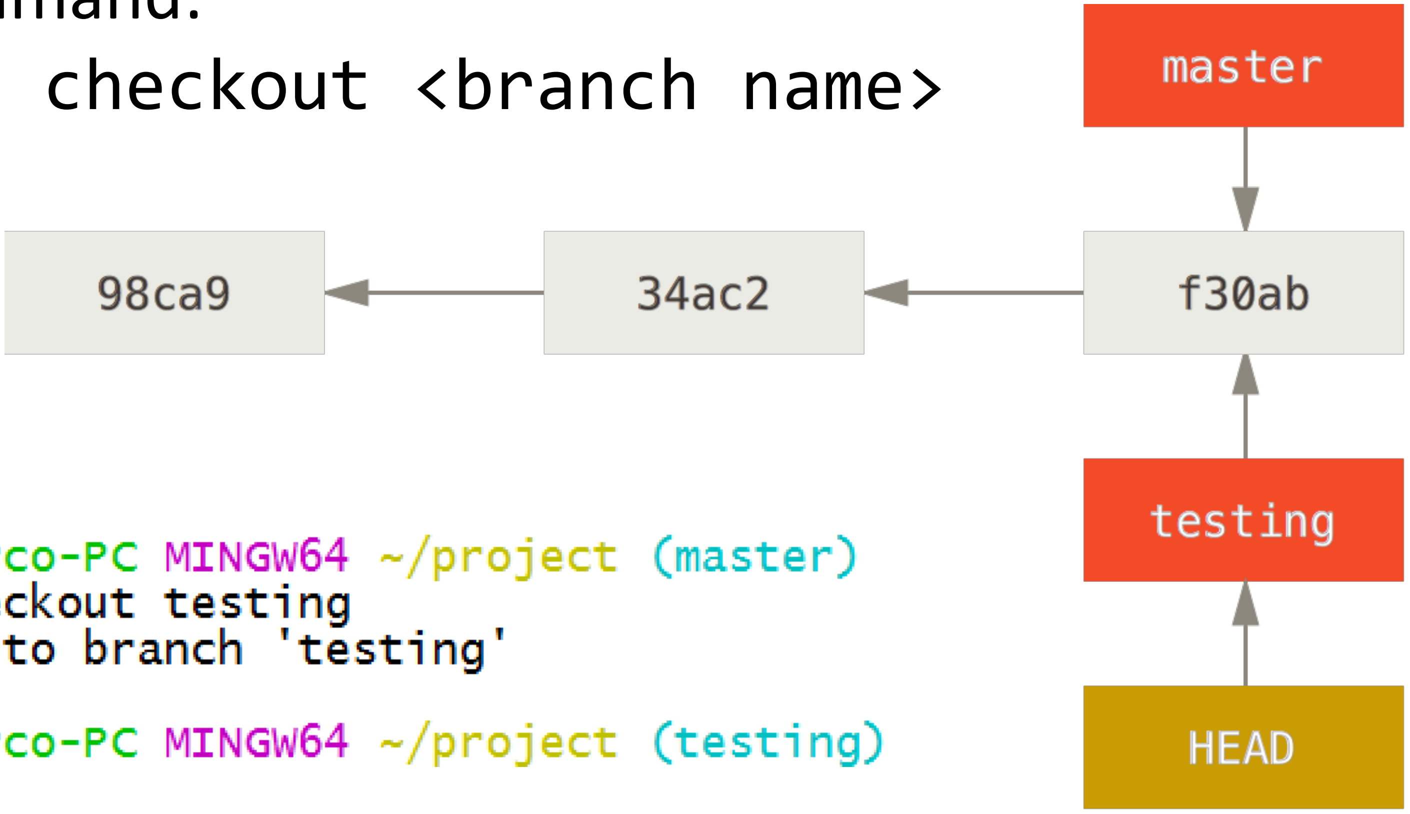
More about HEAD

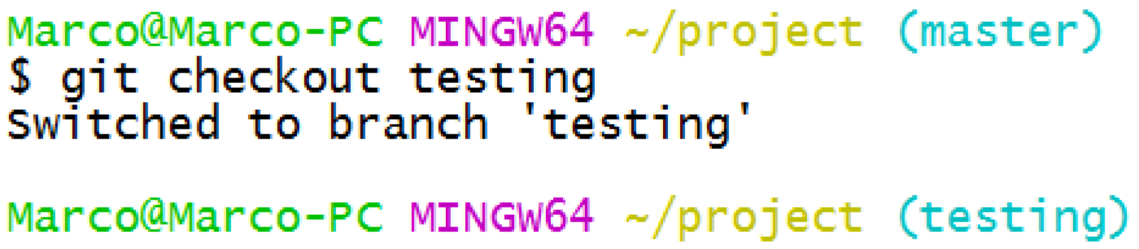
- HEAD is the pointer to the current branch and commit
- HEAD can be moved using the `checkout` command
- HEAD is <u>detached</u> if it does not point to any branch but points to a specific commit:



Switch branch

- To move the HEAD to a different branch use the command:
  `git checkout <branch name>`



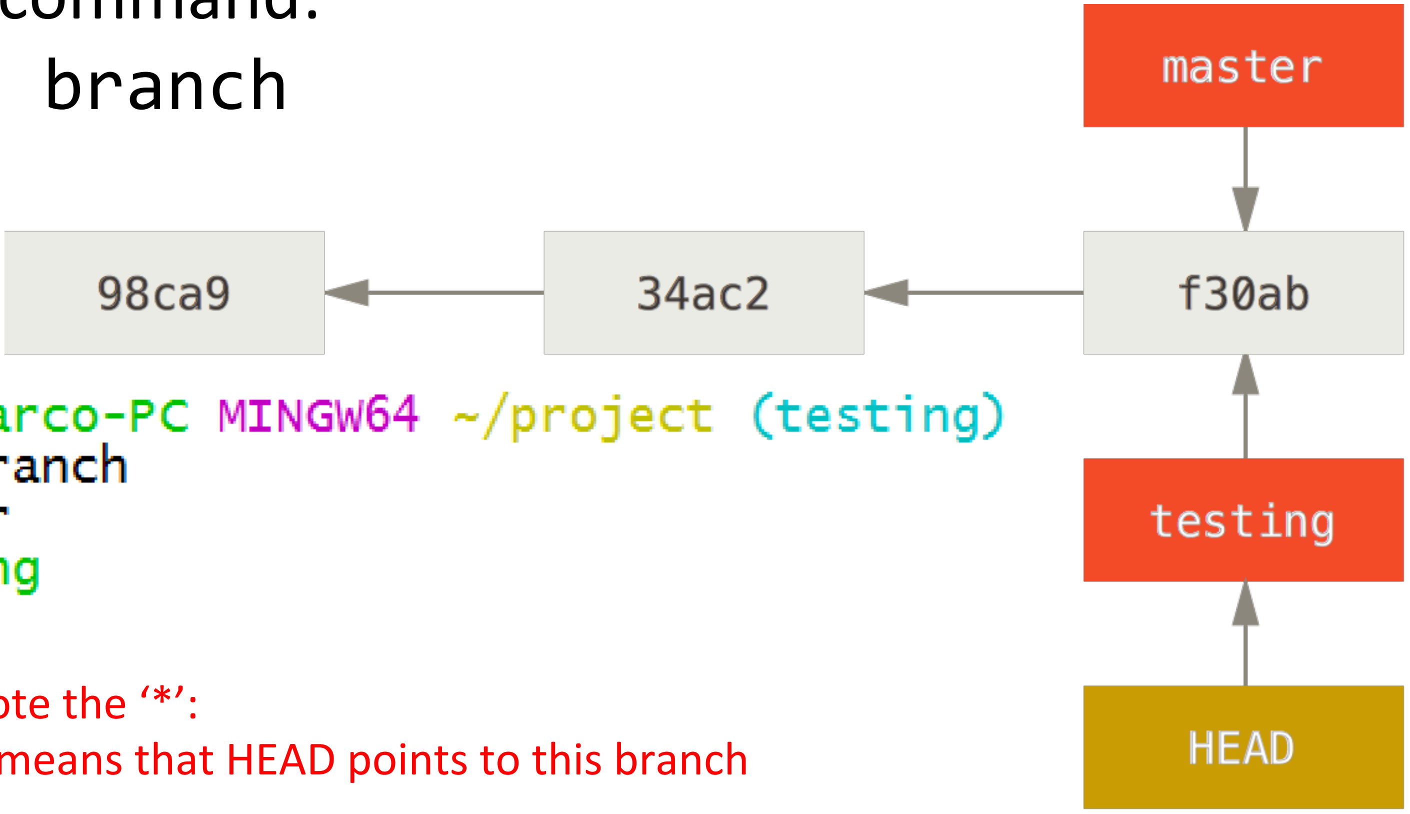


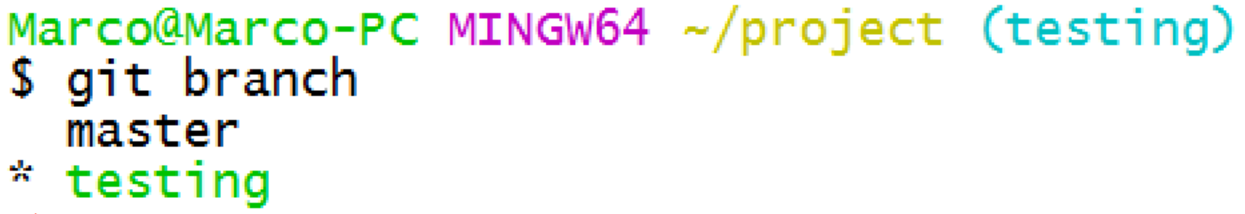
```
Marco@Marco-PC MINGW64 ~/project (master)
$ git checkout testing
Switched to branch 'testing'

Marco@Marco-PC MINGW64 ~/project (testing)
```

List branches

- To check the list of current available branches, use the command:
  `git branch`



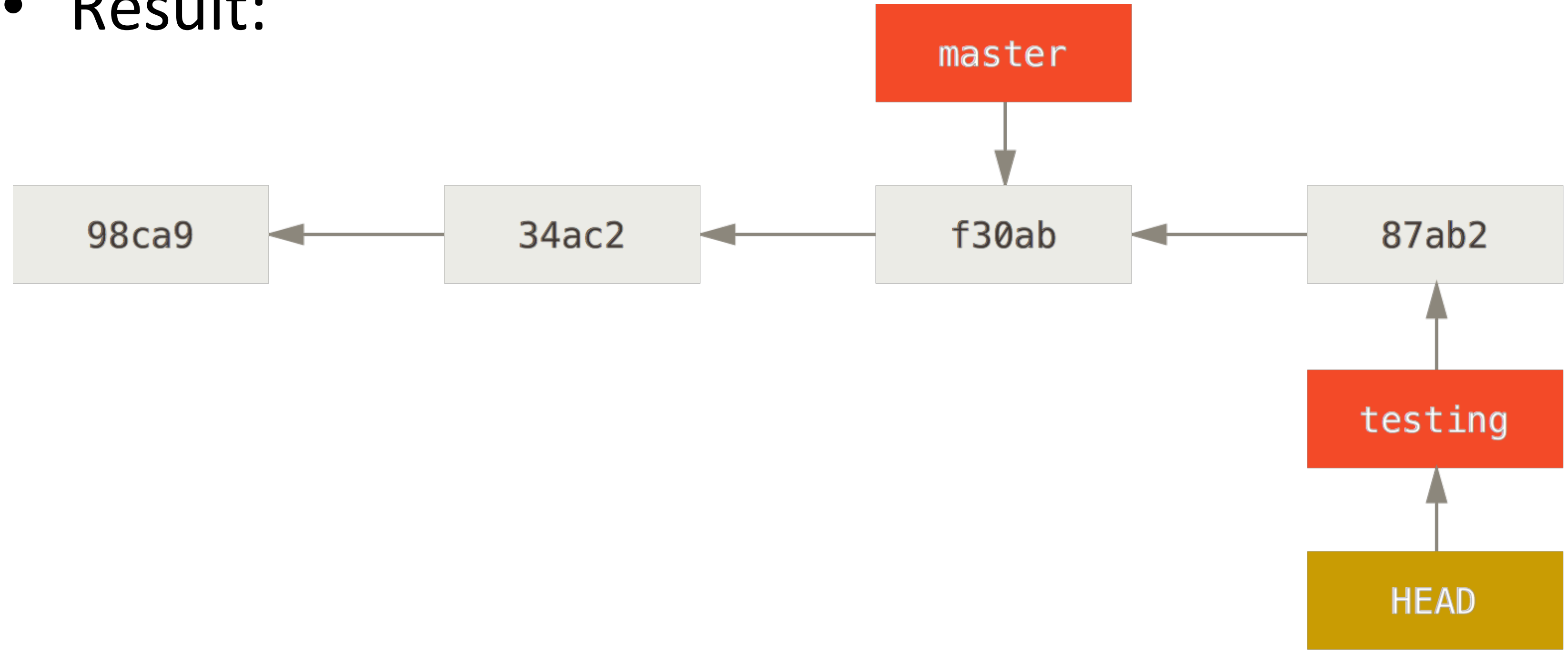


```
Marco@Marco-PC MINGW64 ~/project (testing)
$ git branch
  master
* testing
```

Note the '*':
It means that HEAD points to this branch

What happens when we commit?

- Assume that we are on branch "testing"
- We edit a file, add it, and commit
- Result:



What if we commit on the other branch?

- Now, switch to branch “master”
- Edit a **<u>different</u>** file, add it and commit

- Result? Try running the command:

```
git log --graph --decorate --all
```

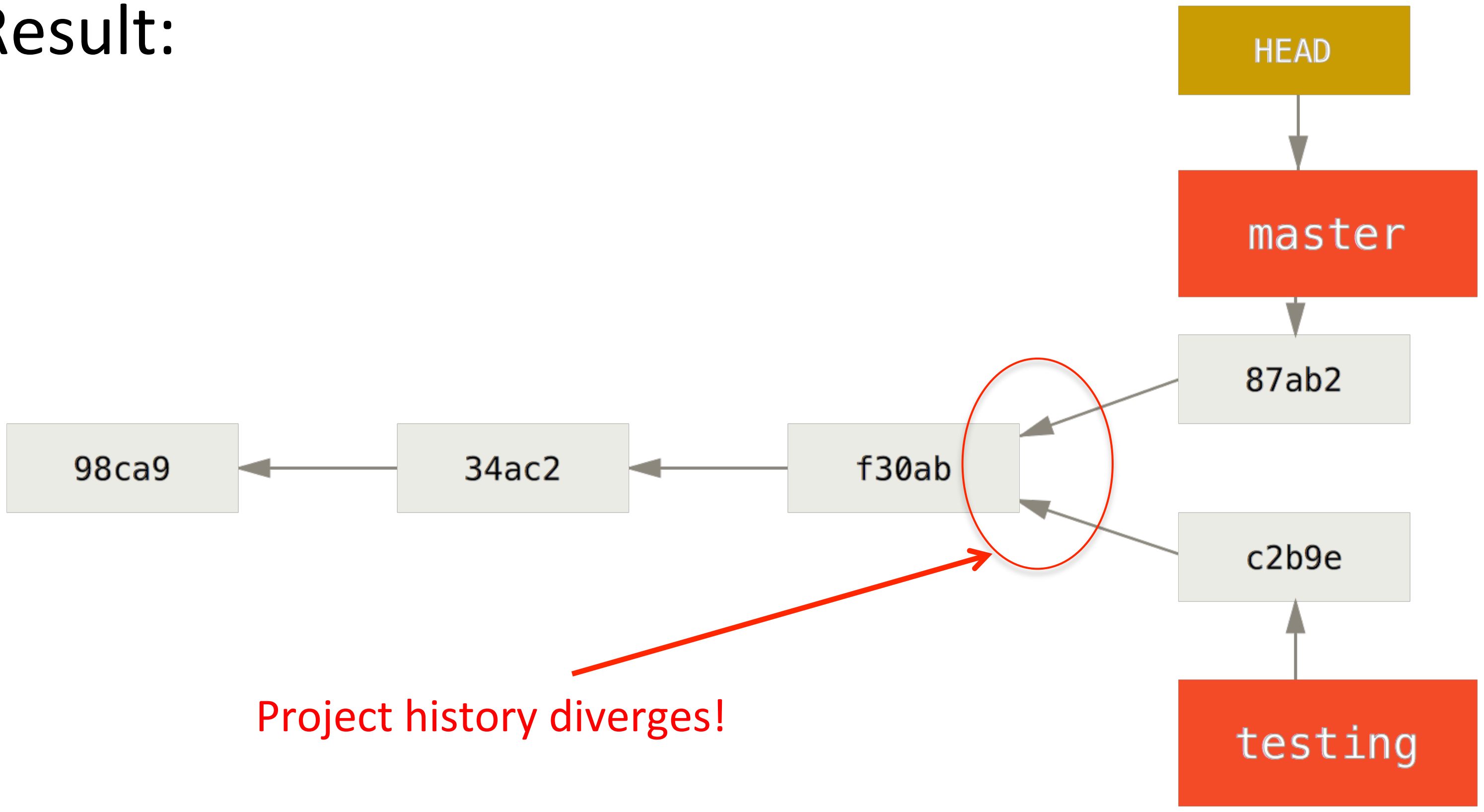
Shows lines representing commit history

Adds information about branches and HEAD position

Shows history for all the branches

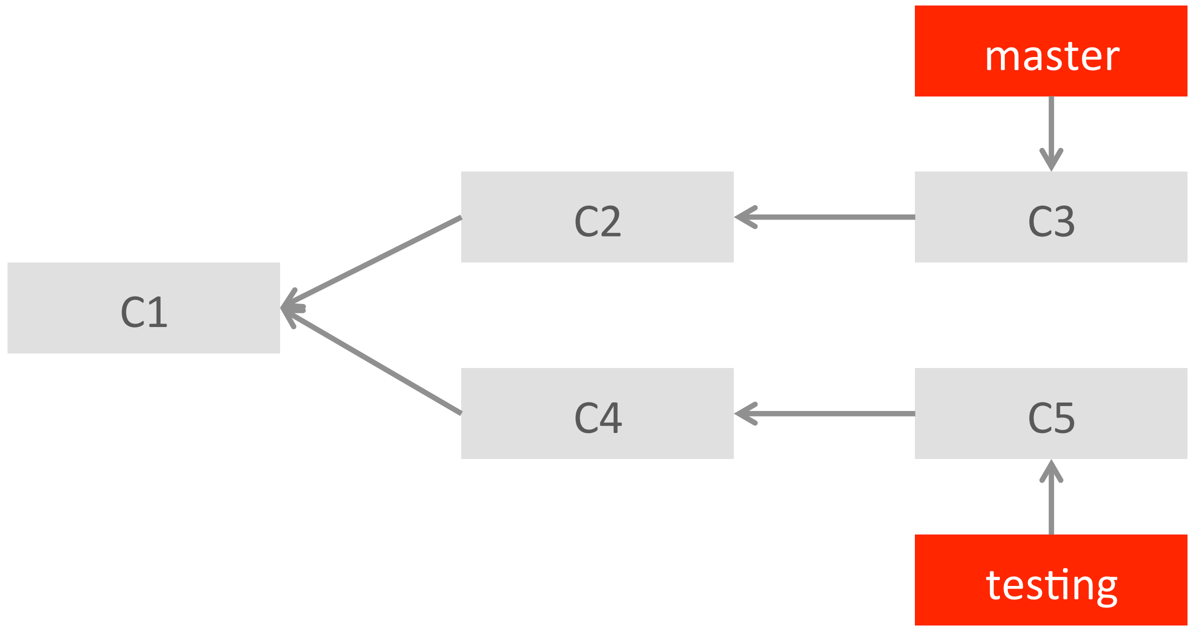
What if we commit on the other branch?

- Result:



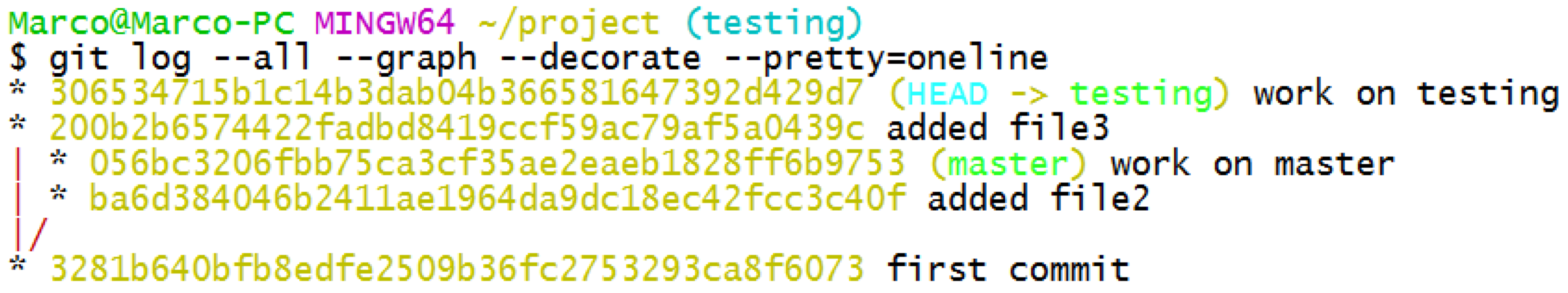
Divergent history

- After some commits on branch “testing” and “master” the history may look like this:



Divergent history

`git log --graph –decorate --all --pretty=oneline`



```
Marco@Marco-PC MINGW64 ~/project (testing)
$ git log --all --graph --decorate --pretty=oneline
* 306534715b1c14b3dab04b366581647392d429d7 (HEAD -> testing) work on testing
* 200b2b6574422fadbd8419ccf59ac79af5a0439c added file3
| * 056bc3206fbb75ca3cf35ae2eaeb1828ff6b9753 (master) work on master
| * ba6d384046b2411ae1964da9dc18ec42fcc3c40f added file2
|/
* 3281b640bfb8edfe2509b36fc2753293ca8f6073 first commit
```

Merge branches (divergent)

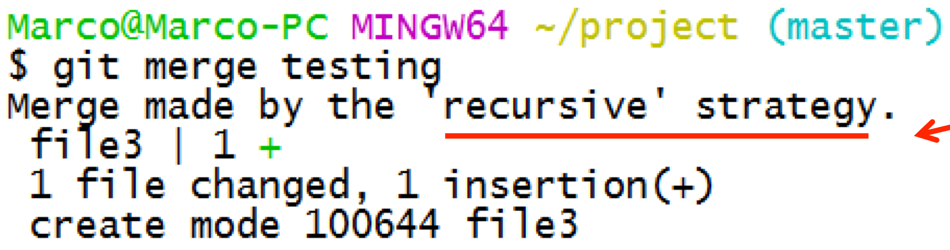
- The work done in “testing” is now stable and we are ready to merge it into “master”:

1. Switch to branch “master”
```
git checkout master
```

2. Merge branch “testing” into current branch (“master”)
```
git merge testing
```

Merge result (divergent)

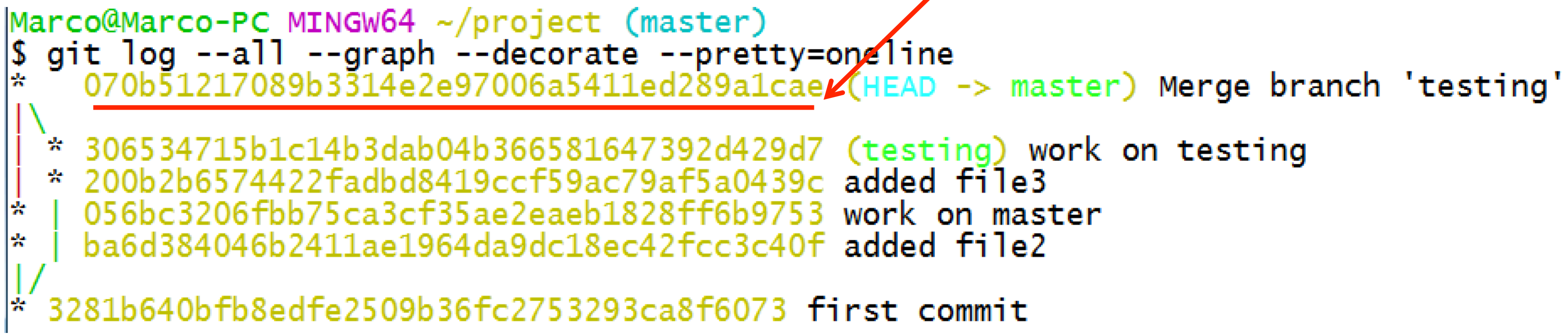
- Merge message:



```
Marco@Marco-PC MINGW64 ~/project (master)
$ git merge testing
Merge made by the 'recursive' strategy.
 file3 | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 file3
```

Strategy used in case of divergent history

- History after merge



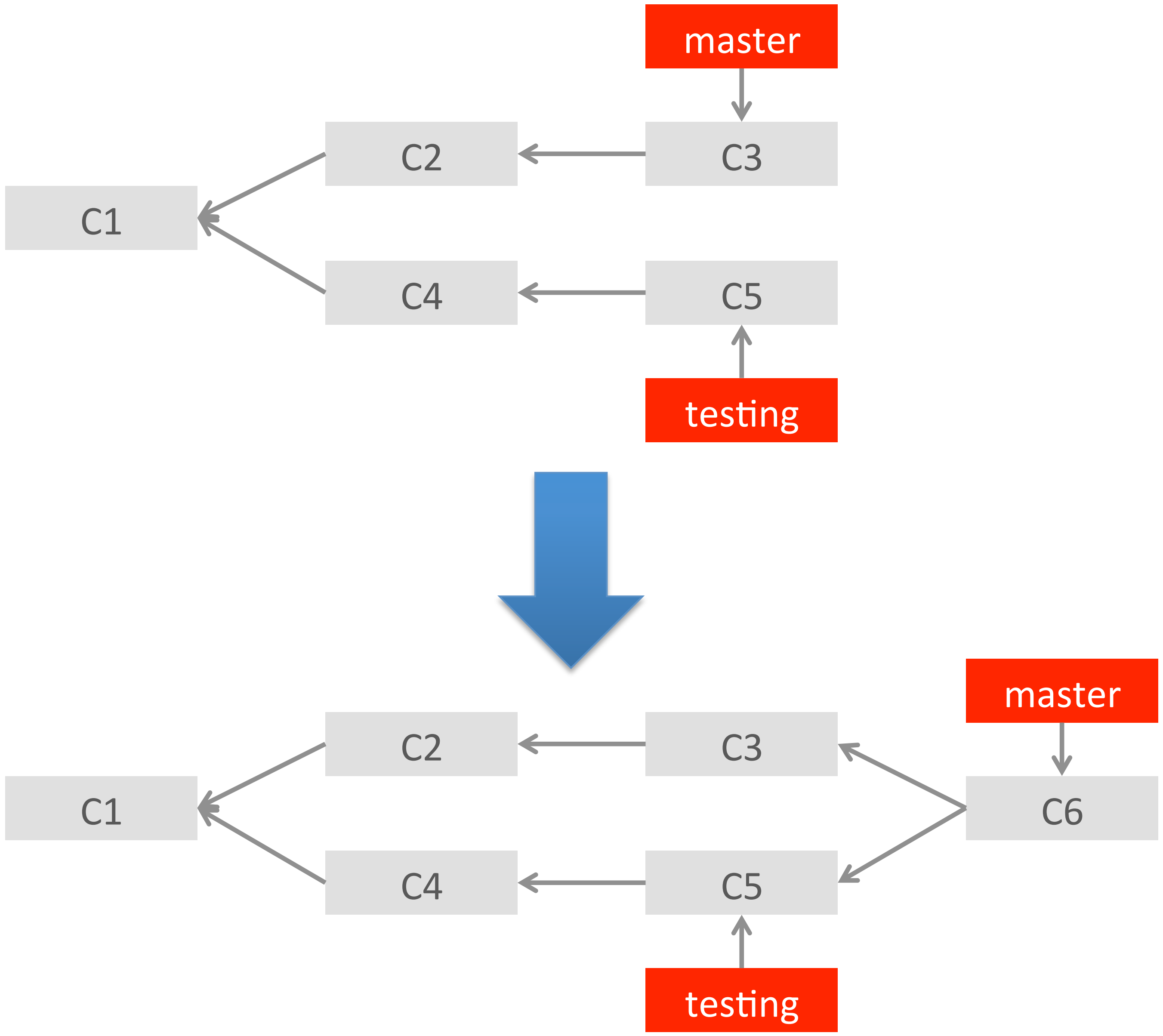
```
Marco@Marco-PC MINGW64 ~/project (master)
$ git log --all --graph --decorate --pretty=oneline
*   070b51217089b3314e2e97006a5411ed289a1cae (HEAD -> master) Merge branch 'testing'
|\
| * 306534715b1c14b3dab04b366581647392d429d7 (testing) work on testing
| * 200b2b6574422fadbd8419ccf59ac79af5a0439c added file3
* | 056bc3206fbb75ca3cf35ae2eaeb1828ff6b9753 work on master
* | ba6d384046b2411ae1964da9dc18ec42fcc3c40f added file2
|/
* 3281b640bfb8edfe2509b36fc2753293ca8f6073 first commit
```

Automatic merge commit generated

POLITECNICO MILANO 1863

Before merge…

…After merge



Merge branches (forward)

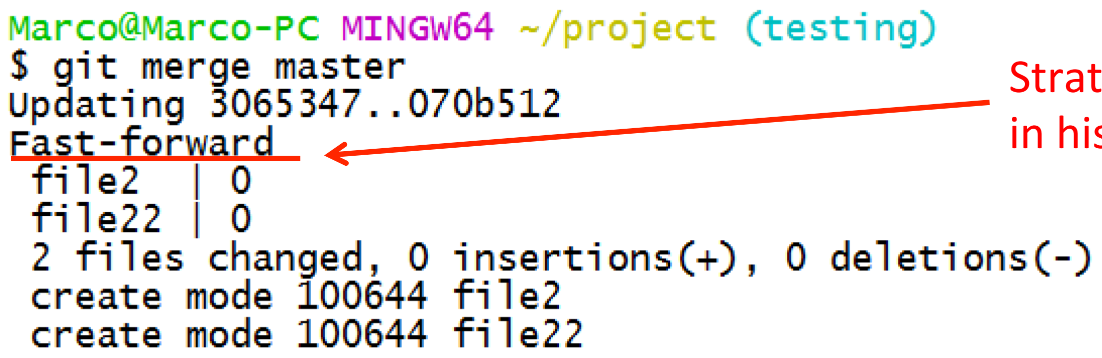
- After the merge of a divergent history into “master”, branch “testing” is behind “master”. To update “testing” do:

1. Switch to branch “testing”
   ```
   git checkout testing
   ```

2. Marge branch “master” into current branch (“testing”)
   ```
   git merge master
   ```

Merge result (forward)

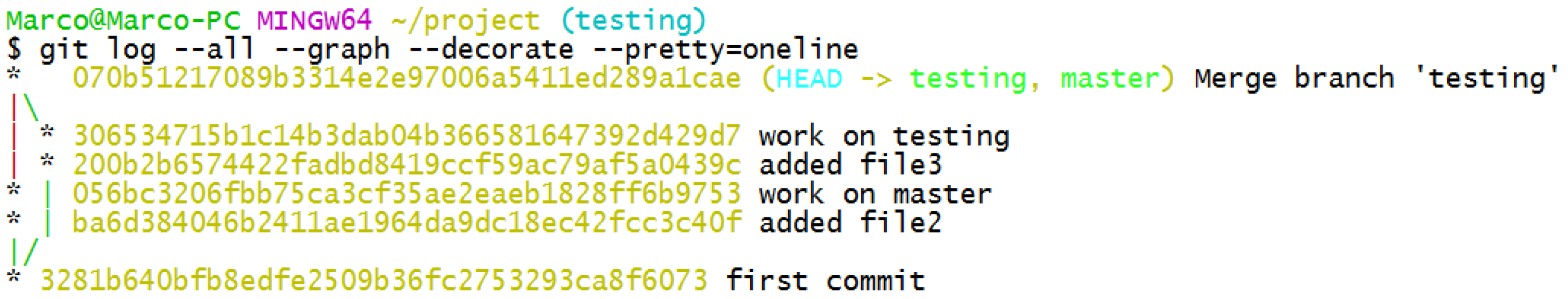
- Merge message:



```
Marco@Marco-PC MINGW64 ~/project (testing)
$ git merge master
Updating 3065347..070b512
Fast-forward
 file2  | 0
 file22 | 0
 2 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 file2
 create mode 100644 file22
```

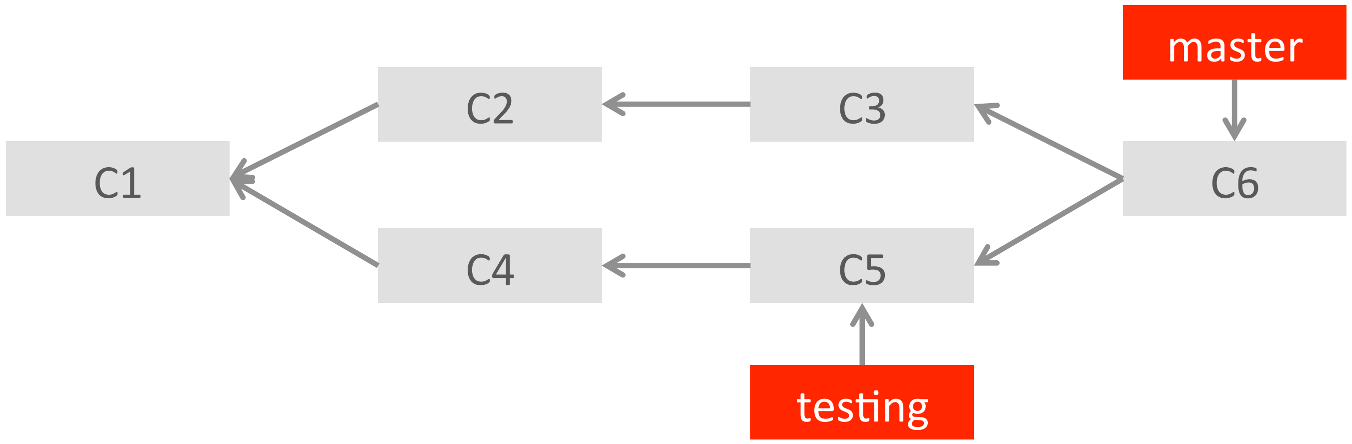
Strategy used if exists a direct path in history among the two branches

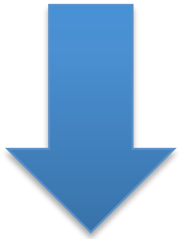
- History after merge



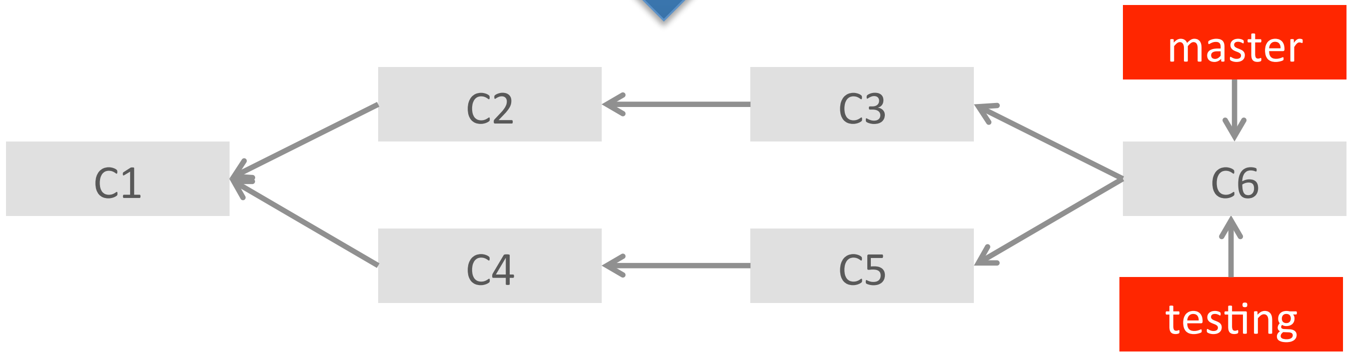
```
Marco@Marco-PC MINGW64 ~/project (testing)
$ git log --all --graph --decorate --pretty=oneline
*   070b51217089b3314e2e97006a5411ed289a1cae (HEAD -> testing, master) Merge branch 'testing'
|\
| * 306534715b1c14b3dab04b366581647392d429d7 work on testing
| * 200b2b6574422fadbd8419ccf59ac79af5a0439c added file3
* | 056bc3206fbb75ca3cf35ae2eaeb1828ff6b9753 work on master
* | ba6d384046b2411ae1964da9dc18ec42fcc3c40f added file2
|/
* 3281b640bfb8edfe2509b36fc2753293ca8f6073 first commit
```

Before merge…





…After merge



How to handle conflicts

- Previous merge operations were performed automatically by GIT:
  - No changes on different branches to the same file (no conflicts)

- Consider the following case:
  - Checkout master
  - Edit file page.html add it, commit
  - Checkout testing
  - Edit file page.html (same lines) add it, commit

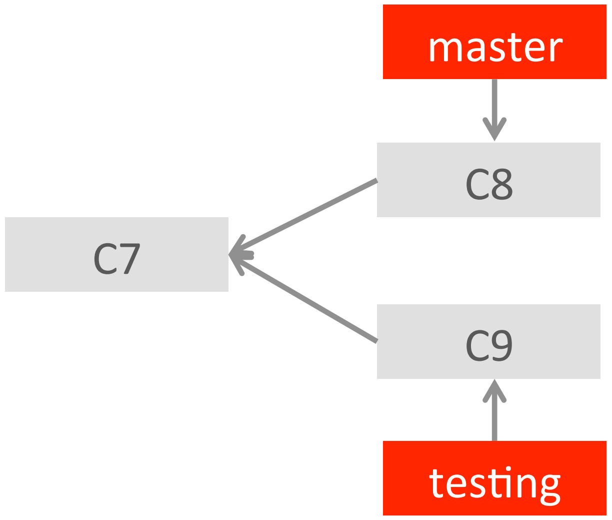
How to handle conflicts

page.html (C8 master)

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

page.html (C9 testing)

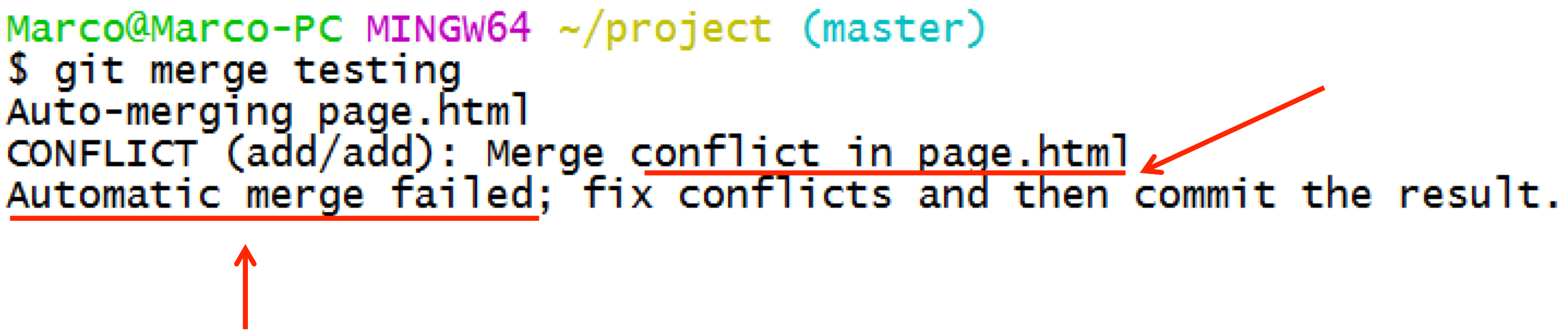
```
<html>
</head>
<head>
<body>
        <h1>Testing title</h1>
</body>
</html>
```



- What if we merge “testing” into “master”?

Merge with conflicts

- Try merging:
  - `git checkout master`
  - `git merge testing`

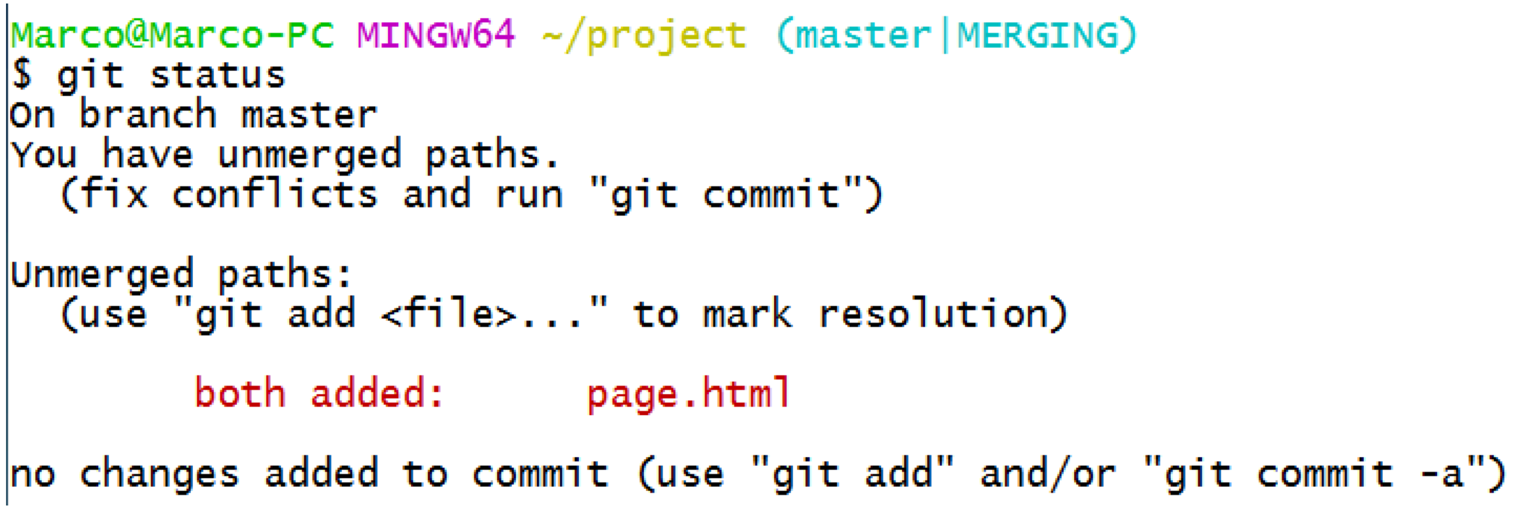


```
Marco@Marco-PC MINGW64 ~/project (master)
$ git merge testing
Auto-merging page.html
CONFLICT (add/add): Merge conflict in page.html
Automatic merge failed; fix conflicts and then commit the result.
```

POLITECNICO MILANO 1863

Merge with conflicts

- Git status:



```
Marco@Marco-PC MINGW64 ~/project (master|MERGING)
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")

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

        both added:      page.html

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

- We have to resolve merge conflicts and do a commit

Resolve conflicts

page.html

```
<html>
</head>
<head>
<body>
<<<<<<< HEAD
        <h1>Master title</h1>
=======
        <h1>Testing title</h1>
>>>>>>> testing
</body>
</html>
```

Current version (HEAD)

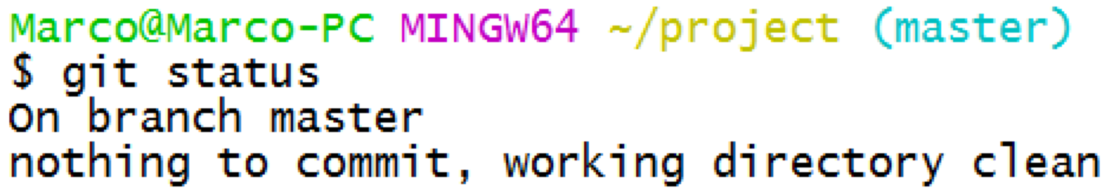
Version in the other branch (testing)

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

File fixed manually

Resolve conflicts

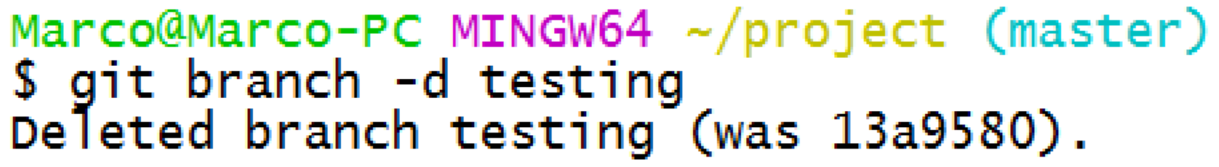
- After all the conflicted file have been manually fixed, commit them:
  - `git add page.html`
  - `git commit -m”solved conflicts: inserted correct title”`
- Git status:



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

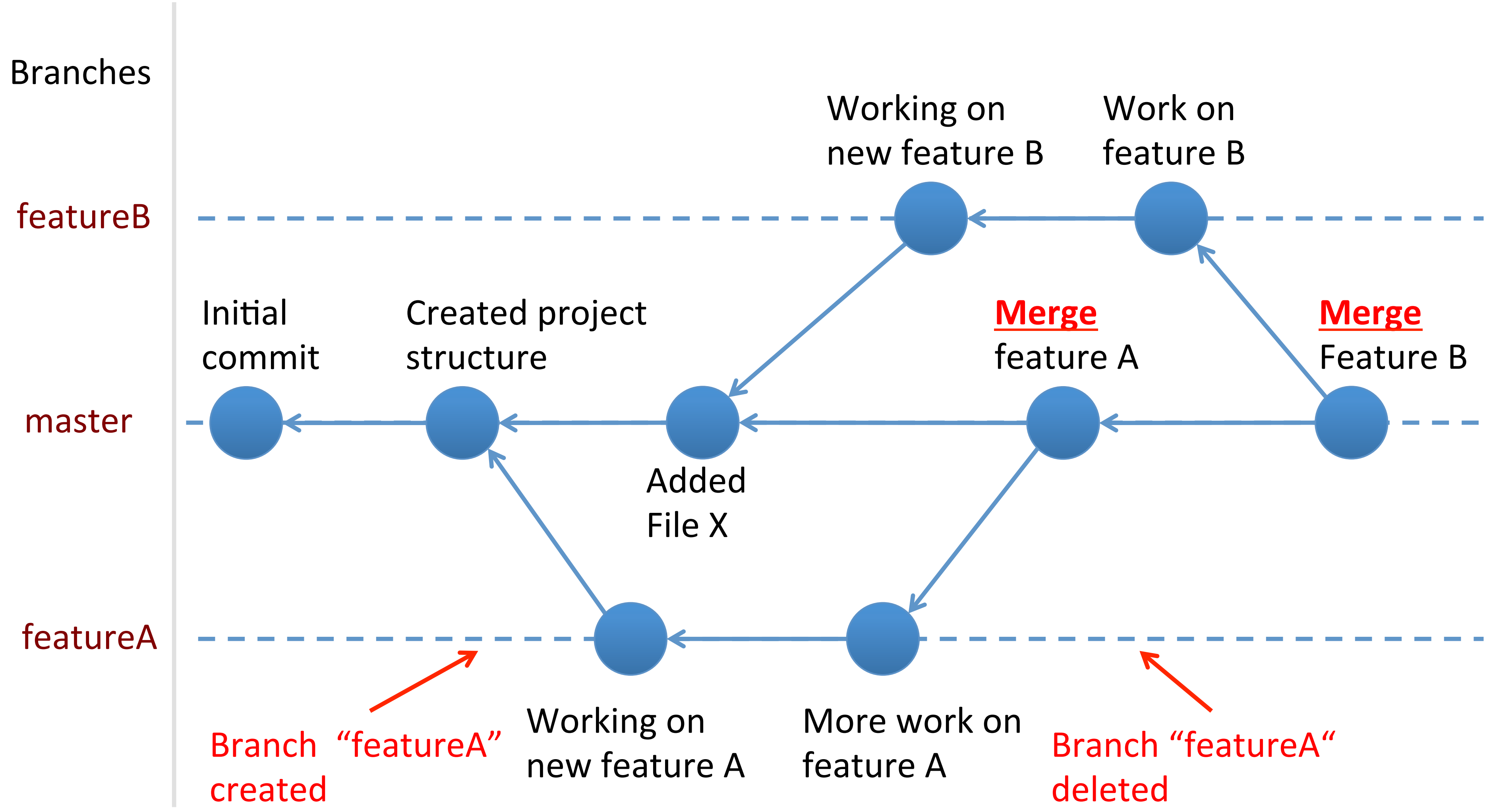
Deleting a branch

- After you are done with a branch and you have merged its content, the branch can be safely removed:
  - `git branch -d <branch name>`



```
Marco@Marco-PC MINGW64 ~/project (master)
$ git branch -d testing
Deleted branch testing (was 13a9580).
```

Branching usage example



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 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

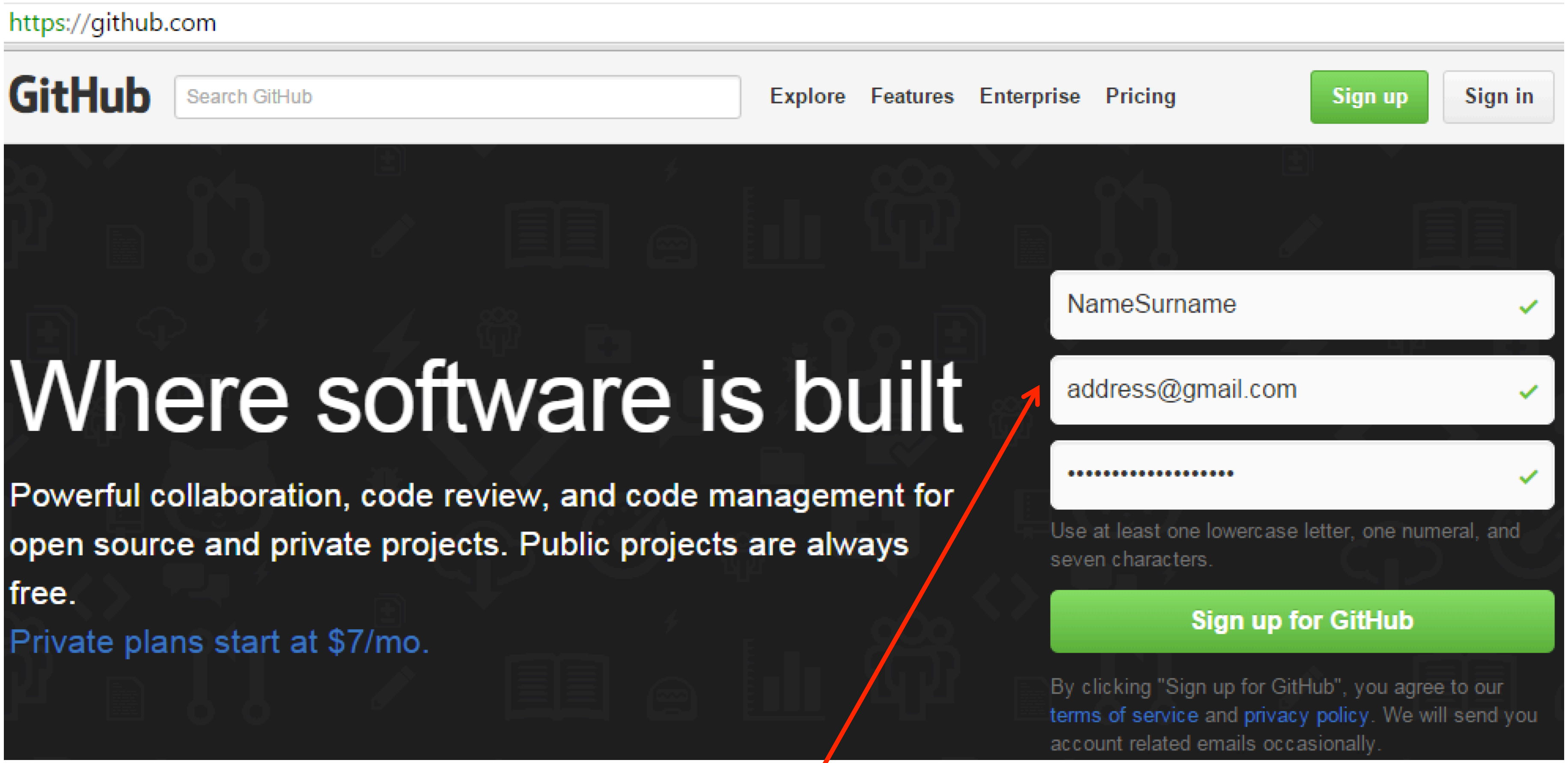
Bitbucket (https://bitbucket.com)
  – Public repositories for free
  – Private repositories for free
  – Number of users sharing private repositories limited to 5, pay for extra users

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

Info update at 14/09/2015

Create a GitHub account

Go to: https://github.com



https://github.com

GitHub | Search GitHub | Explore | Features | Enterprise | Pricing | Sign up | Sign in

Where software is built

Powerful collaboration, code review, and code management for open source and private projects. Public projects are always free.

Private plans start at $7/mo.

NameSurname ✓

address@gmail.com ✓

•••••••••••••••••••• ✓

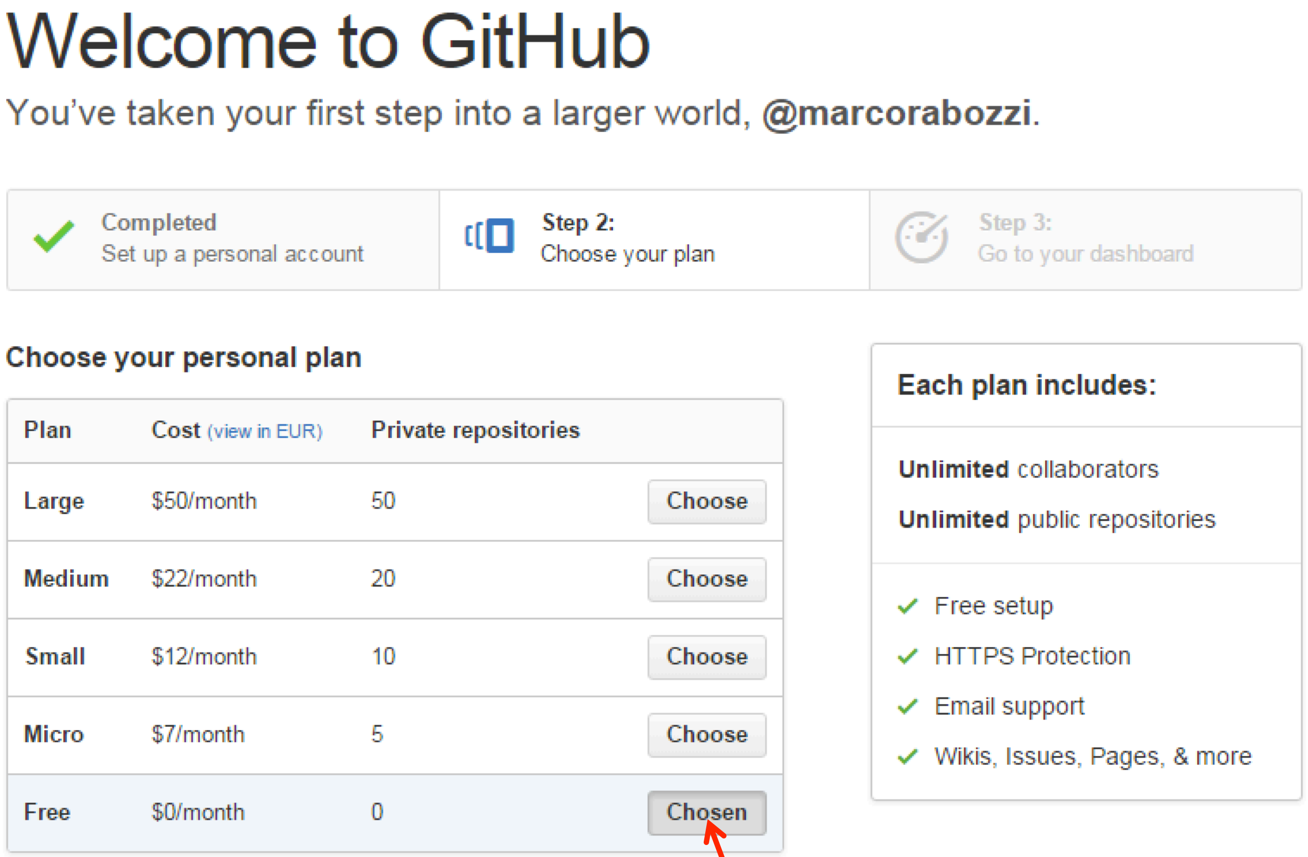
Use at least one lowercase letter, one numeral, and seven characters.

Sign up for GitHub

By clicking "Sign up for GitHub", you agree to our terms of service and privacy policy. We will send you account related emails occasionally.

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 Set up a personal account	**Step 2:** Choose your plan	Step 3: 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:

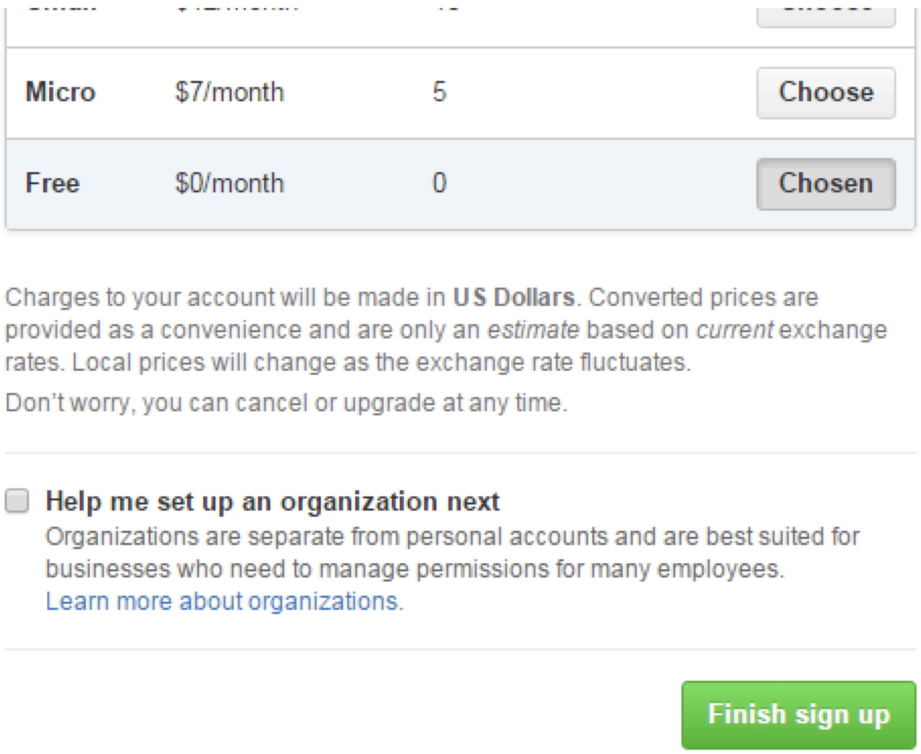
Unlimited collaborators

Unlimited public repositories

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

Select the free plan

Create a GitHub account



Micro	$7/month	5	Choose
Free	$0/month	0	Chosen

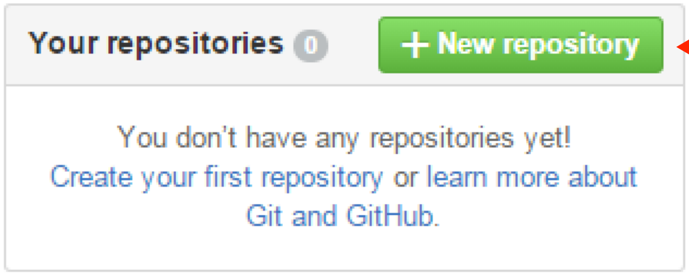
Charges to your account will be made in **US Dollars**. Converted prices are provided as a convenience and are only an *estimate* based on *current* exchange rates. Local prices will change as the exchange rate fluctuates.

Don't worry, you can cancel or upgrade at any time.

☐ **Help me set up an organization next**
Organizations are separate from personal accounts and are best suited for businesses who need to manage permissions for many employees.
Learn more about organizations.

Confirm account creation

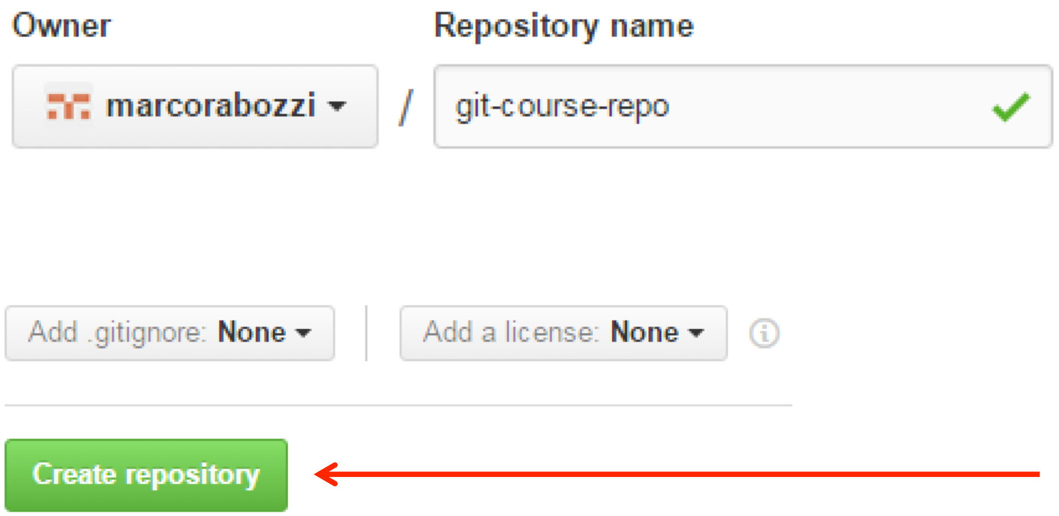
Create a remote repository



Your repositories 0 + New repository

You don't have any repositories yet!
Create your first repository or learn more about Git and GitHub.

Create a new repo



Owner: marcorabozzi / Repository name: git-course-repo

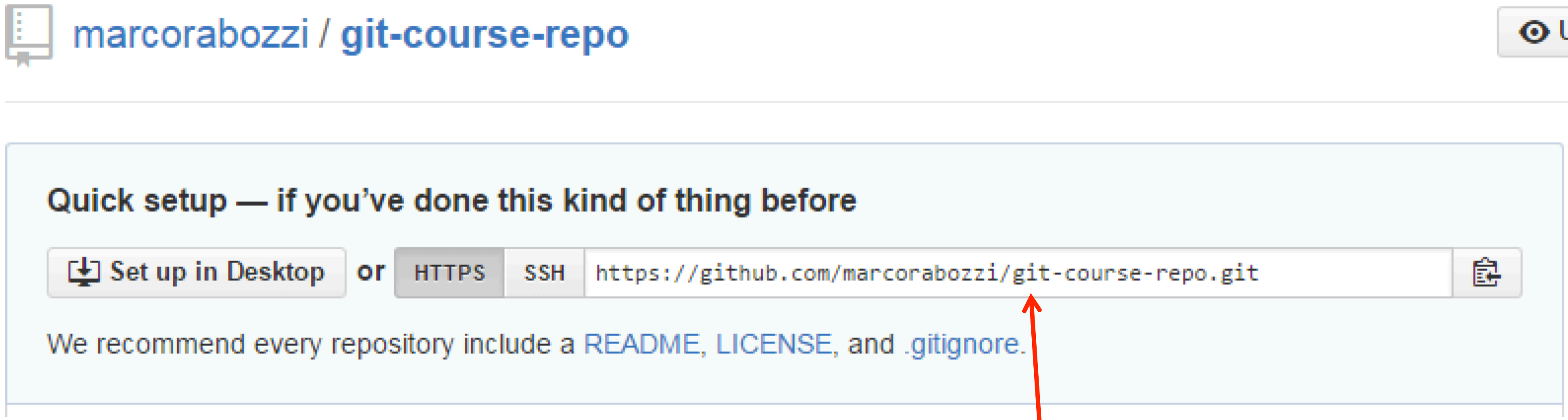
Add .gitignore: None | Add a license: None

Create repository

Name the repo as: git-course-repo

Use default options and press "create repository"

Create a remote repository



marcorabozzi / git-course-repo

Quick setup — if you've done this kind of thing before

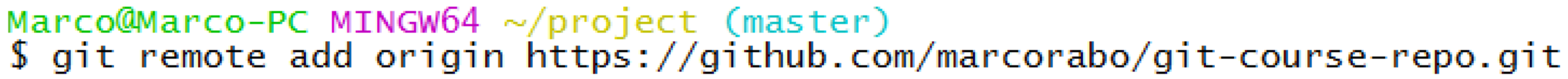
```
https://github.com/marcorabozzi/git-course-repo.git
```

We recommend every repository include a README, LICENSE, and .gitignore.

This URL represents your remote repository

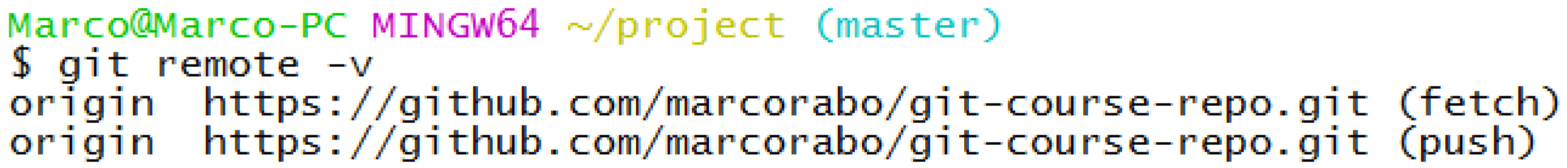
Initialize remote repo from local repo

- 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
```

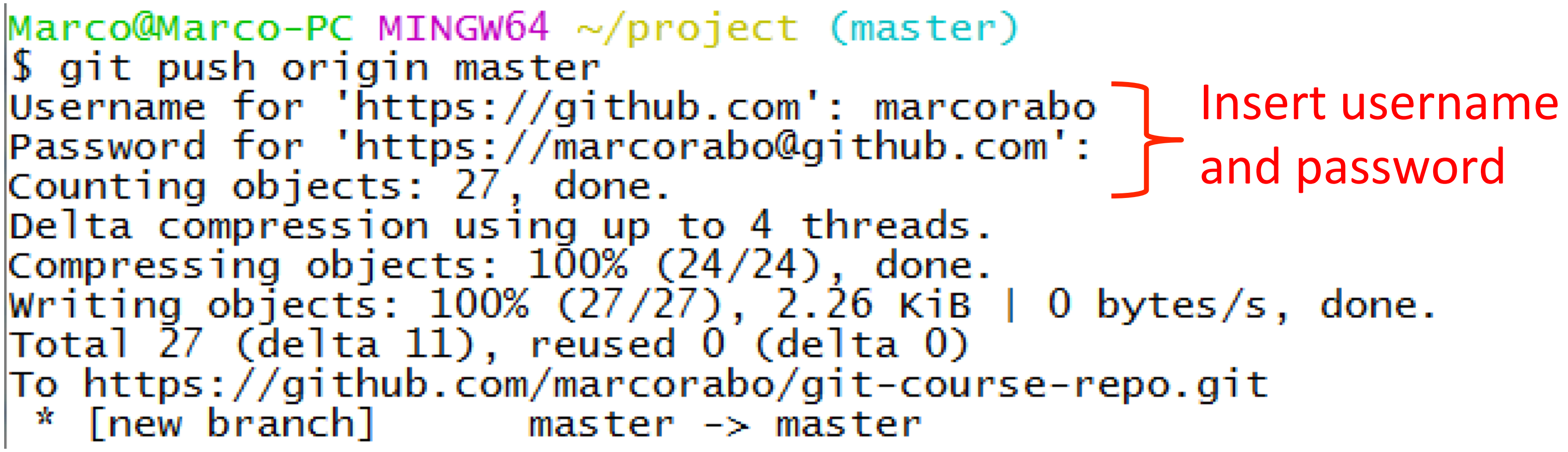
- Check that your remote has been added using command:
  - `git remote -v`



```
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 branches to 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>`



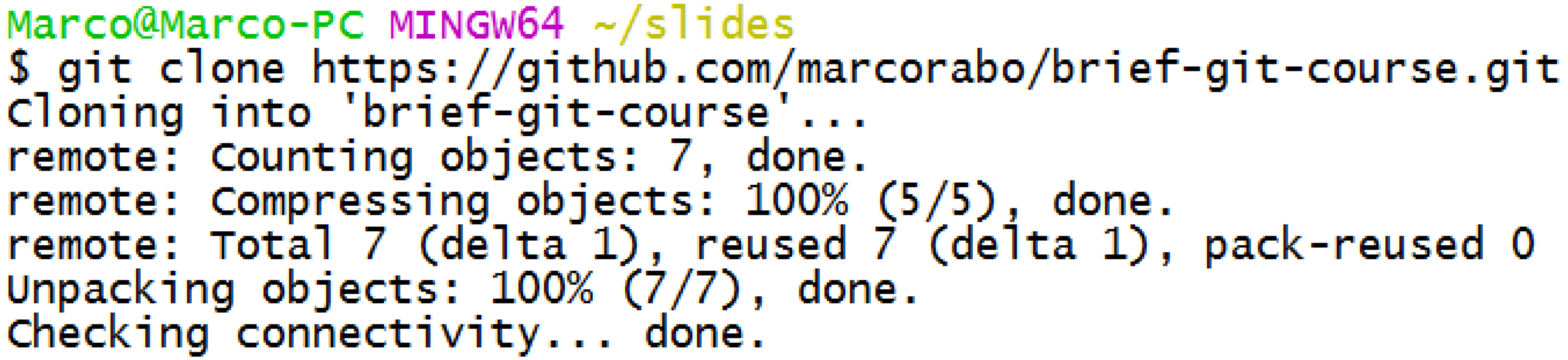
```
Marco@Marco-PC MINGW64 ~/project (master)
$ git push origin master
Username for 'https://github.com': marcorabo
Password for 'https://marcorabo@github.com':
Counting objects: 27, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (24/24), done.
Writing objects: 100% (27/27), 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
```

Push branches to 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 winstore`
  - On Linux / Mac OS X
    - `git config --global credential.helper cache`

Initialize local repo from remote repo

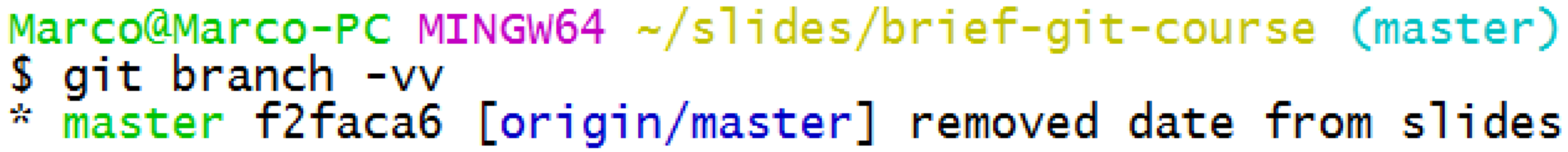
- If someone else has already created a repo for you and you want a copy to work on locally use:
  - `git clone <remote_url>`



```
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

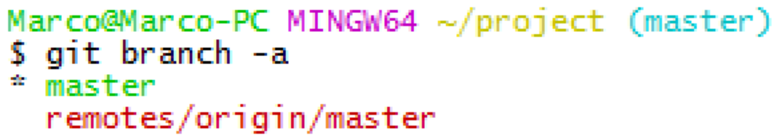
- Creates a copy of a remote repository into a folder in your current working directory
- Creates a local branch “master” and sets upstream for master -> origin/master



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

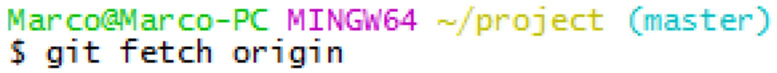
Remote branches

- Remote branches are shown locally as special branches that cannot be moved, to show all the branches (remote and local ones):
  - `git branch -a`



```
Marco@Marco-PC MINGW64 ~/project (master)
$ git branch -a
* master
  remotes/origin/master
```

- To "download" the status of all the branches from a remote:
  - `git fetch <remote_alias>`

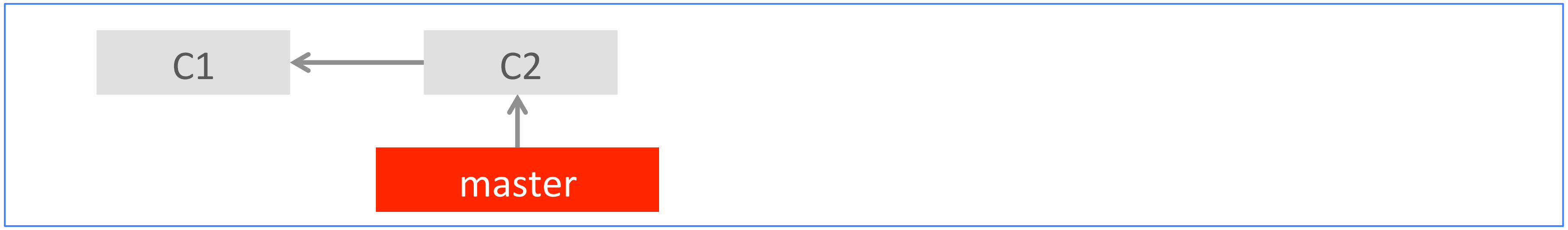


```
Marco@Marco-PC MINGW64 ~/project (master)
$ git fetch origin
```

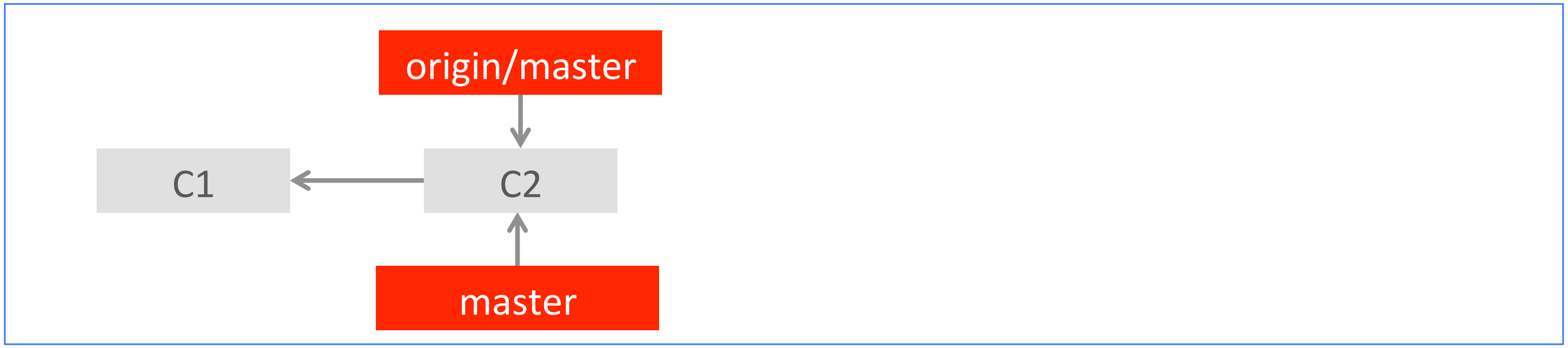
Working with remotes

- Assume the following situation

Remote server

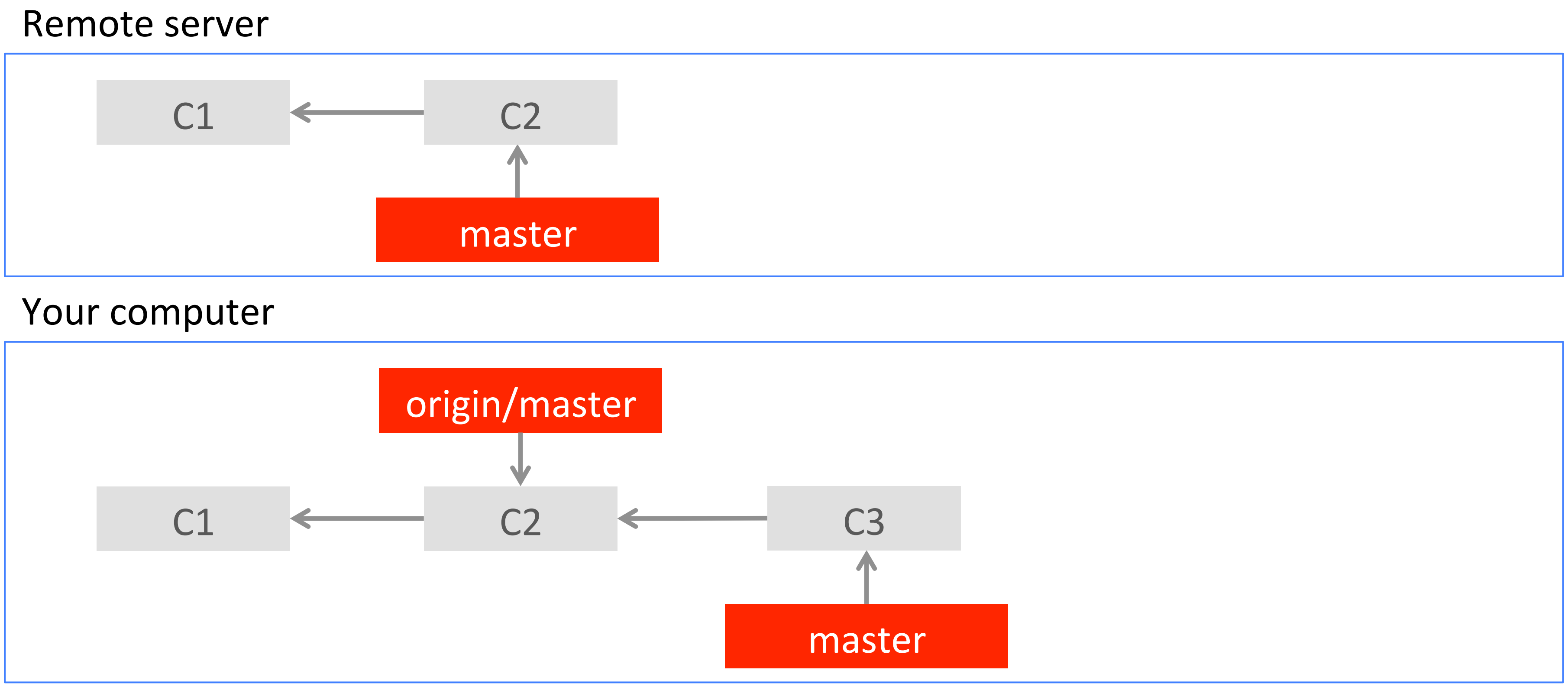


Your computer



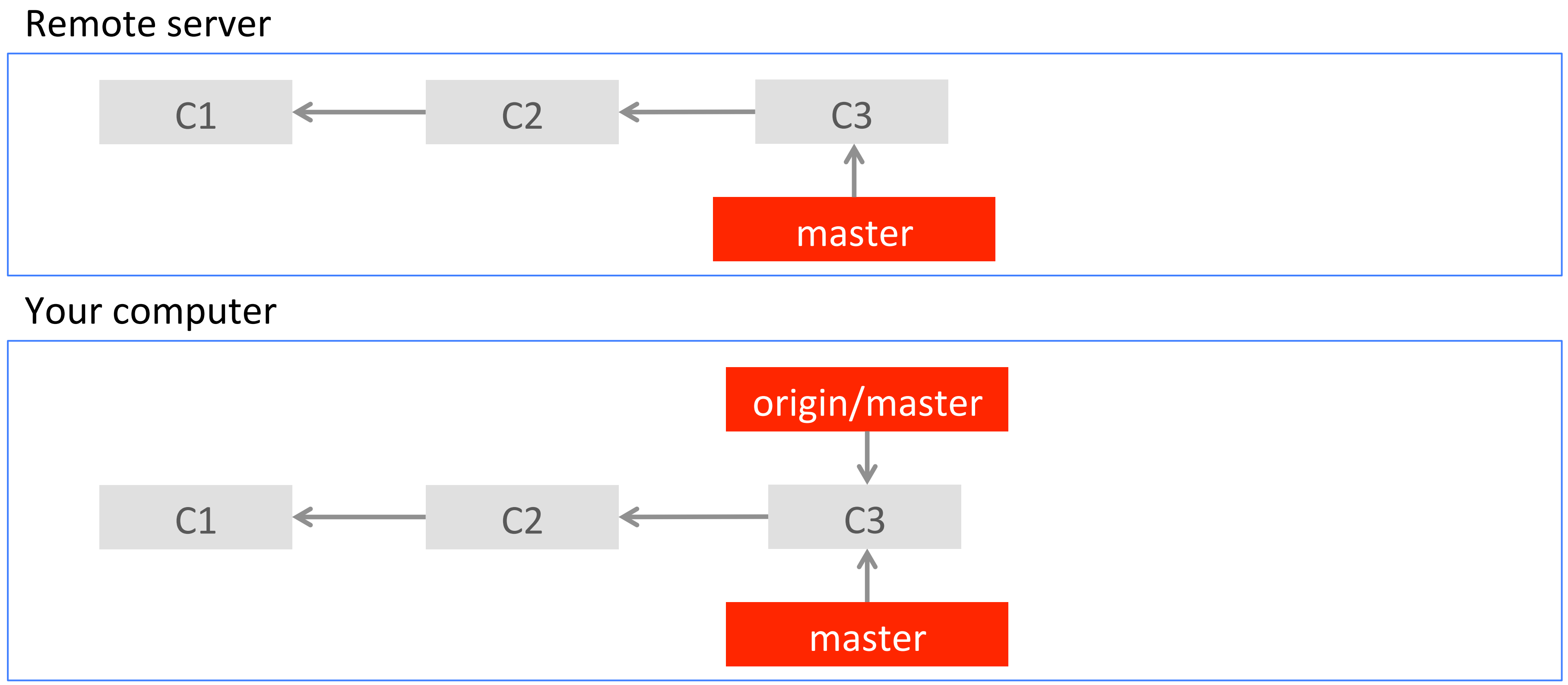
Working with remote

- Do a commit on your computer



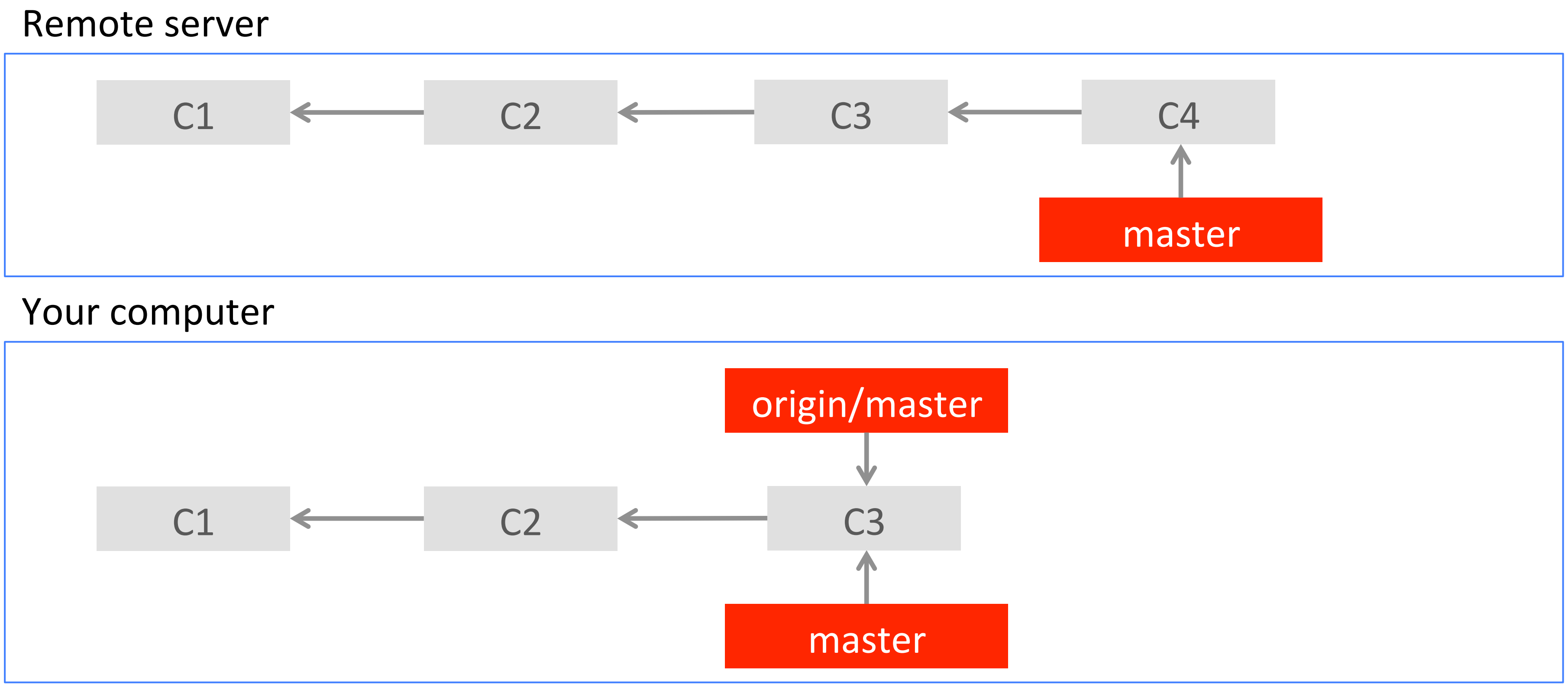
Working with remotes

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



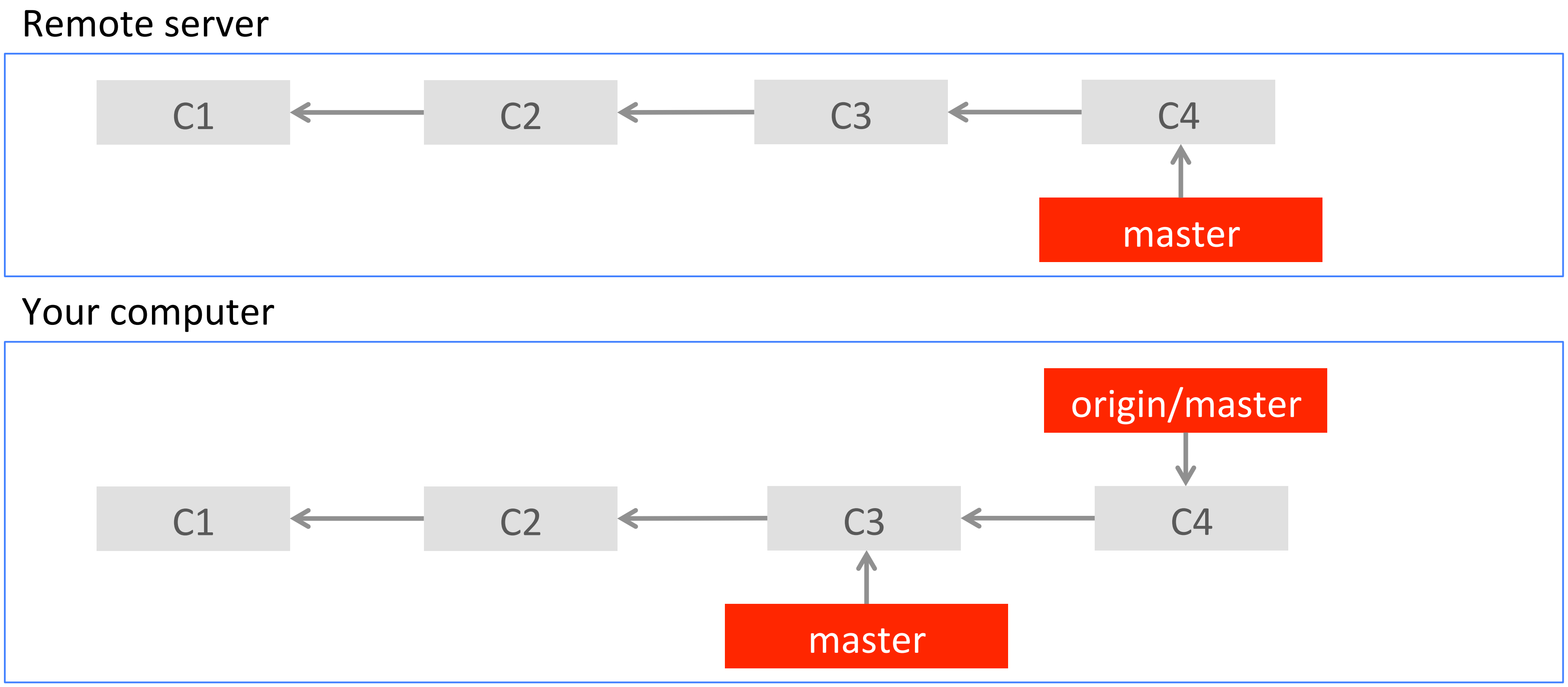
Working with remotes

- Someone else on your team pushes a commit



Working with remotes

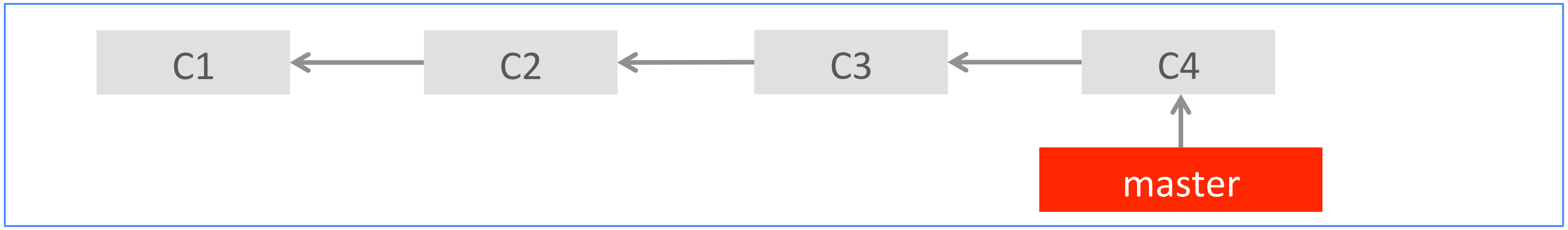
- Run `git fetch origin` to get updates from remote



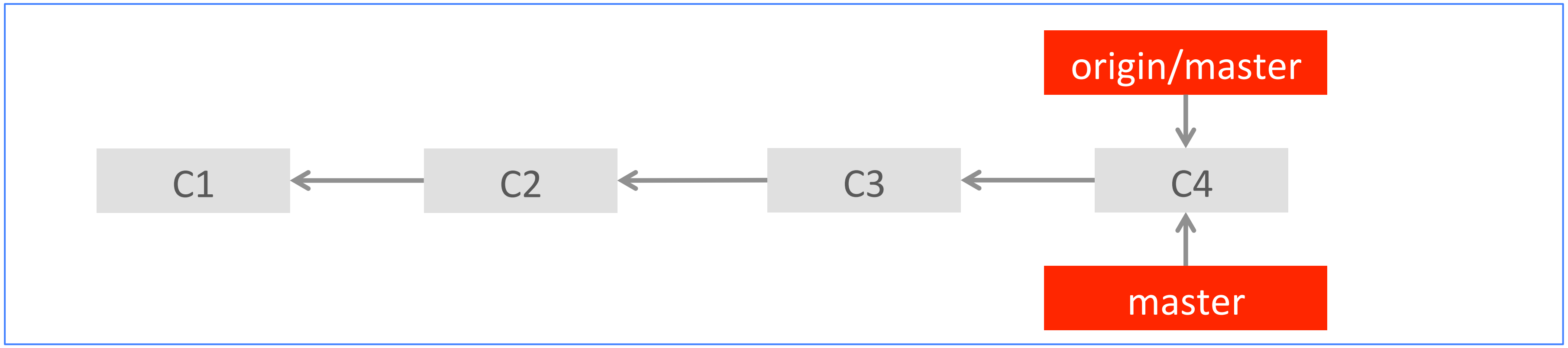
Working with remotes

- To synchronize your local branch: `git merge origin/master`

Remote server

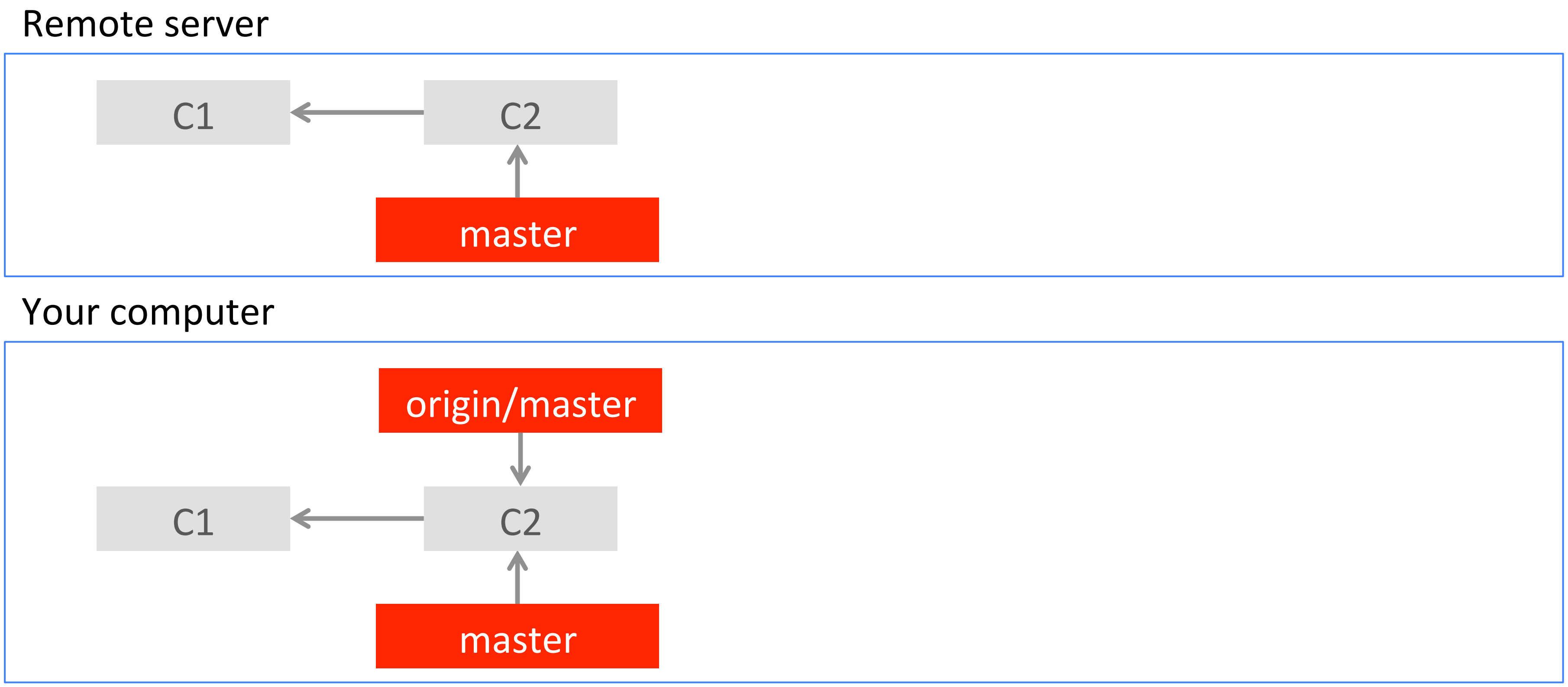


Your computer



Working with remotes (divergent work)

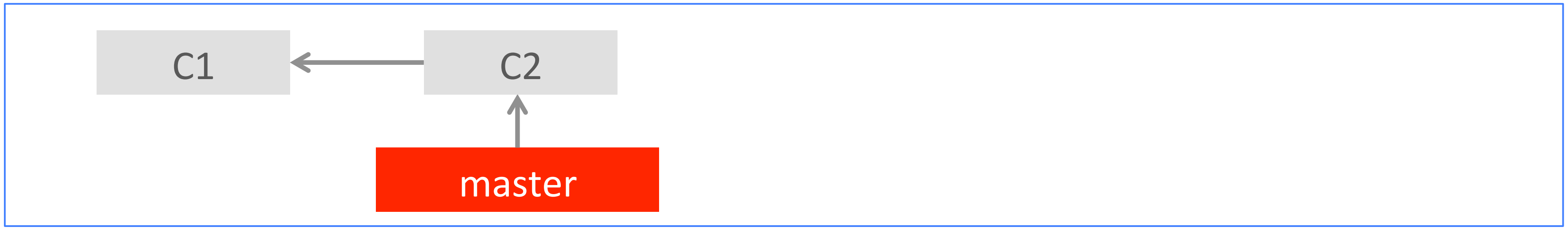
- Assume the following situation



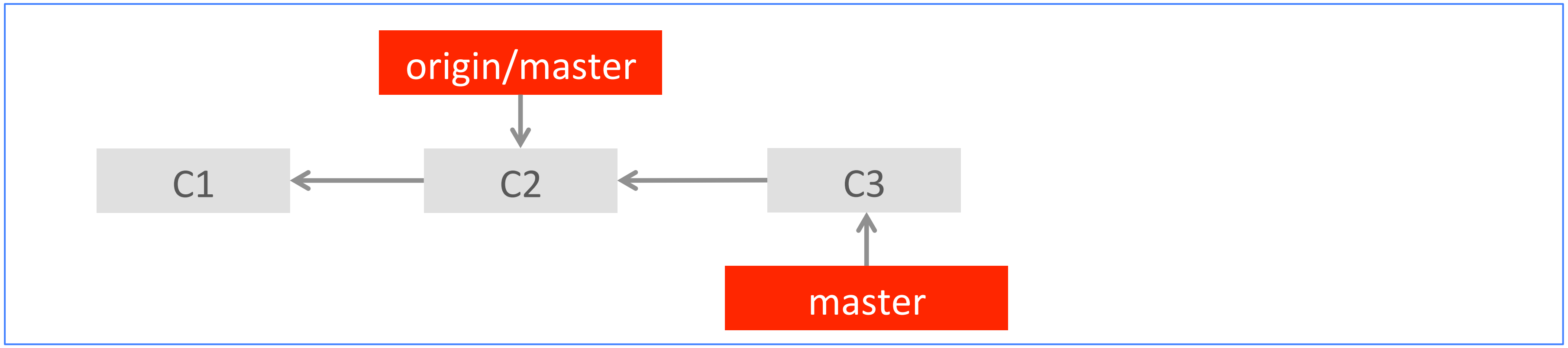
Working with remotes (divergent work)

- Do a commit on your computer

Remote server

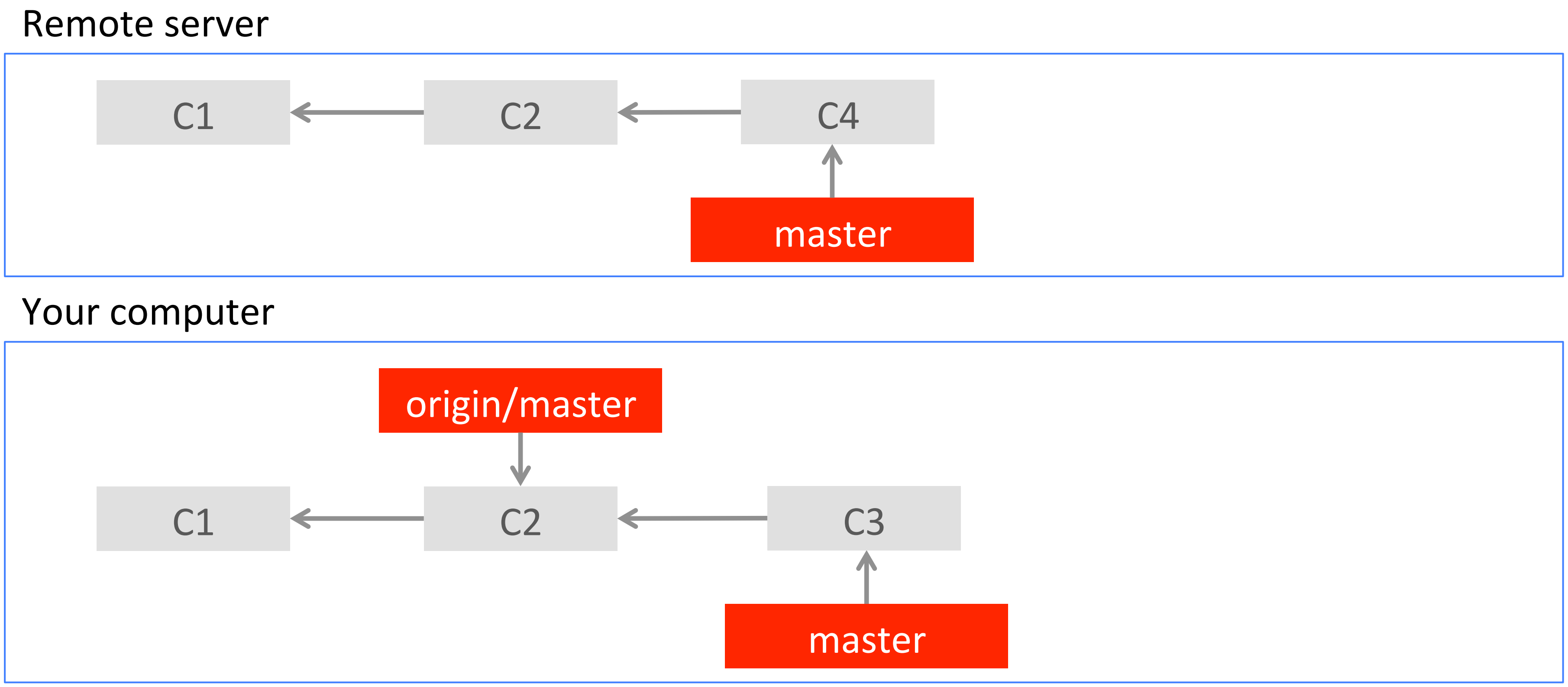


Your computer



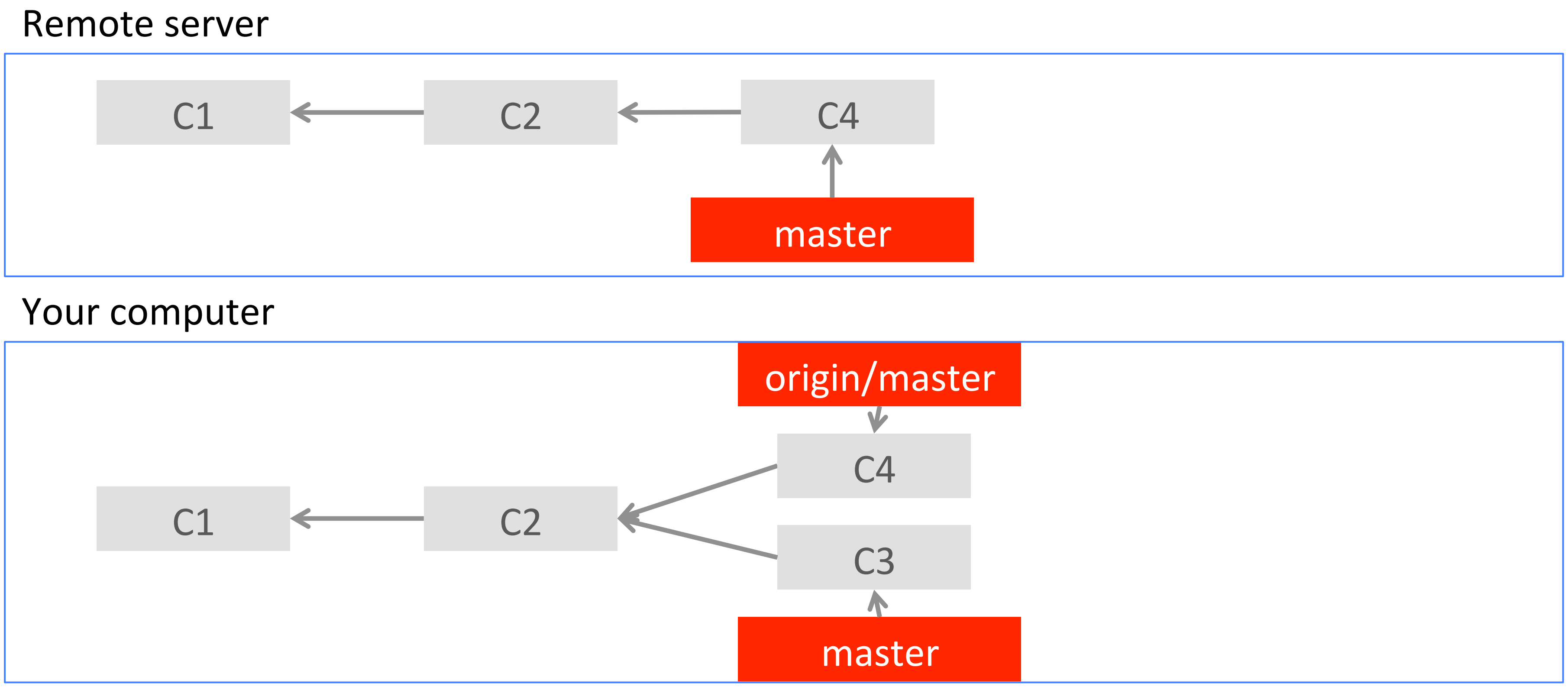
Working with remotes (divergent work)

- Someone else on your team pushes a commit



Working with remotes (divergent work)

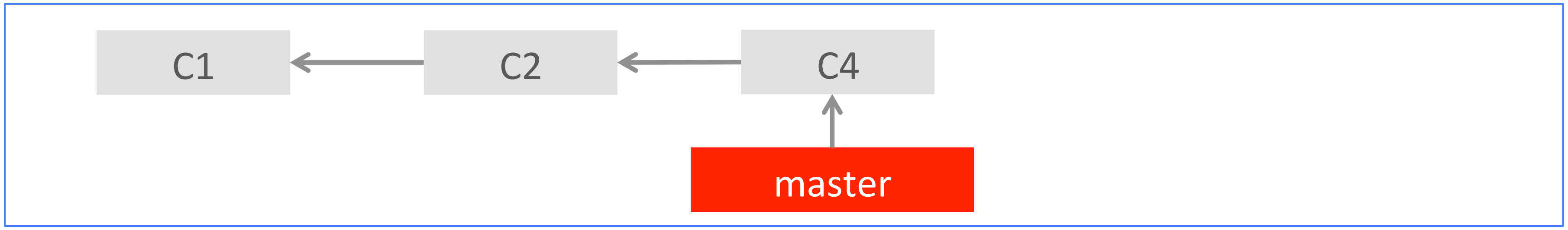
- Before doing a push you synchronize your repo using fetch



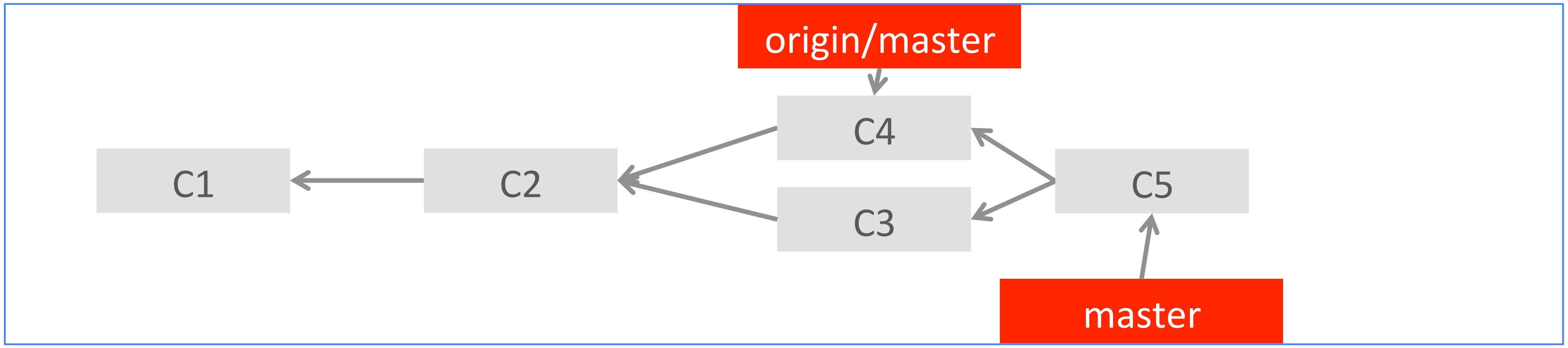
Working with remotes (divergent work)

- Now you need to merge your work with your colleague:
  `git merge origin/master`

Remote server



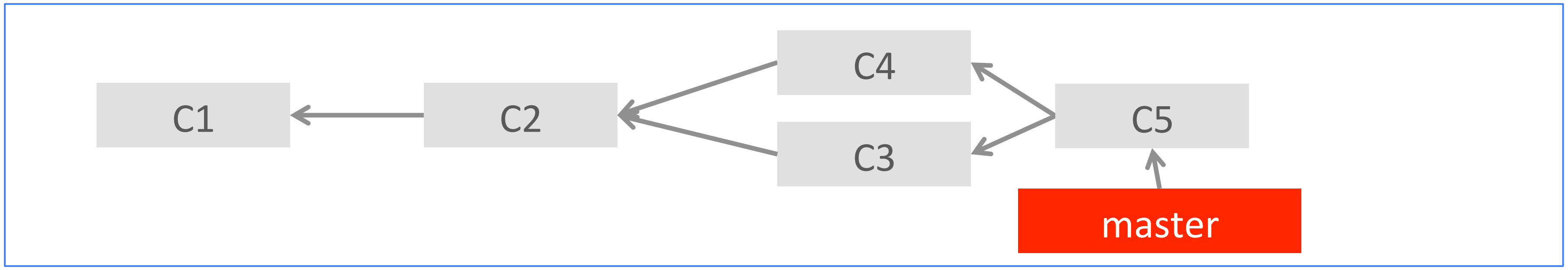
Your computer



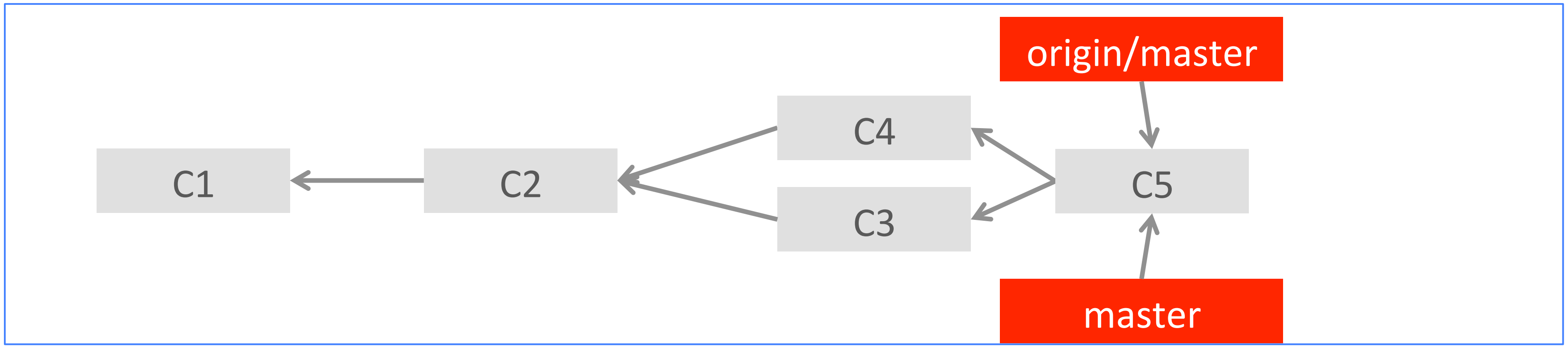
Working with remotes (divergent work)

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

Remote server



Your computer



Working with remotes simple flow

1. Fetch last changes from remote
2. Merge with your work
3. Work on your project
4. Commit changes
5. Push updates
6. If push fails:
    1. Fetch last changes
    2. Merge your work
    3. Push your work

Fetch followed by merge can often be replaced with: `git pull`

This can often be avoided if you work on different branches!

branches and remotes recap

- `git branch`
- `git branch –a`
- `git branch -vv`
- `git branch <branch name>`
- `git branch -d <branch name>`
- `git checkout <branch name>`
- `git log --graph --decorate --all --pretty=oneline`
- `git merge <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 fetch`
- `git pull`