

# Git & friends

---

Paolo Burgio  
paolo.burgio@unimore.it



**UNIMORE**  
UNIVERSITÀ DEGLI STUDI DI  
MODENA E REGGIO EMILIA

High Performance  
Real Time **Lab**



**They call me 007**

**0 LINES ADDED**

**0 LINES MODIFIED**

**7 MERGE CONFLICTS**



# What is a version control?

---

A system that keeps records of your changes

- › Allows you to revert any changes and go back to a previous state

Enables collaborative development

- › Allows you to know who made what changes and when (with a bug...)

...but not only this!!

- › Forces you to follow well-known development flows
  - (Ever heard of DevOps?)
- › Provides a set of tools to automate testing, integration and deployment
  - (Ever heard of CI/CD?)
- › Provides an easy way to write documentation

*Ultimately, let you focus on coding, coding, coding,  
removing all of (what programmers think is) clutter!*



# Git



- › “Widestly” adopted version control
- › Based on distributed repositories
- › Created by Linus Torvalds to support Linux kernel development, in 2005

**git** 🔊 LISTEN: US

UK: \* /'ɡɪt/ | US: (ɡɪt)

[definizione](#) | [Sinonimi inglesi](#) | [in spagnolo](#) | [in francese](#) | [Coniugatore \[IT\]](#) | [Conjugator \[EN\]](#) | [nel contesto](#) | [immagini](#)

WordReference English-Italiano Dictionary © 2022:

Principal Translations/Traduzioni principali		
<u>Inglese</u>		<u>Italiano</u>
<b>git</b> <i>n</i>	UK, pejorative, slang (contemptible man) (colloquiale, offensivo)	cretino, idiota, scemo <i>nm</i>
	George is such a git!	

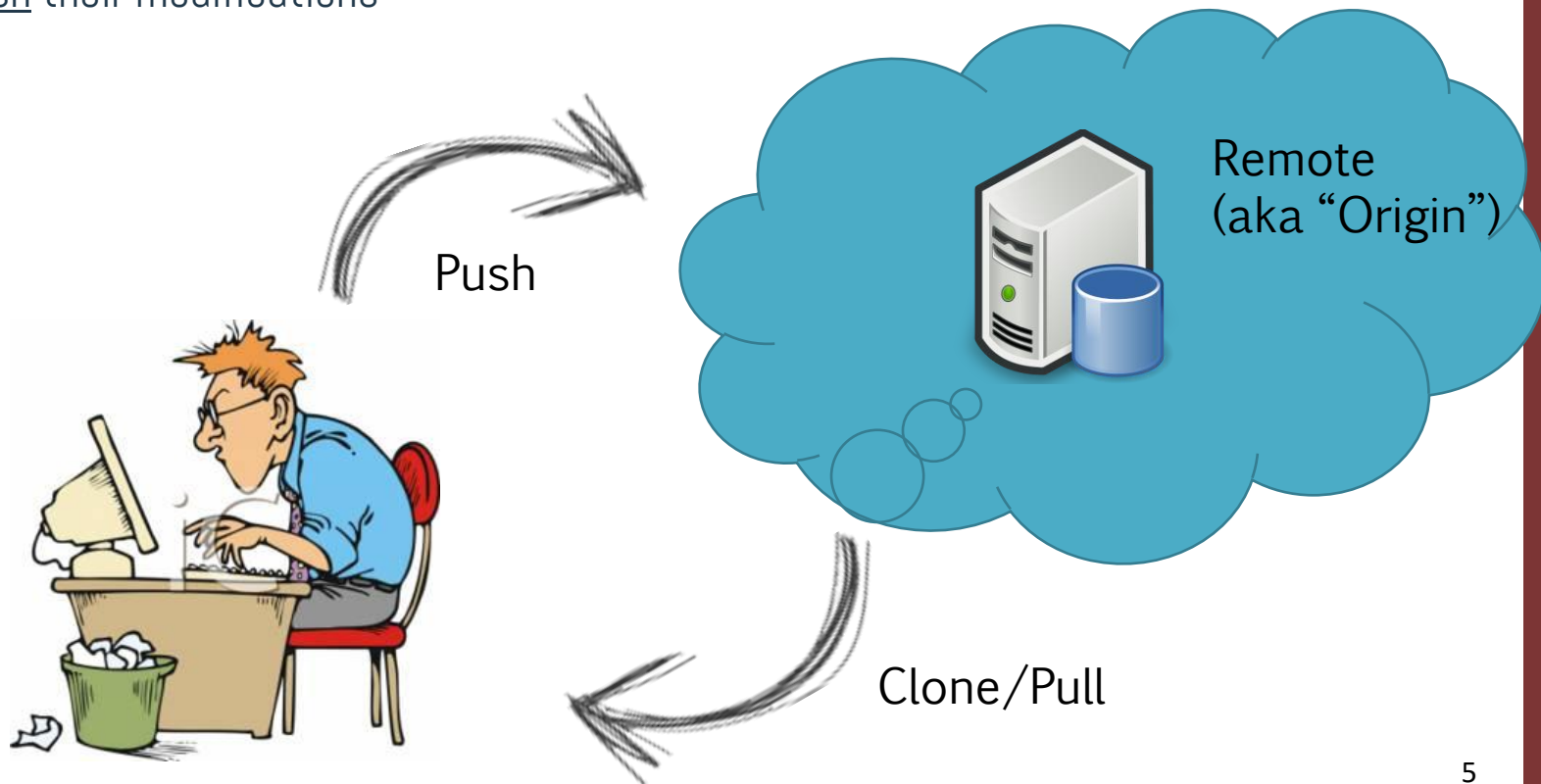


# Step 1: let's start simple

---

"Somewhere"/"on cloud" there is a remote repository with your codebase (called "origin")

- › Users clone this repo on their local machine
- › ...keep their local copy updated by pulling recent changes from the remote
- › And push their modifications





# <https://github.com>

---



The most famous public git repo service

- › Free version + payment version
- › Acquired by BigM in 2016 (tbc)
- › Web console to access

Why do we use this?

- › Preview for slides, past exams, code...
- › Issues!! You now are a team!
- › Let's set up an account

Why GitHub?

- › And not, for instance, GitLab.com?
- › ...or HiPeRT's on-premise GitLab?





# Local tools

---

Baseline: command line tool

- › Comes with most of the GNU/Linux distros
- › (You can always “apt” it)

Under Win, multiple options

- › <https://git-scm.com/> - also, with (very ugly) UI
- › Use WSL
- › Powershell?

Integrated in most commonly used IDEs

- › .but, soon, we'll only work on web tools



# Let's start!

---

Do the following

- › Create a GitHub account
- › Navigate to HiPeRT Lab's page
  - <https://github.com/HiPeRT>
  - <https://github.com/pburgio> for older stuff
- › Clone the repo
  - We will use git over https, but there are also other protocols

```
$ git clone https://<URL>
```

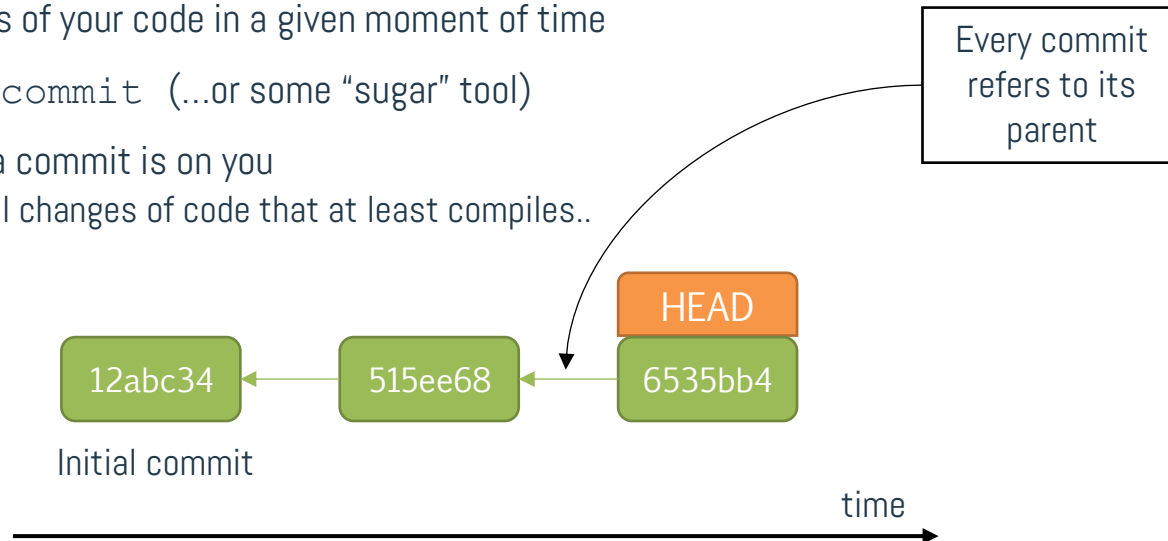




# Basic workflow

A project is a sequence of commits

- › They are snapshots of your code in a given moment of time
- › Create with `git commit` (...or some "sugar" tool)
- › The granularity of a commit is on you
  - Typically, small changes of code that at least compiles..



Commits track incremental changes! (`git diff`)

- › Every commit is identified by a hashcode, and has a parent
- › Most recent (...) commit is also called HEAD
- › It is **mandatory** to add a comment to every commit

\$ `git log` to see all information





# The git flow

---

## Working with commits

- › Before committing, files must be added to the staging area

```
$ git add <file> <file> && git commit
```

..or...

```
$ git commit -a
```

## To check the status of your staging area and commits

```
$ git status
```

## Watch out

- › -a options does not apply to new files
- › Empty folders are never added by git
- › You can always amend a previous commit, if you forgot to add something

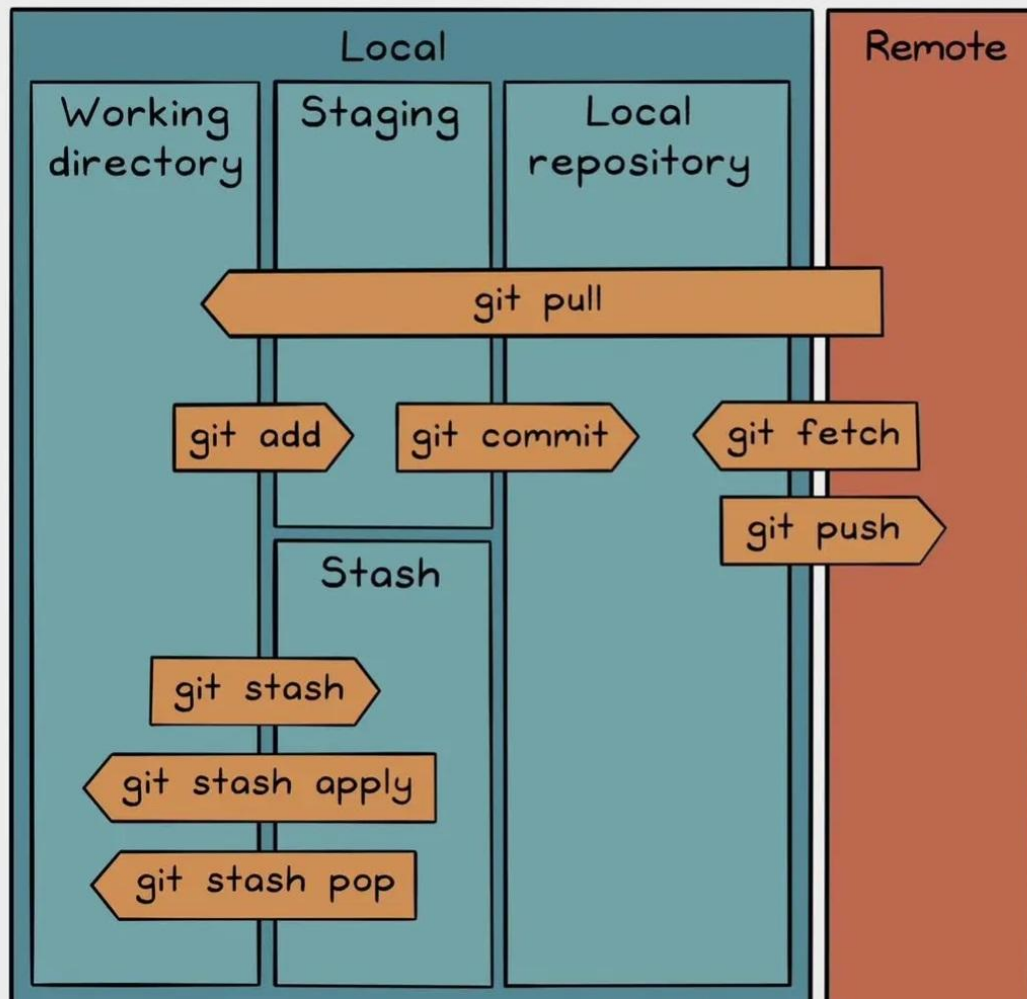
```
$ git commit --amend
```



# The git flow (cont'd)

## Git workflow

by nikkiandchris.io





# The git flow (cont'd)

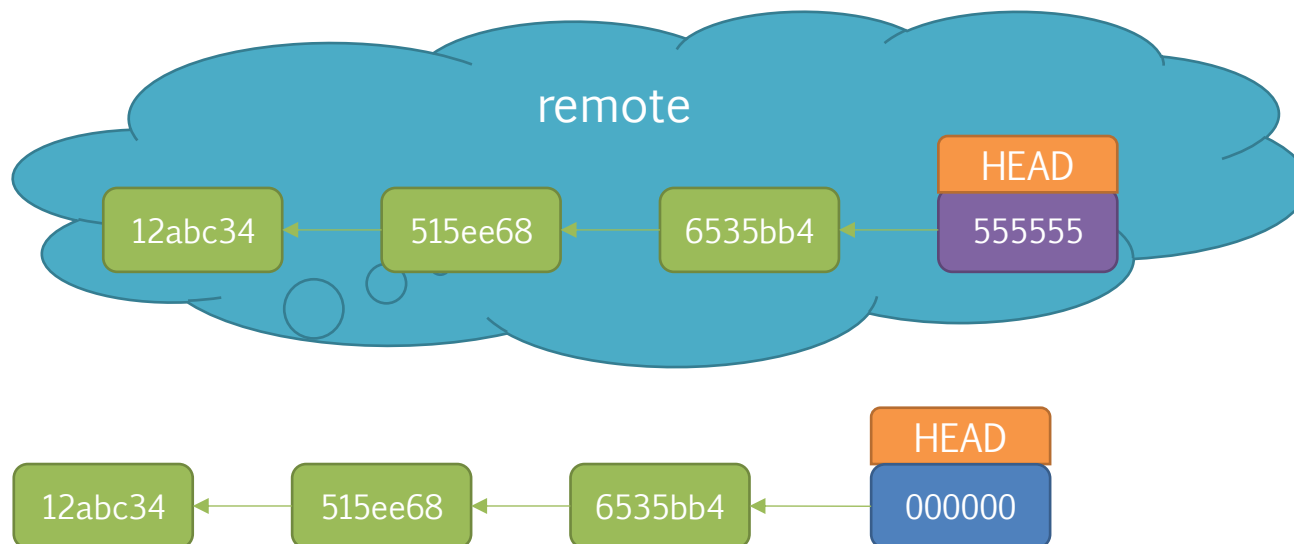
- › You can remove/delete files
- › Renaming files means deleting + adding

When you're done, transfer files to the remote repo by

```
$ git push [origin] [master]
```

Always make sure you have the latest version!

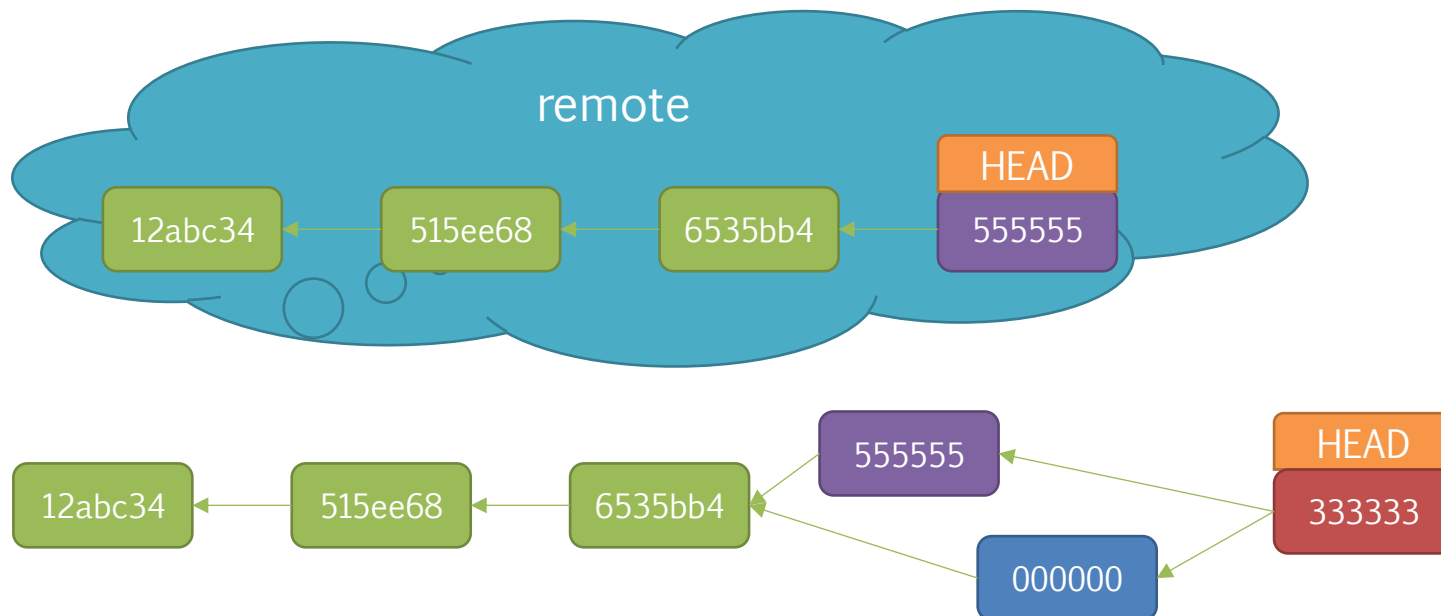
```
$ git pull [origin] [master]
```





# Pulling and auto-merging

- › Based on the actual time of commit, local history is updated
- › The local codebase is automatically updated (aka: files are merged), including modifications both from local and remote
- › A new commit is created



Let's assume this is the most recent of the two



# Merge...and conflicts

---



- › It is extremely easy to mess things up, if codebases are significantly different!!!
- › Git works at the code line level
  - What if we modify the same code line?
  - Some modern systems track also single words

If there are merge conflicts, the local repo stays in conflicting state

- › Until you solve the conflicts locally, and manually merge them
  - Easier if you do it on IDEs
  - Using appropriate flags in the checkout command

## Tips & tricks

- › Frequently pull changes
- › Always make sure your code works by re-running the testing automation
- › Create small commits, “packed” by “working area” (ex: one for the code, one for the makefiles, one for the scripts...)
  - This also forces you to keep the workspace clean and well structured!!



# Useful commands

---

Checkout and (hard or soft) reset

- › To unstage/delete local modifications and/or commits

Revert

- › To switch back to a specific commit

Cherry-pick

- › Commits are incremental. They simply trace the difference against parent commit
- › You can apply a commit/difference also to any other commit (not only parent)
- › By cherry-picking



# The history of my project

---

A typical sw project is structured in quite a rigid way

- › A main branch ("master"), containing the latest released version (with full commit history)
- › Multiple branches that correspond to specific works/subprojects
- › You have total freedom on branches. Company-specific rules apply here
  - "develop", "bugfix/", "hotfix/", "features/", "pb\_<SOMETHING>" (initials of the developer)

Once a project is started, you might **never** push onto the master branch

- › You typically fork the master, or the "Development" branch, and then issue a pull request
- › Which is served by the repo Maintainer
  - Typically a sadist, with very limited empathy and sense of humour
  - Sysadmins are good candidates for this role
- › There are access rules and user roles, both at the org and repo level, and also at the branch level, etc





# Torvalds is a nice person....

---

```
On Sun, Sep 18, 2011 at 1:35 PM, Eric Dumazet <eric.dumazet@gmail.com> wrote:
> [PATCH] tcp: fix build error if !CONFIG_SYN_COOKIE
> commit 946cedccbd7387 (tcp: Change possible SYN flooding
> messages) added a build error if CONFIG_SYN_COOKIE=n
```

**Christ Eric, you clearly didn't even compile-test this one either.**

Which is pretty bad, considering that the whole and only *\*point\** of the patch is to make it compile.

The config option is CONFIG\_SYN\_COOKIES (with an 'S' at the end), but your patch has 'CONFIG\_SYN\_COOKIE' (without the S).

Which means that now it doesn't compile when syncookies are *\*enabled\**. I really wanted to release -rc7 today. But no way am I applying these kinds of totally untested patches. Can you guys please get your act Together?

PLEASE?

Stop with the "this might just work" crap. Because -rc7 is just too late to dick around like that.

Linus





# ...sometimes he really is

---

```
On Thu, Aug 25, 2011 at 1:21 PM, Arnaud Lacombe  
<lacombar@gmail.com> wrote:  
> On Thu, Aug 25, 2011 at 4:10 PM, Andy Lutomirski  
<luto@mit.edu> wrote:  
>>  
>> Arnaud, can you test this?  
>>  
> All good.  
>  
> Tested-by: Arnaud Lacombe <lacombar@gmail.com>
```

Thanks guys. Applied and pushed out,

Linus





# What typically happens...

- › Developers clone a branch from an updated repo
  - Typically, you clone Master, or Develop
- › Anyway: your starting point

```
$ git clone <SOME_URL>
```

```
$ git checkout Develop # Assume it exists
```

```
$ git branch MY_LOCAL # Create from the Develop branch
```

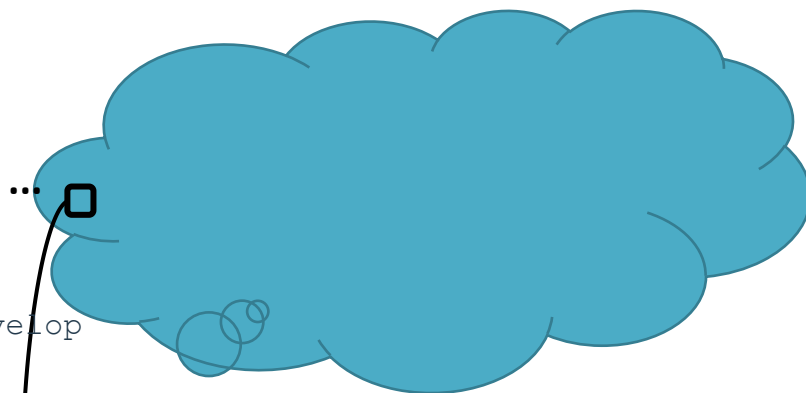
```
$ git branch -l # To check
```

```
$ git checkout MY_LOCAL # Switch to it
```

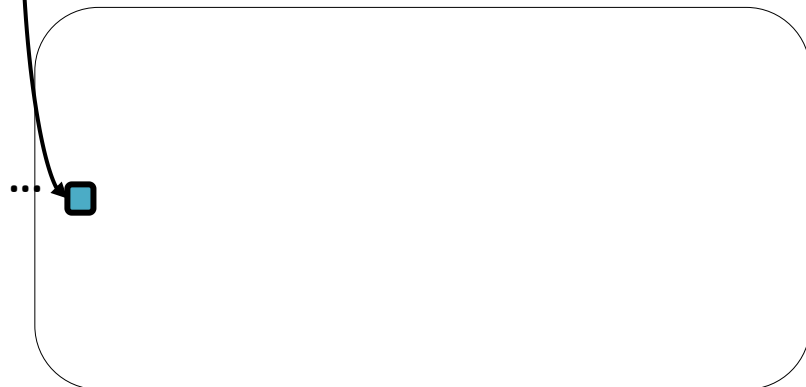
You can also do all of these together

```
$ git checkout -b MY_LOCAL
```

**Maintainer**



**Developer**



■ Master, or Develop branch

■ MY\_LOCAL branch





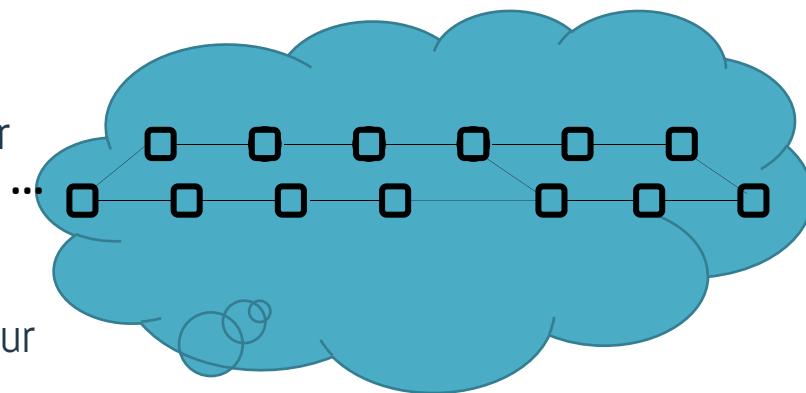
# What typically happens...

- › Developer starts his work, producing new commits
- › In a local branch
- › Similarly, remote repo gets some updates for some reason

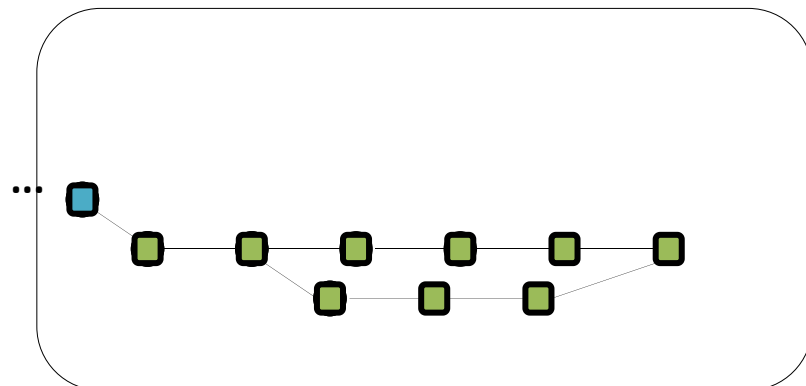
(You can push your local branch to back up your work on the cloud)

```
$ git push origin MY_LOCAL
```

**Maintainer**



**Developer**



■ Develop branch

■ MY\_LOCAL branch





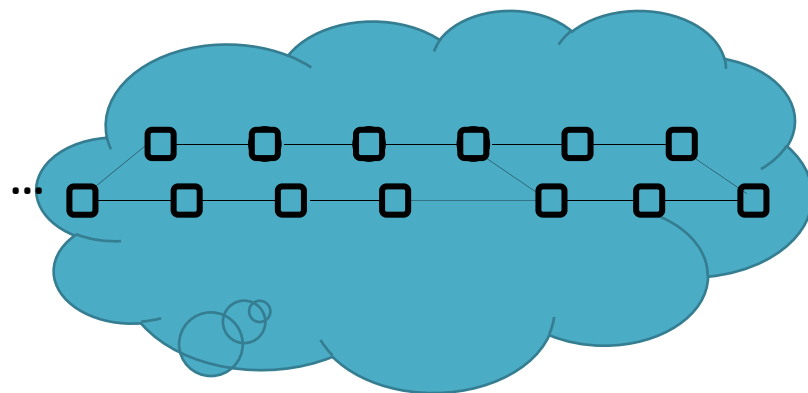
# What typically happens...

- › When developer is ready, he pulls to update with whatever is in the cloud
- › He's in charge of "making his commits consistent" with the whole story
- › This implies re-testing the whole thing!!

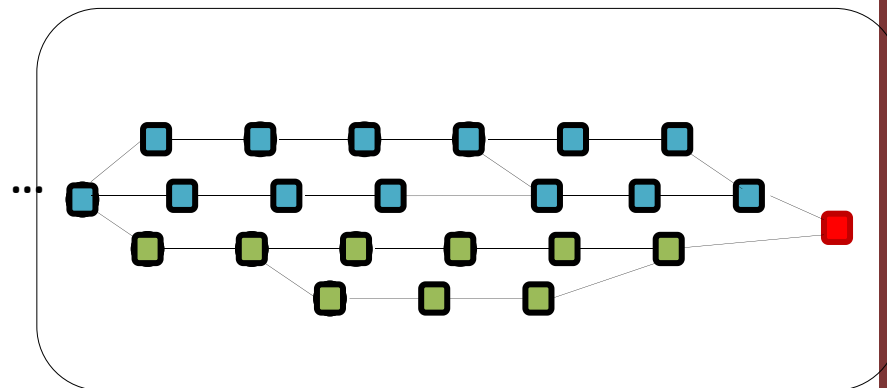
```
$ git pull origin Develop
```

```
$ git merge Develop
```

**Maintainer**



**Developer**



■ Develop branch

■ MY\_LOCAL branch



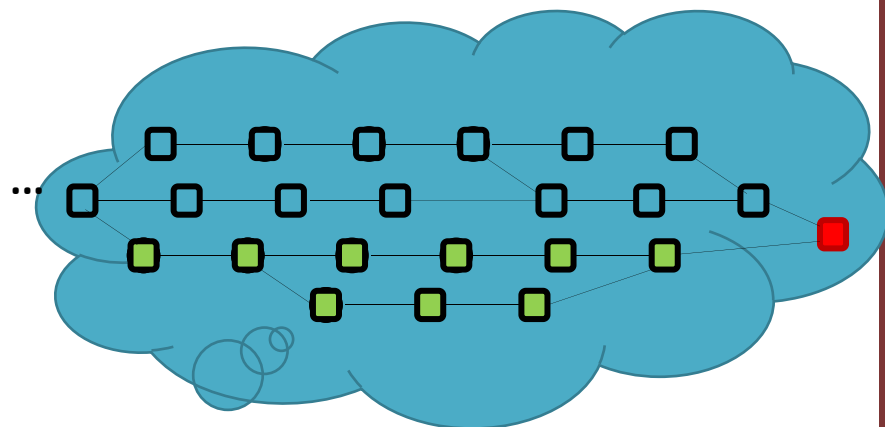
# What typically happens...

- › After the merge is done, the “final commit” is created
- › And then we can push on the cloud
- › And issue a pull request
  - By email, or by automated tools

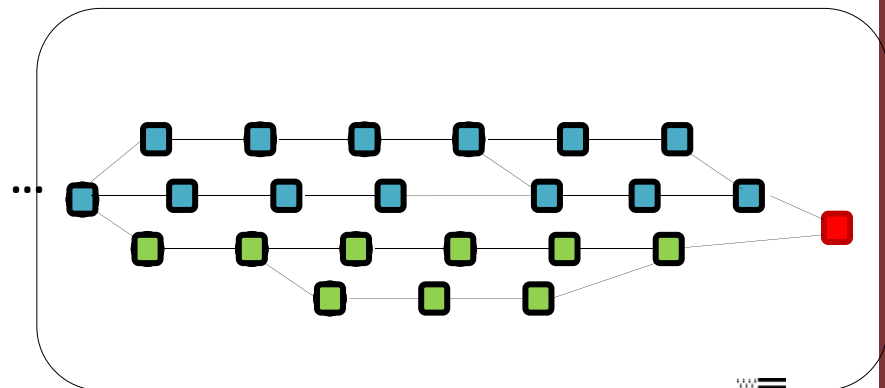
```
$ git push origin MY_LOCAL
```

(Remember, you cannot push Master)

**Maintainer**



**Developer**



□ Develop branch

□ MY\_LOCAL branch

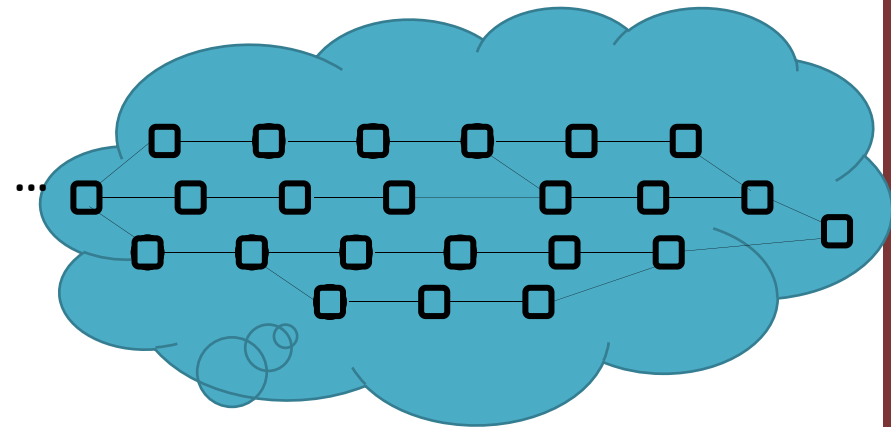




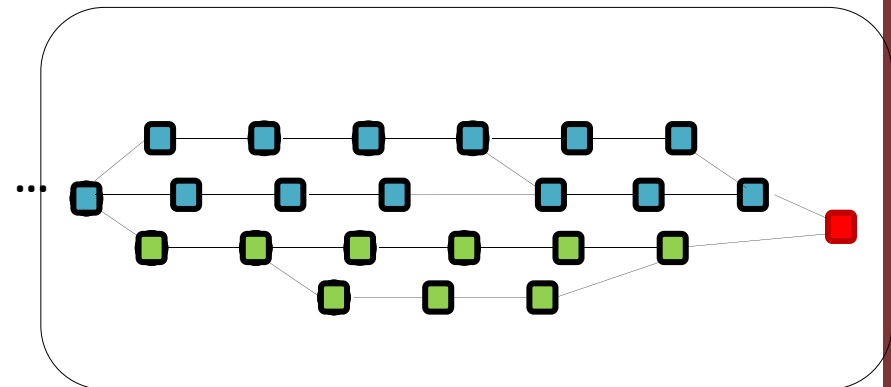
# What typically happens...

- › Request is accepted, and modifications apply to the Develop branch
- › Hopefully

**Maintainer**



**Developer**



■ Develop branch

■ MY\_LOCAL branch





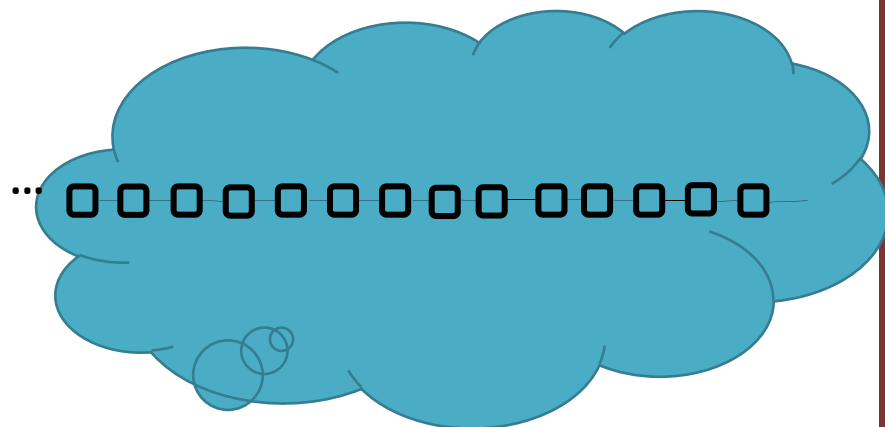
# What typically happens...

- › Typically, the “other” branch is then deleted remotely by maintainer
- › The story is now consistent, and a “unique” timeline exist
- › We need to pull the Develop branch one more time, to make my local copy consistent (and delete local branch

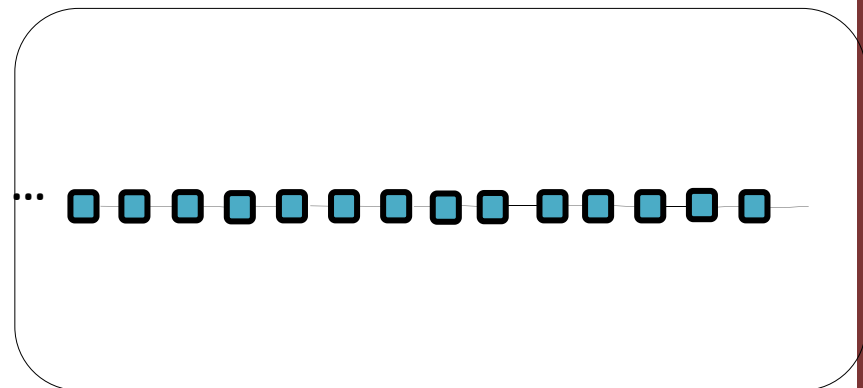
```
$ git pull origin Develop
```

```
$ git branch -d[D] MY_LOCAL
```

## Maintainer



## Developer



■ Develop branch

■ MY\_LOCAL branch







## Step 2: multiple repos

- › You can add as many remote repos you want
- › In Git philosophy, all repos are equal!
- › At least, from the tech viewpoint
- › Each of them has a maintainer

Origin (created by default when you clone)



```
$ git remote -v # To list them
```

```
$ git remote add <name> <url>
```



repo1



pburgio



# Ignore files



Some folders and files are not useful for your projects!

- › `.vscode`
- › `ROS2:install/ build/ log/ ...`

So, why adding them to the repo?

Add `.gitignore` file in your repo (or in some subfolder)..

- › Specify files to ignore
- › Can use wildcards
- › Comments start with `#`

Remember to commit your  
`.gitignore` file!

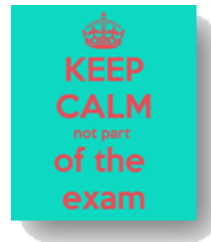
```
# Ignore VSCode local configurations
.vscode/*

# Ignore ROS2 temp folders
install/*
logs/*
build/*

# Ignore specific file
i_love_maneskin.txt
```



# Submodules



You can include a git repo as part of another git repo!

- › Maintainability, scalability, etc...

## Syntax

- › ..from the folder where you want to include...
- › `$ git submodule add <REPO-URL>`
- › It will create a `.gitmodules` file in the root folder

```
[submodule "my_module"]  
    path = src/my_module  
    url = https://github.com/something/my_module.git
```

Where:

- › `url` can also be relative
- › You will also need to commit the `.gitmodules` file
- › Update with `$ git submodules update -init --recursive`



## Step 3: Continuous Integration, Continuous Deployment

---

- › We will see this later in the course....

# References

---



## Course website

- › <http://hipert.unimore.it/people/paolob/pub/ProgSW/index.html>

## My contacts

- › [paolo.burgio@unimore.it](mailto:paolo.burgio@unimore.it)
- › <http://hipert.mat.unimore.it/people/paolob/>