

# Introduction to Git(Hub)

---

20/04/2020

Stelios Vitalis



# Versioning and Collaboration

*The general concept*

It's useful because:

- It tracks history of our work
- It allows us to work as a team
- It can be used to manage a project

# Version Control System (VCS)

## Definition

### Definition

*Version Control is the **management** of changes to documents, computer programs, large web sites, and other collections of information.<sup>1</sup>*

---

<sup>1</sup>Wikipedia

# VCS

## *Definitions*

### Repository

A storage location where all versions and information about them are stored.

### Workspace

The actual working directory of the user.

# Types of VCS

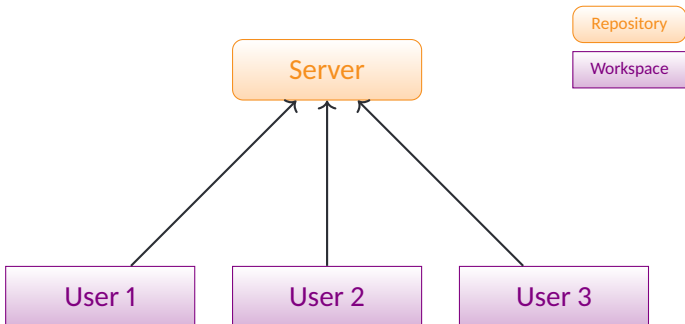
There are two types:

- Centralised
  - Subversion (SVN)
  - Microsoft Team Foundation Server (TFS)
  - Concurrent Versions System (CVS)
- Distributed
  - Git
  - Mercurial

# Centralised

## Architecture

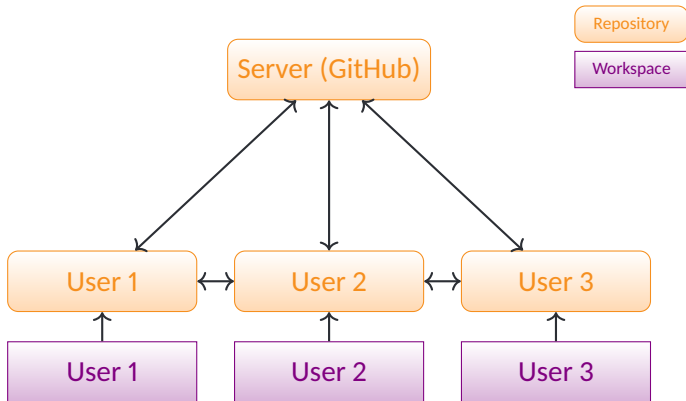
A server is the only repository and every user has a workspace.



# Distributed

## Architecture

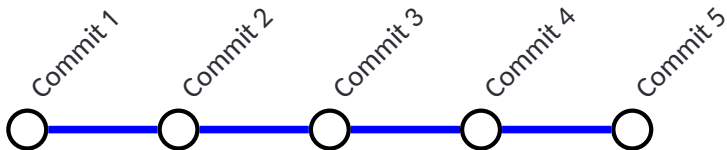
Every user has a copy of the repository and a workspace.





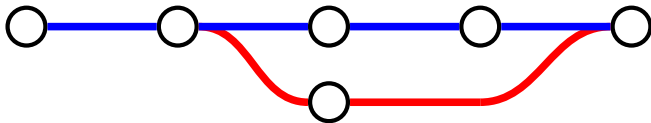
# What is a repository?

*Seems like a time line*



## What is a repository?

*But it's more like a metro network*



# Repository “internals”

*It's a graph*



This nodes are called **commits** or **revisions**.

# Commit or Revision

## *Information*

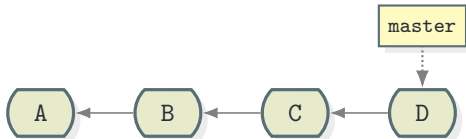
Every commit has:

- **ID:** some identifier
- **Author:** name and email of user who commits
- **Timestamp:** time of commit
- **Message:** what the commit contains

and, of course, the **changes** of the files that are submitted.

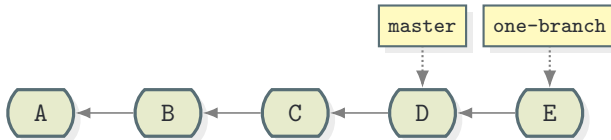
# Repository “internals”

## *Branching*



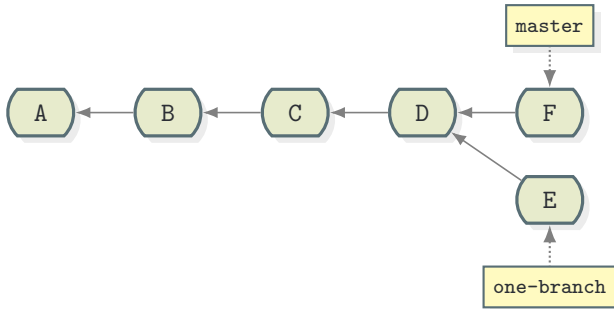
# Repository “internals”

## Branching



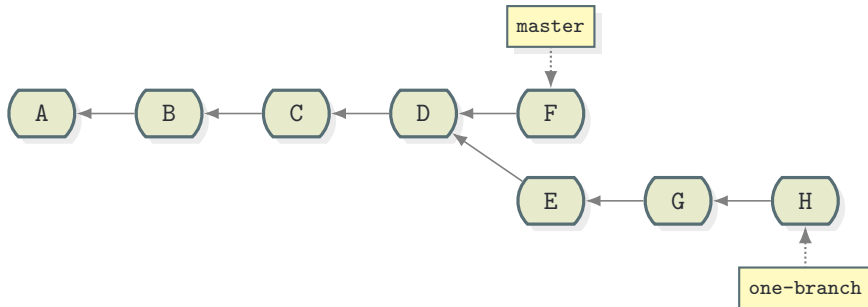
# Repository “internals”

## Branching



# Repository “internals”

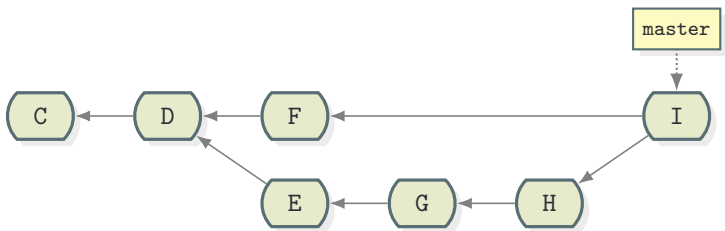
## Branching





# Repository “internals”

## Merging



See you at the lab

13.00pm at Discord git-help (voice channel).

Check the git-help text channel for any preparation instructions.



# Git

## *Some History*

Git was created by Linus Torvalds in 2005 because there was no decent version control system to maintain the Linux kernel. He described the tool as "the stupid content tracker".

# Git

## *Some History*

He setup the following objectives:

- Performance
- Take CVS as an example of what not to do; if in doubt, make the exact opposite decision
- Support a distributed, BitKeeper-like workflow
- Include very strong safeguards against corruption, either accidental or malicious

# Git

## *Definitions*

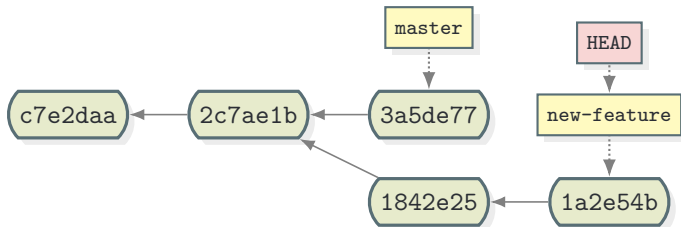
It's an open-source destributed VCS.

Specific definitions:

- Every commit has an ID which is its contents hash. E.g.:  
2c7ae1b9865e58797ba326d2f7a115bebb034fd7
- We call the “current” commit as **HEAD**.

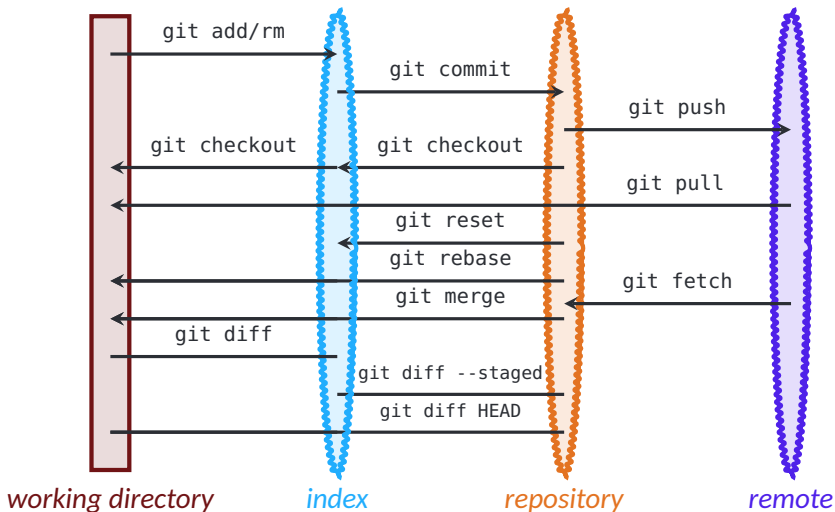
# Git

## Definitions



# Git

## Definitions





# Create a repository

*From scratch*

```
git init
```

Creates a new empty repository.

The *working directory* is not affected, but an empty repository and index is created.

## Create a repository

*From a remote*

```
git clone remote_address
```

Creates a copy of an existing online repository.

- A new folder is created.
- All commits/branches etc. are copied locally.
- The source repository is set as the *origin* remote.

## Status

*See where you stand*

```
git status
```

Gives all information about the current state of repository and index.

- Shows current branch and difference with remote.
- Shows the staged files.
- Shows changed but not staged files.
- Shows untracked files.

## Create a commit

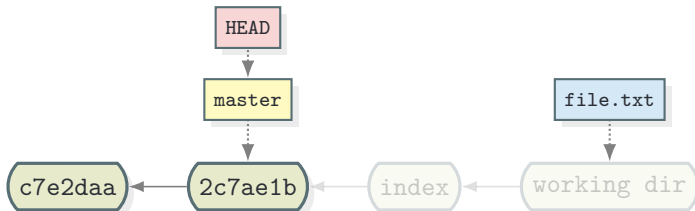
*Add files to the index*

```
git add filename
```

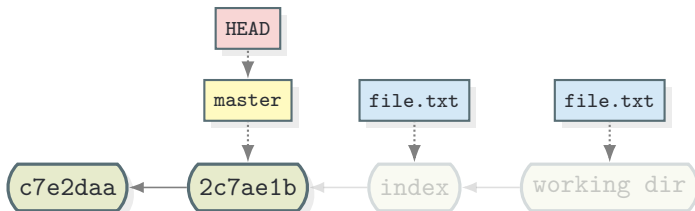
Adds the file to the index. We say it's **staged**.

- The current file from *working directory* is copied to the *index* only if it has changes compared to HEAD.
- The *filename* can be a pattern. Eg. "git add ." will add all files.
- Nothing has been committed yet.

git add file.txt



(a) Before...



(b) ... and after

## Create a commit

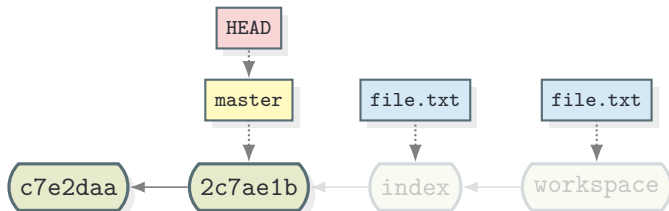
*Commit staged files*

```
git commit -m "message"
```

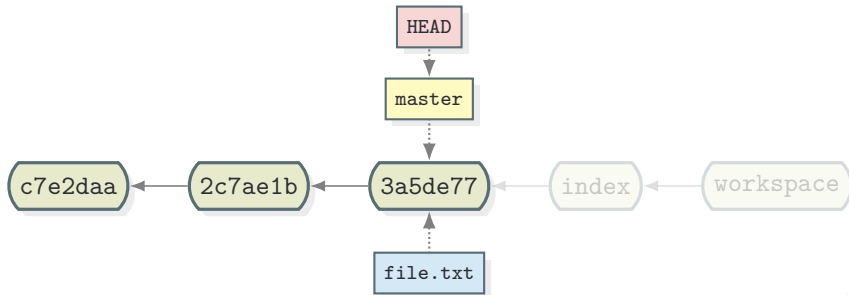
Creates a commit from a copy of the index.

- The new commit has the given message.
- After the commit, the index is cleared.
- The *HEAD* and the current *branch* tags are moved to the new commit.

`git commit -m "Changes to file.txt"`



(a) Before...



(b) ... and after

## Move to a commit

*Change branch or version*

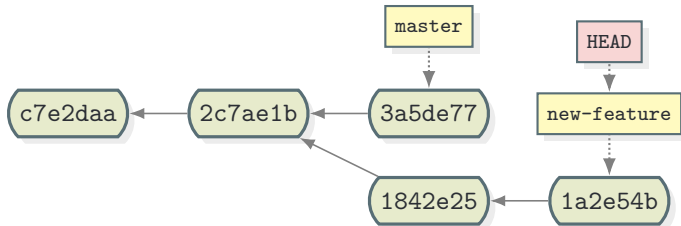
```
git checkout ref
```

Moves to a branch/commit and changes the working directory accordingly.

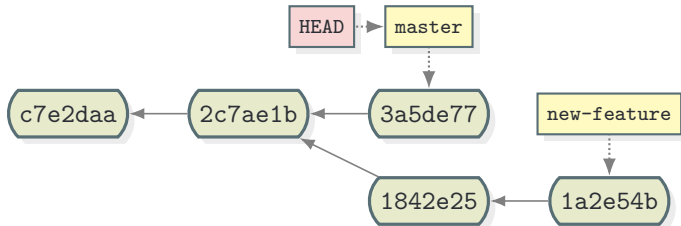
- The *ref* can be a branch name, commit id or something else...
- The *HEAD* moves to the referred commit.
- The current branch changes (if a branch name is given).



## git checkout master

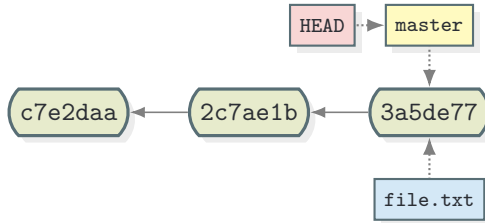


(a) Before...

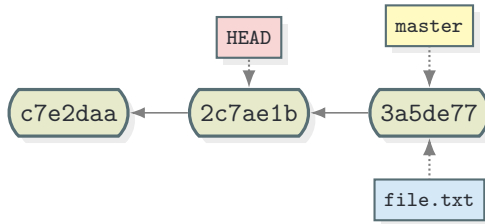


(b) ... and after. **The working directory will change as well!**

git checkout 2c7ae1b

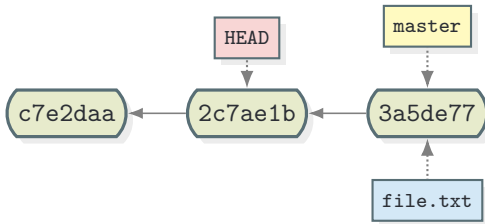


(a) Before...

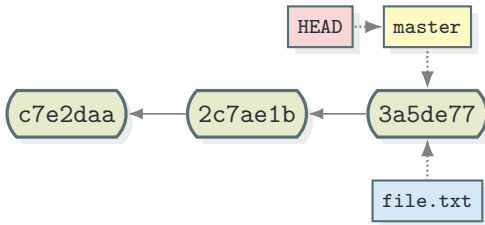


(b) ... and after. That's called a **detached HEAD** state!

## git checkout master



(a) Before on a 'detached HEAD' state...



(b) ... and after. Back to normal.

## Create a branch

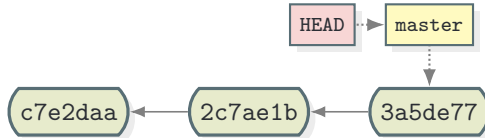
*Use the branch command*

```
git branch new-branch-name
```

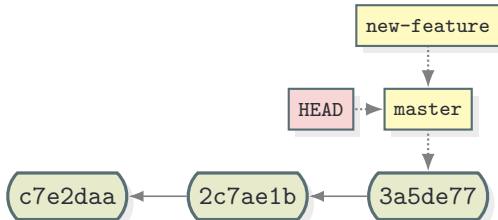
Create a new branch here.

- The new branch is created on the position of HEAD.
- The HEAD still points to the previous position.

## git branch new-feature

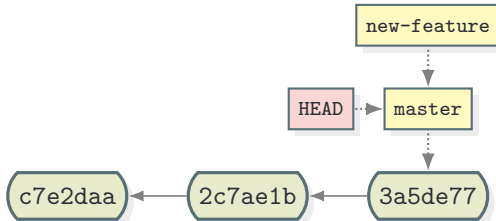


(a) Before...

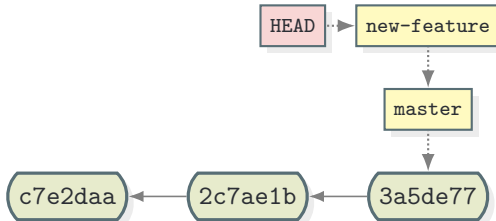


(b) ... and after.

## git checkout new-feature



(a) Before...



(b) ... and after.

## Create a branch

*Use Checkout instead*

```
git checkout -b new-branch-name
```

Create a new branch here and switch to it.

- The new branch is created on the position of HEAD.
- The HEAD now points to the new branch.

## Merge branches

*There is a command for that*

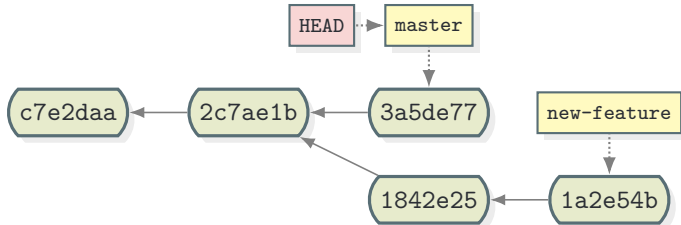
```
git merge other-branch
```

Merges the *other-branch* to this one.

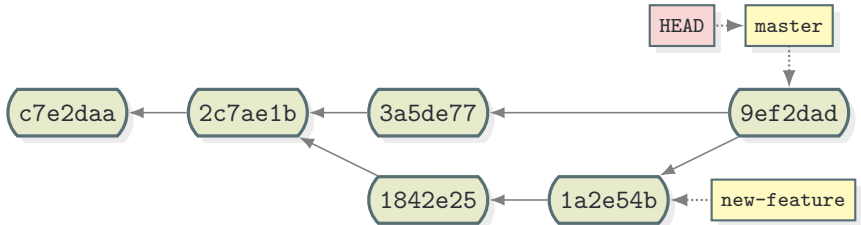
- You call merge when you are on the branch that wants to “receive” the changes.
- Both branches remain after the merge, but changes have been incorporated to the current.



## git merge new-feature



(a) Before...



(b) ... and after. **The working directory will change as well!**

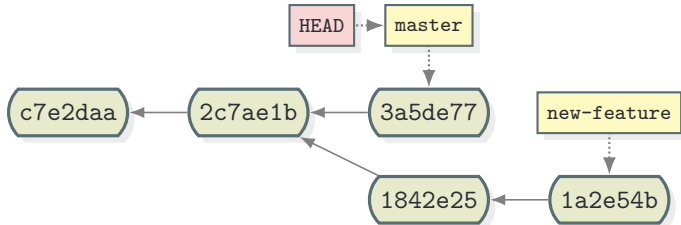
## Conflict

A conflict happens when during a merge there are changes to the same lines of the same document or when there is contradictory changes.

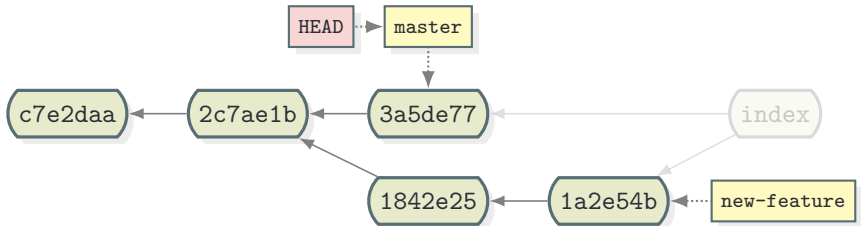
```
1 <<<<<< HEAD
2
3 Here is the original change.
4 =====
5 Here is the modified change.
6 >>>>>> 58326c301d09b58f3ac23d616e73f7b478424cc5
7
```

- Both versions are shown.
- You change your files as normally.
- You add them again to the index.
- You commit.

## git merge new-feature



(a) Before...



(b) ... and after. **The conflicts are marked and you have to resolve!**

# Remotes

*The remote repositories*

## Remotes

A list of remote repositories that we can exchange commits.

- Every *remote* is reached through a url.
- It is given a *name* to be distinguished.
- Normally we call the “main” remote as **origin**.

## Fetch

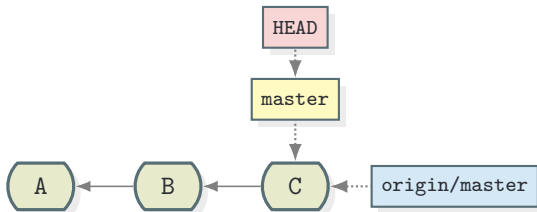
*Get commits from remote*

```
git pull
```

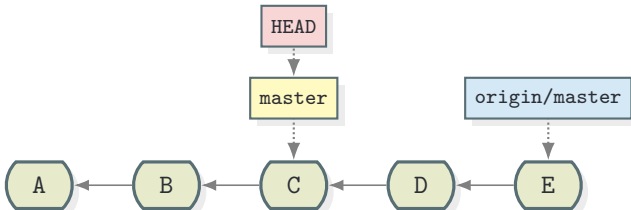
Fetches all commits from the remote and tries to merge the upstream to the current one.

- Remember, remote branches are also branches, so they can be merged.

## git fetch origin

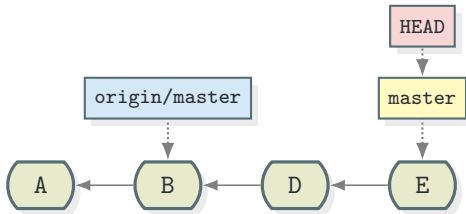


(a) Before...

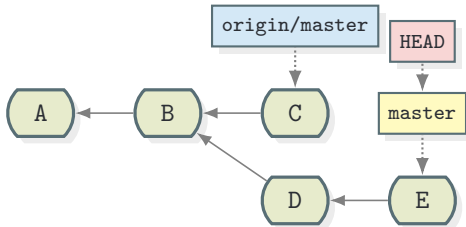


(b) ... and after.

## git fetch origin



(a) Before...



(b) ... and after.

## Pull

*It's a fetch and merge*

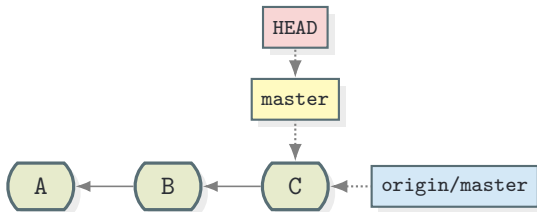
```
git pull [remote-name]
```

Fetches all commits from the remote and merges.

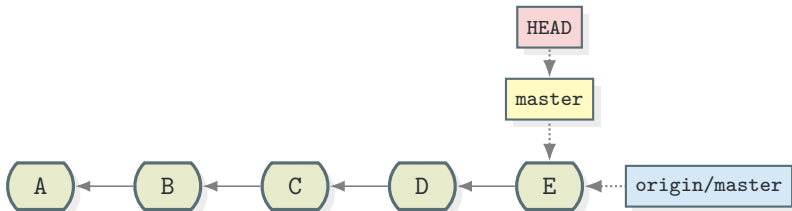
- It does `git fetch` and `git merge remote-name/branch`



# git pull

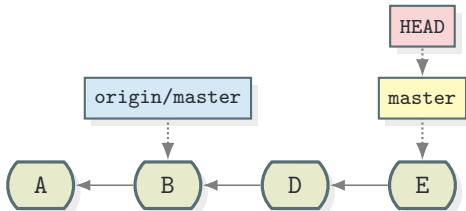


(a) Before...

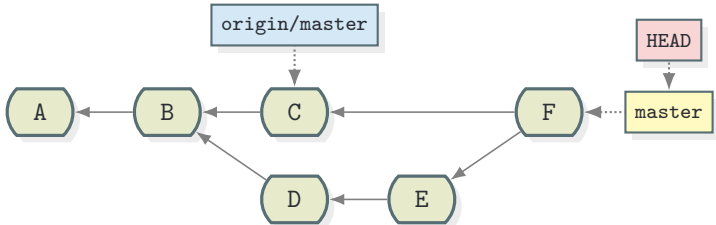


(b) ... and after.

# git pull



(a) Before...



(b) ... and after.

## Push

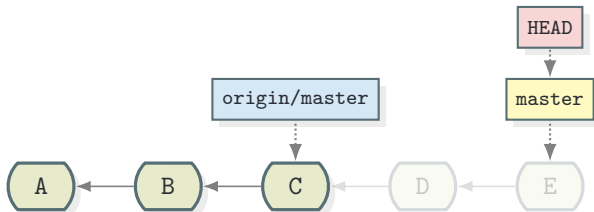
*Share your changes to the world*

```
git push
```

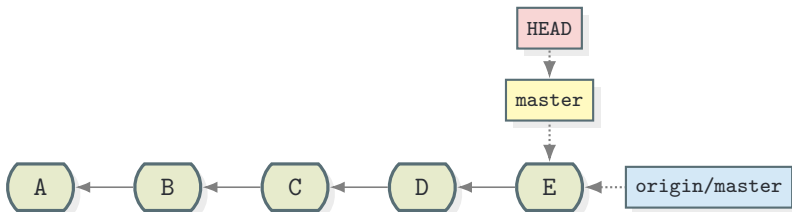
Push your local branch(es) to the remote.

- Normally it just pushes the current branch to the upstream.
- Will only work if the remote branch is updated and there is a fast-forward to the local branch.

## git push

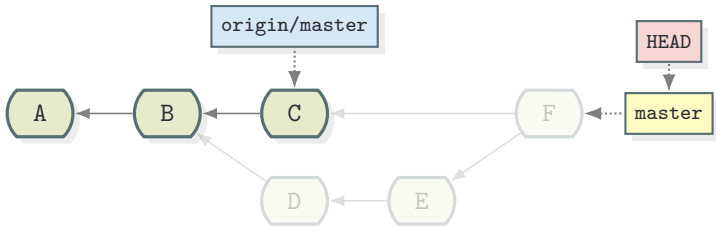


(a) Before...

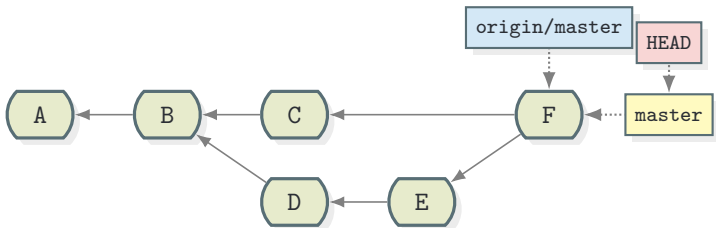


(b) ... and after.

## git push



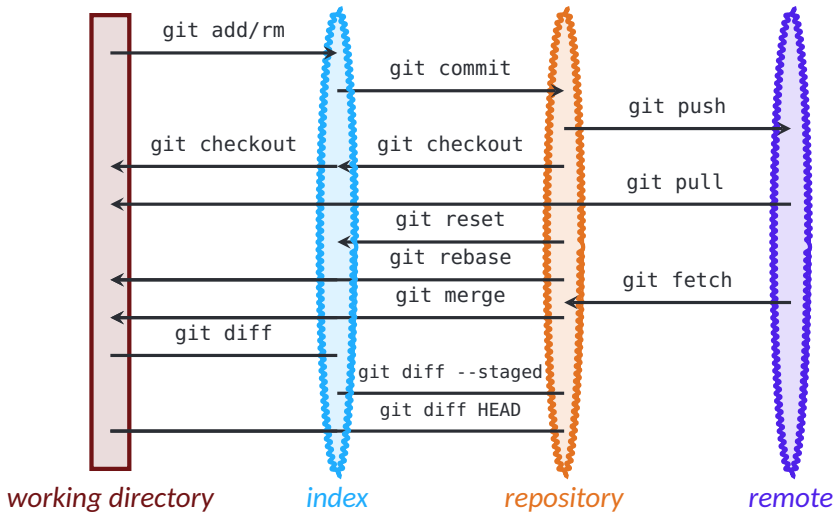
(a) Before...



(b) ... and after.

# Git

## Overview





# Github

*It's just a web app*

It's a repository hosting service, based on an closed-source web app that wraps git!

- You can create remote repositories there (free for public, paid for private use).
- It incorporates some management tools as well (issue-tracking, pull requests, continuous integration).

There are other platforms out there as well, like Gitlab.



# Github

*Host your own project*

You can create repositories in GitHub and host your source code.

Same with GitLab.

You can do that before or after you create a local repository.

# Github

## Clone an existing repository

```
git clone https://github.com/qgis/QGIS.git
```

This repository Search Pull requests Issues Marketplace Explore

qgis / QGIS Watch 267 Star 1,725 Fork 1,158

<> Code Pull requests 95 Projects 0 Wiki Insights

QGIS is a free, open source, cross platform (lin/win/mac) geographical information system (GIS) <http://qgis.org>

44,491 commits 48 branches 97 releases 244 contributors GPL-2.0

Branch: master New pull request Create new file Upload files Find file Clone or download

ridhont Merge pull request #5561 from pvalsecc/wms\_ogc\_filters

.ci/travis	[docker] allow updating dependencies	
.docker	Remove dependency on QtScript	
.github	Rename CONTRIBUTE.md to CONTRIBUTING.md	
.tx	[18n] integrate esperanto translation	a year ago
cmake	Bump minimum spatialite version to 4.2	29 days ago
cmake_templates	Don't use thread_local on mingw or OpenBSD builds	a month ago
debian	Remove dependency on QtScript	5 days ago
doc	update HTML data provider metadata (#5700)	2 days ago
editors/QtCreator	Add header to Qt templates	8 months ago
external/astyle	ok is spelled OK	6 months ago

Clone with HTTPS ? Use SSH

Use Git or checkout with SVN using the web URL.

<https://github.com/qgis/QGIS.git>

Open in Desktop Download ZIP