

# Git workshop

Ángel de Vicente

*POLMAG project (<http://research.iac.es/proyecto/polmag/>  
[\(http://research.iac.es/proyecto/polmag/\)\)](http://research.iac.es/proyecto/polmag/)*

E-mail: *angel.de.vicente@iac.es*

Date: June 2, 2020



# **Index**

1. Git basics
2. Git basic usage
3. Working with remotes (i.e. GitHub)
4. Branches
5. Merging
6. Other

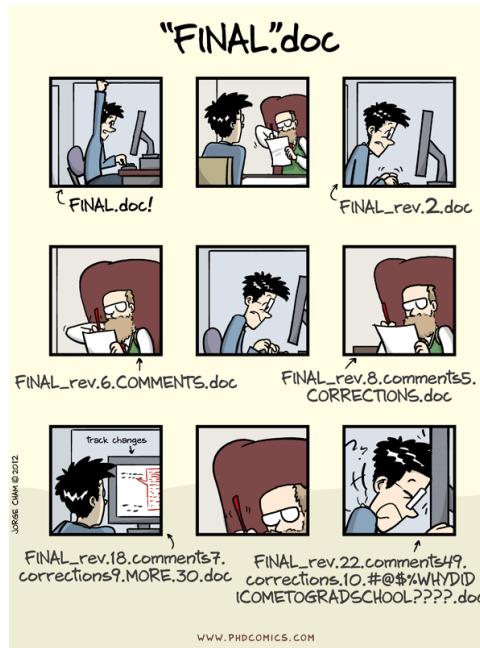
# **1. Git basics**

**I'm assuming that you are familiar with the following concepts:**

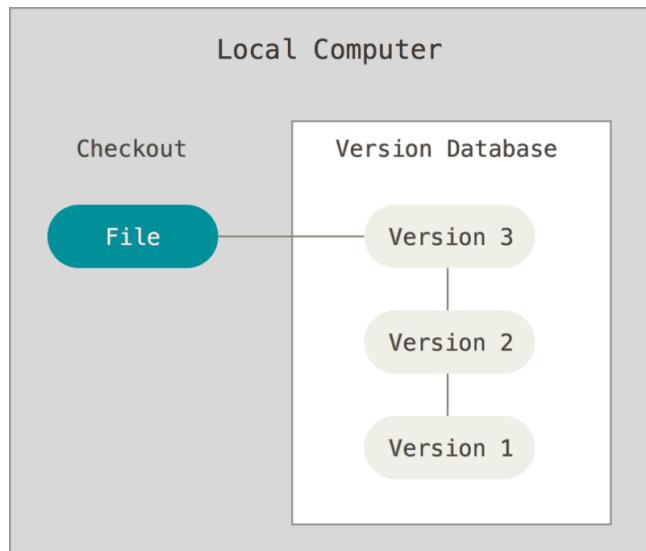
- What is version control?
- What do we version control (text vs. binary files)
- What is a version control system
- You have used a version control system directly or indirectly (e.g. Dropbox, Overleaf, etc.)
- Version control ideas: record/playback, branching, merging
- Benefits of version control systems

# Types of Version Control Systems

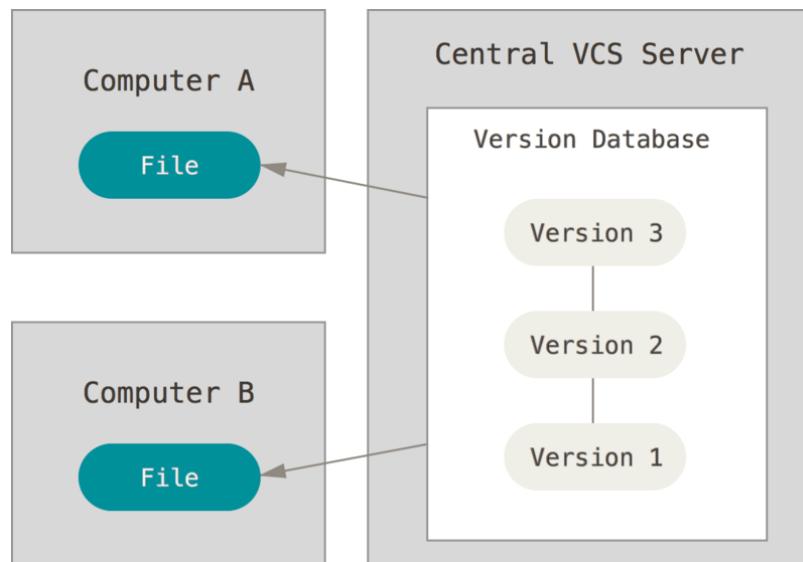
- No VCS (*don't be the one doing this!*)



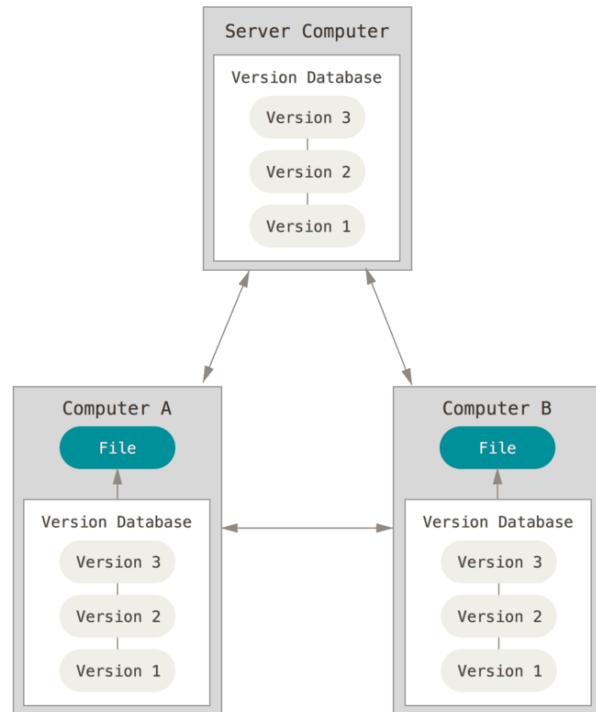
- Local VCS (e.g. RCS)



- Centralized VCS (e.g. Subversion)



- Distributed VCS (e.g. Git)



## Setting up git

- *git config* (details at: <https://git-scm.com/book/en/v2/Getting-Started-First-Time-Git-Setup>)
- Configuration variables stored in three different places:
  - */etc/gitconfig* file (*--system* option)
  - *~/.gitconfig* or *~/.config/git/config* file (*--global* option)
  - *config* file in the Git directory (*--local* option)

## Setting up git (2)

- Basic settings

```
$ git config --global user.name "John Doe"  
$ git config --global user.email johndoe@example.com
```

- Many other options, e.g:

```
$ git config --global core.editor emacs  
$ git config --global color.ui "auto"
```

## Getting help

```
In [17]: ! git --version  
git version 2.25.1
```

```
In [3]: ! git --help | head -n 25
```

```
usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           <command> [<args>]
```

These are common Git commands used in various situations:

start a working area (see also: `git help tutorial`)

clone	Clone a repository into a new directory
init	Create an empty Git repository or reinitialize an existing one

work on the current change (see also: `git help everyday`)

add	Add file contents to the index
mv	Move or rename a file, a directory, or a symlink
restore	Restore working tree files
rm	Remove files from the working tree and from the index
sparse-checkout	Initialize and modify the sparse-checkout

examine the history and state (see also: `git help revisions`)

bisect	Use binary search to find the commit that introduced a bug
diff	Show changes between commits, commit and working tree, etc
grep	Print lines matching a pattern
log	Show commit logs
show	Show various types of objects

```
In [4]: ! git status -h
```

```
usage: git status [<options>] [--] <pathspec>...

    -v, --verbose            be verbose
    -s, --short              show status concisely
    -b, --branch              show branch information
    --show-stash              show stash information
    --ahead-behind            compute full ahead/behind values
    --porcelain[=<version>]   machine-readable output
    --long                    show status in long format (default)
    -z, --null                terminate entries with NUL
    -u, --untracked-files[=<mode>]  show untracked files, optional modes: all, normal, n
o. (Default: all)
    --ignored[=<mode>]        show ignored files, optional modes: traditional, mat
ching, no. (Default: traditional)
    --ignore-submodules[=<when>]
                                ignore changes to submodules, optional when: all, di
rty, untracked. (Default: all)
    --column[=<style>]         list untracked files in columns
    --no-renames              do not detect renames
    -M, --find-renames[=<n>]  detect renames, optionally set similarity index
```

```
In [19]: ! git status --help | head -n 22
```

GIT-STATUS(1)

Git Manual

GIT-STATUS(1)

#### NAME

git-status - Show the working tree status

#### SYNOPSIS

git status [<options>...] [--] [<pathspec>...]

#### DESCRIPTION

Displays paths that have differences between the index file and the current HEAD commit, paths that have differences between the working tree and the index file, and paths in the working tree that are not tracked by Git (and are not ignored by gitignore(5)). The first are what you would commit by running git commit; the second and third are what you could commit by running git add before running git commit.

#### OPTIONS

-s, --short

Give the output in the short-format.

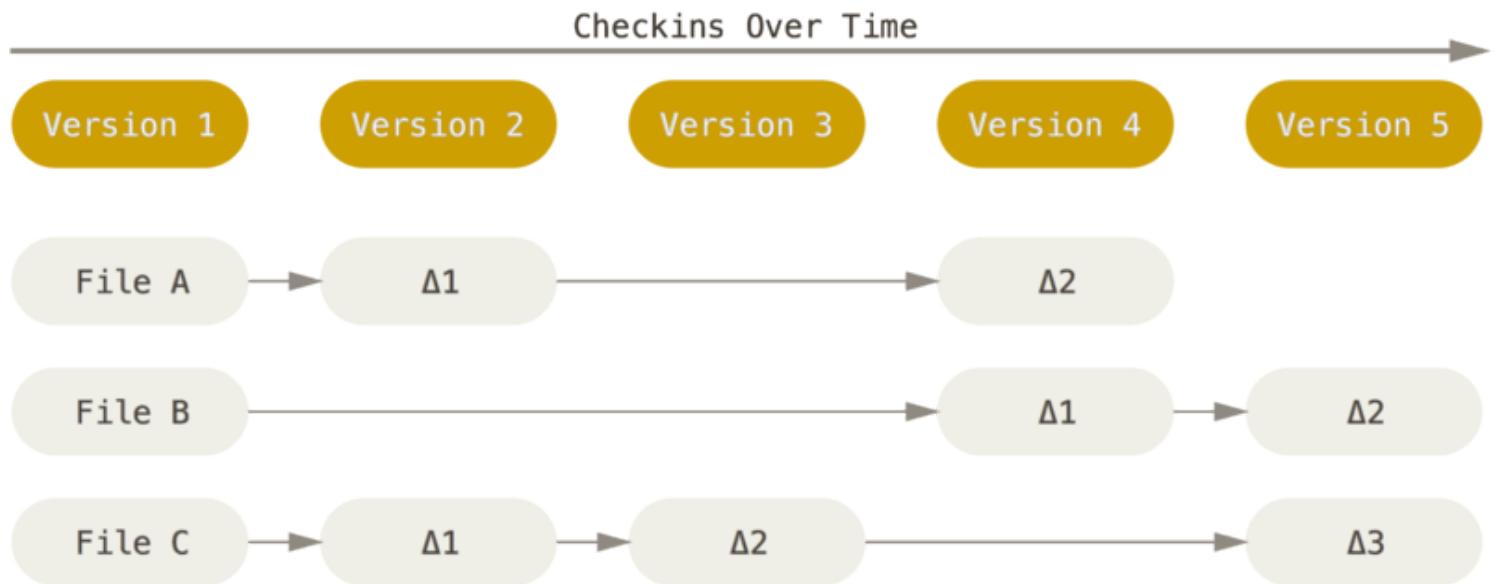
-b, --branch

Show the branch and tracking info even in short-format.

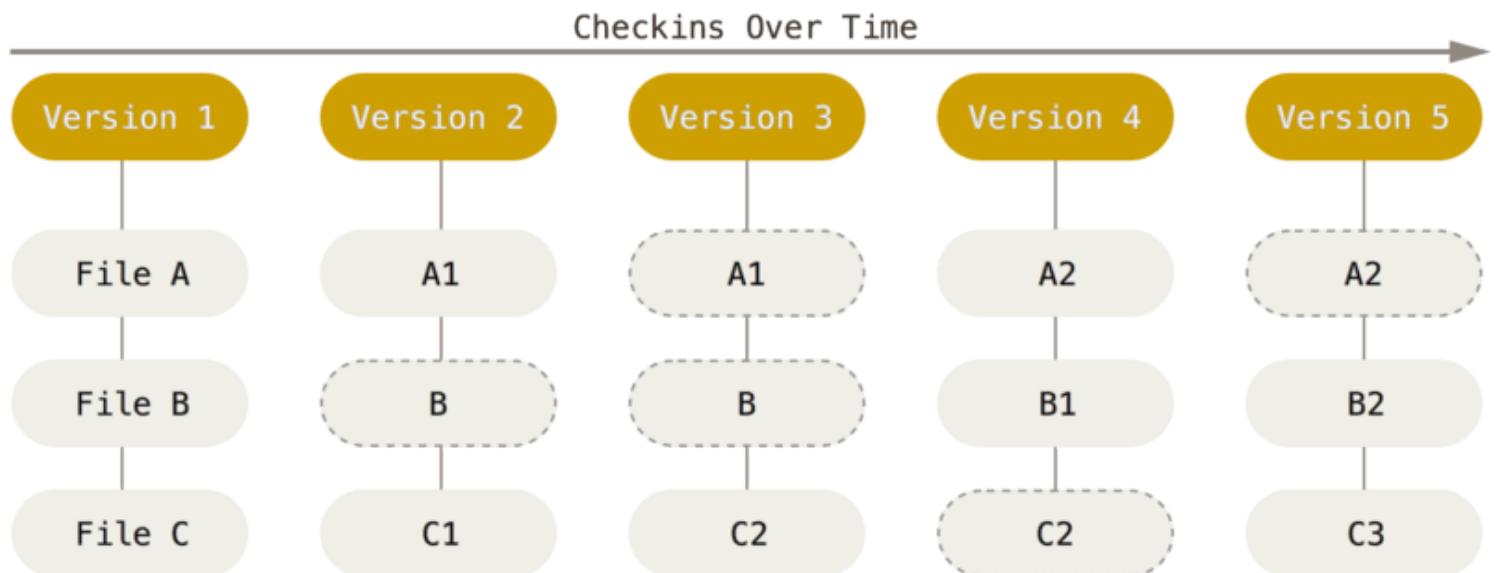
## Main ideas in Git

- Nearly all operations are local
  - .git directory keeps all history
  - very little that you cannot do when offline
- Integrity
  - Everything checksummed before stored
  - Hashes: 24b9da6552252987aa493b52f8696cd6d3b00373
- Snapshots, not differences

- Delta-based VCS

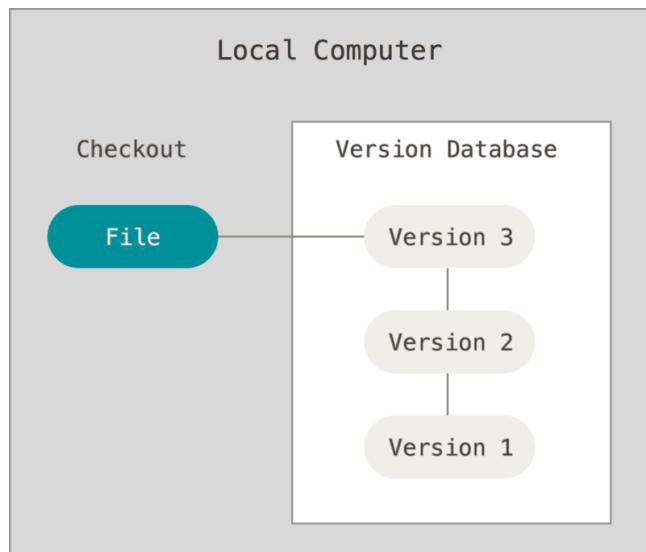


- Snapshots (Git)



## **2. Git basic usage**

## Using git as a local VCS



## Create a repository

**git init**

In [21]:

```
%%bash

cd ../Demos/Repos
rm -rf First_Demo ; mkdir First_Demo ; cd First_Demo

git init -q
ls -al

total 12
drwxr-xr-x 3 angelv angelv 4096 May 12 15:39 .
drwxr-xr-x 6 angelv angelv 4096 May 12 15:39 ..
drwxr-xr-x 7 angelv angelv 4096 May 12 15:39 .git
```

- All repository information goes to .git directory (DO NOT EDIT by hand)

In [7]:

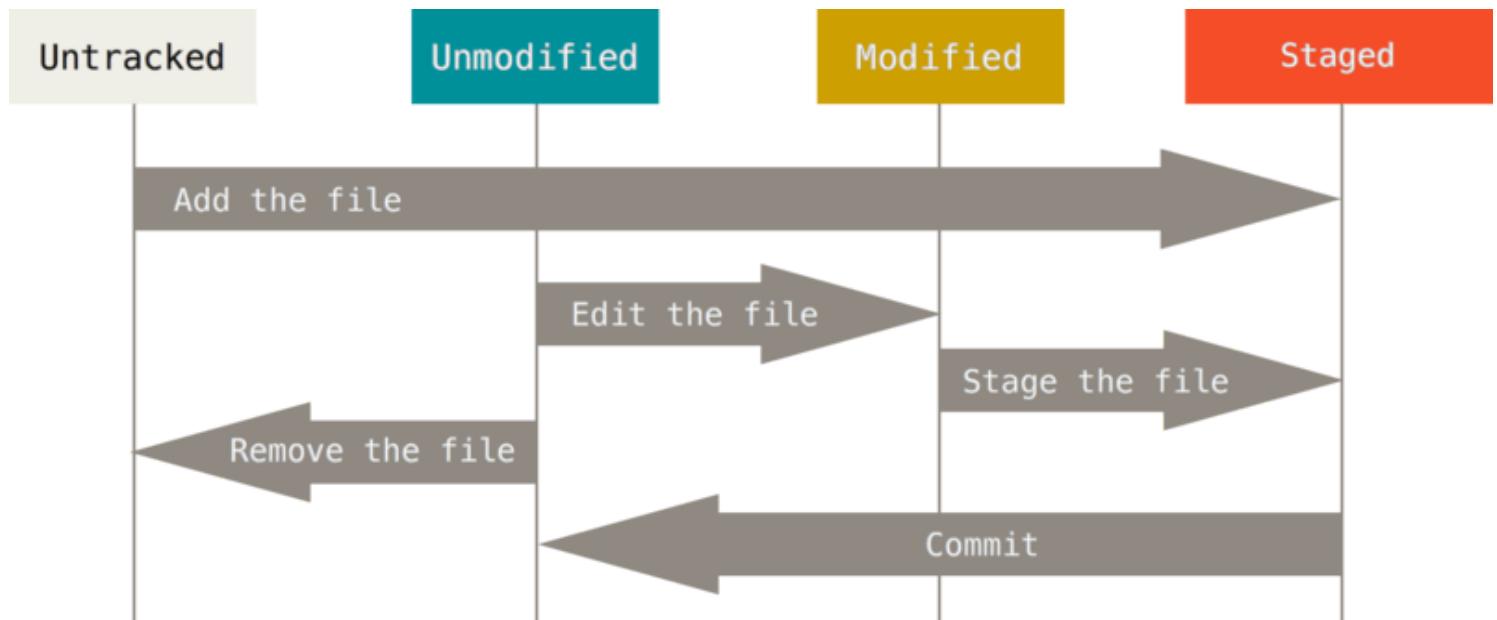
```
%%bash  
  
cd ../../Demos/Repos  
ls First_Demo/.git
```

```
branches  
config  
description  
HEAD  
hooks  
info  
objects  
refs
```

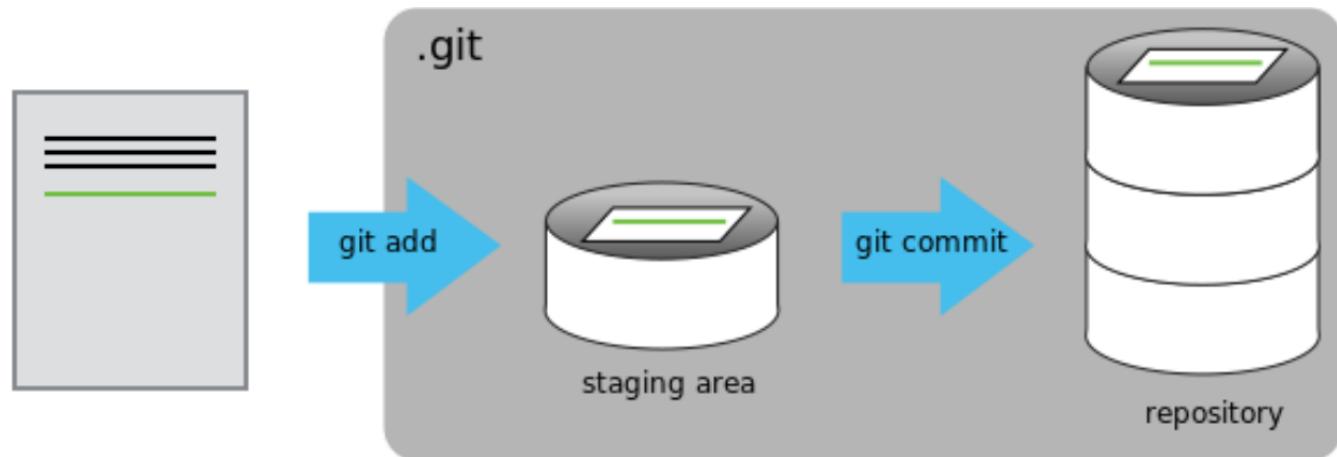
## Recording changes

- git status
- git add
- git commit
- git rm
- git mv

- Tracking changes



- Committing changes



## **Recording changes demo**

[demo-1-1.cast](https://asciinema.org/a/Rff5PnrBC79FMI7ThpbWBLX8D?autoplay=1&cols=180&rows=40) (<https://asciinema.org/a/Rff5PnrBC79FMI7ThpbWBLX8D?autoplay=1&cols=180&rows=40>).

## **Viewing history**

- git log
- git diff

## **Viewing history demo**

[demo-1-2.cast \(https://asciinema.org/a/zHPylAmuKN3TWmGhQJiPOidZd?autoplay=1&cols=180&rows=40\)](https://asciinema.org/a/zHPylAmuKN3TWmGhQJiPOidZd?autoplay=1&cols=180&rows=40)

## Undoing things

- `git commit --amend`
- `git restore --staged CONTRIBUTING.md`
- `git restore README.md`

## Undoing things demo

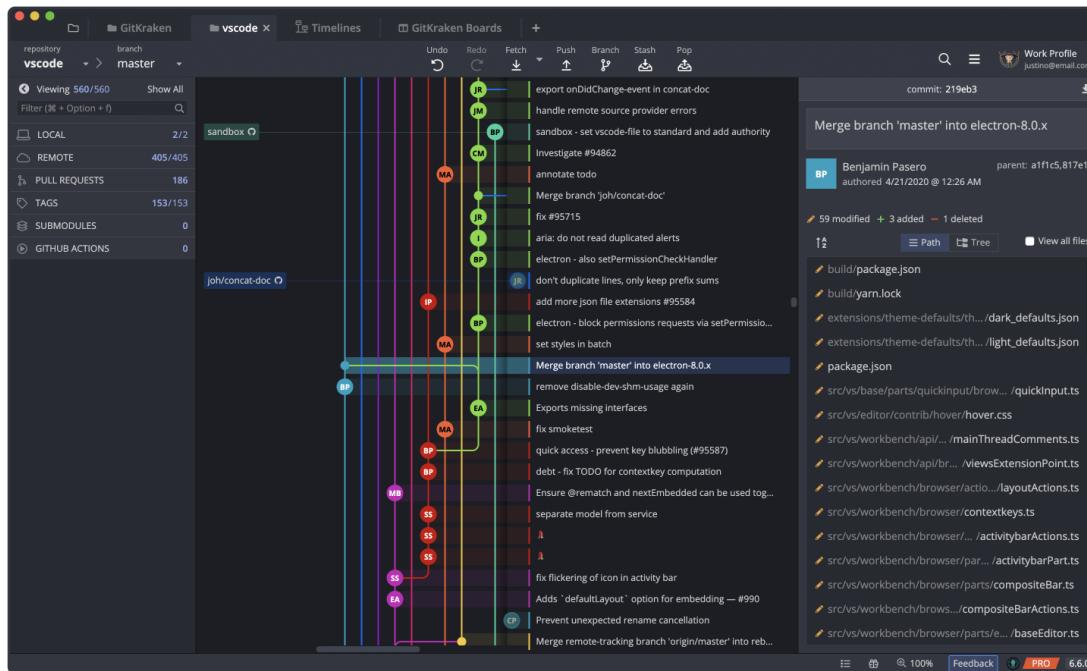
[demo-1-3.cast](https://asciinema.org/a/Sq5xBDSbbbdJ1AxdZmz755xaJ?autoplay=1&cols=180&rows=40) (<https://asciinema.org/a/Sq5xBDSbbbdJ1AxdZmz755xaJ?autoplay=1&cols=180&rows=40>)

- In older versions of `git`, instead of `git restore` you would use:
    - `git reset HEAD ...`
    - and `git checkout ...`
- (see examples in <https://git-scm.com/book/en/v2/Git-Basics-Undoing-Things> (<https://git-scm.com/book/en/v2/Git-Basics-Undoing-Things>))

## Command-line vs GUIs

- Command-line lets you:
  - access *every* aspect of Git
  - can be automated in scripts
  - 'lingua franca'
- GUIs
  - the entry barrier for users is lower
  - can group common usage patterns
  - help with advanced options
  - choose your poison: GitKraken, GitAhead, Magit, ...
    - <https://git-scm.com/download/gui/linux> (<https://git-scm.com/download/gui/linux>)

- GitKraken <https://www.gitkraken.com/> (<https://www.gitkraken.com/>)



- GitAhead <https://gitahead.github.io/gitahead.com/> (<https://gitahead.github.io/gitahead.com/>)

The screenshot shows the GitAhead application interface. The main window title is "gitahead - master - behind: 2 (origin/master)". The left sidebar displays a list of recent commits from the "master" branch:

- Jason Haslam c02358e Merge branch 'ExternalTools'
- Jason Haslam 1d80824 v1.1.4 Changed version to 1.1.4.
- Jason Haslam c054b29 Updated libgit2 to tip of master.
- Jason Haslam 6898810 Simplify and tweak toolbar button style. ★
- Jason Haslam cea72d Set toolbar button colors through the palette and use the button ... ★
- Shane Gramlich 112eac Merge branch 'master'
- Shane Gramlich f347b5f Remove failed on start error

The commit message field contains "Make a really great change." Below it, the "Commit Message:" field shows "1 of 3 files staged, 1 file partially staged". The "Commit" button is visible. The right side of the interface shows the staged files:

- conf/editor.lua** (M)
 

```
@@ -2,5 +2,5 @@ local family = "Courier New"
2 2 local size = 11
3 3 if platform == "mac" then
4 4   family = "Menlo"
5 5   size = 13
6 6 elseif platform == "win" then
```
- src/app/GitAhead.cpp** (M)
 

```
@@ -15,4 +15,5 @@
15 15 int main(int argc, char *argv[])
16 16 {
17 17   // Set attribute.
18 18   Application::setAttribute(Qt::AA_UseHighDpiPixmaps);
19 19 }
```

At the bottom, there are status messages: "12:36 PM - Reset - master to 6a82e89" and "12:38 PM - Pull - master from origin". A warning message indicates an issue with a conflict in "src/ci/installer/mac.mm":

- Fetch - origin Everything up-to-date.
- Fast-forward - master to origin/master
  - Unable to fast-forward 'master' - 1 conflict prevents checkout
    - src/ci/installer/mac.mm
    - You may be able to reconcile your changes with the conflicting files by [stashing](#) before you [fast-forward](#). Then [unstash](#) to restore your changes.
    - If you want to create a new merge commit instead of fast-forwarding, you can [merge](#) without [fast-forwarding](#) instead.

- Magit (for Emacs) <https://magit.vc/> (<https://magit.vc/>)

The screenshot shows the Magit interface running in a terminal window. On the left, there's a file browser with a tree view of a Git repository. The root directory contains files like `README.md`, `index.html`, and `languages`. The `languages` directory is expanded, showing subfiles for various languages such as `es_ES`, `fr_FR`, and `de_DE`. The `es_ES` directory is also expanded, showing files like `index.html`, `languages`, and `README`. The `languages` file under `es_ES` contains the text "lingua franca".

On the right, there's a detailed commit log for the `master` branch. The log includes:

- Recent commits:**
  - master Merge pull request #257 from jotoeri/add-guilink
  - origin/master Merge pull request #257 from jotoeri/add-guilink
  - 4.4.0 (17)
- Commit details for master Merge pull request #257 from jotoeri/add-guilink:**
  - Author: Jessica Lord <jlord@glitch.com>
  - Authored Date: Sun Mar 10 20:49:34 2019 -0400
  - Committer: GitHub <40409390+github.com>
  - Commit Date: Sun Mar 10 20:49:34 2019 -0400
  - Parent: 8c64b82 Merge pull request #242 from franciosi/patch-2
  - Parent: fb309c4 Fixed slight typo
  - Contained: master
  - Follows: 4.4.0 (15)
  - Merge pull request #252 from 0x15f9/master
- Commit details for 4.4.0 Merge pull request #231 from mberasategi/es\_ES:**
  - Author: Jessica Lord <jlord@glitch.com>
  - Authored Date: Sun Mar 10 20:49:34 2019 -0400
  - Committer: GitHub <40409390+github.com>
  - Commit Date: Sun Mar 10 20:49:34 2019 -0400
  - Parent: 8c64b82 Merge pull request #242 from franciosi/patch-2
  - Parent: fb309c4 Fixed slight typo
  - Contained: master
  - Follows: 4.4.0 (15)
  - Merge pull request #252 from 0x15f9/master
- Commit details for 4.4.0:**
  - Author: Jessica Lord <jlord@glitch.com>
  - Authored Date: Sun Mar 10 20:49:34 2019 -0400
  - Committer: GitHub <40409390+github.com>
  - Commit Date: Sun Mar 10 20:49:34 2019 -0400
  - Parent: 8c64b82 Merge pull request #242 from franciosi/patch-2
  - Parent: fb309c4 Fixed slight typo
  - Contained: master
  - Follows: 4.4.0 (15)
  - Merge pull request #252 from 0x15f9/master

At the bottom of the commit log, there's a message: "the entry barrier for users is low, and it's common usage to run magit with advanced options".

At the very bottom of the terminal window, the prompt shows the user is in a directory with multiple open terminals (gnus# 1:diary# 2:org# 3:Julia# 4:IPython# 5:IDLE# 6:PORTA# 7:CORONA#) and the current terminal is a Git Workshop#2 session.

### **3. Working with remotes**

## Remote repositories

- Remote repositories are versions of your project that are hosted:
  - on the Internet
  - or more generally in the network somewhere (even in your local machine)
- Crucial for collaborating with others:
  - pushing and pulling data from remotes
- GitHub, GitLab, Bitbucket, etc. are on-line remotes with:
  - extras for collaboration
  - features for code development (e.g. CI/CD)

## Basic remotes commands

- `git remote ...`
  - -v add show rename remove
- `git ...`
  - clone
  - pull
  - push
  - fetch
  - merge

In [9]: %%bash

```
cd ../Demos/Repos/git-it-electron  
git remote -v
```

```
origin https://github.com/jlord/git-it-electron.git (fetch)  
origin https://github.com/jlord/git-it-electron.git (push)
```

In [10]: %%bash

```
cd ../Demos/Repos/git-it-electron  
git remote show origin
```

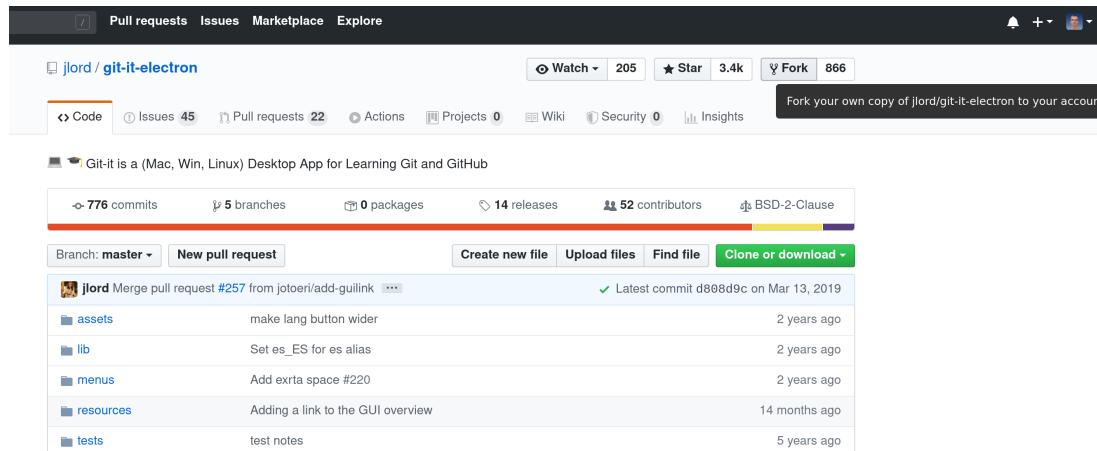
```
* remote origin  
  Fetch URL: https://github.com/jlord/git-it-electron.git  
  Push URL: https://github.com/jlord/git-it-electron.git  
  HEAD branch: master  
  Remote branches:  
    IUTInfoAix-M2105-french-translation      tracked  
    dependabot/npm_and_yarn/handlebars-4.3.0 tracked  
    fix-menu                                tracked  
    lang-tweaks                            tracked  
    master                                 tracked  
  Local branch configured for 'git pull':  
    master merges with remote master  
  Local ref configured for 'git push':  
    master pushes to master (up to date)
```

## Getting started with GitHub (similar for GitLab, Bitbucket, etc.)

- Login to your GitHub account
- Two ways to get a new repository:
  - Fork an existing repository
  - Create a new repository

## *Fork* a repository from GitHub into your account

- Fork the repository *git-it-electron* (<https://github.com/jlord/git-it-electron>) (<https://github.com/jlord/git-it-electron>)



# *Clone your newly forked repository*

The screenshot shows a GitHub repository page for `angel-devicente / git-it-electron`, which is a fork of `jlord/git-it-electron`. The page includes navigation links for Code, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. A descriptive text box states: "Git-it is a (Mac, Win, Linux) Desktop App for Learning Git and GitHub". Below this, there are statistics: 776 commits, 5 branches, 0 packages, 14 releases, 52 contributors, and BSD-2-Clause license. A pull request from `jlord` is listed under the "Branch: master" section. On the right side, there are buttons for "Clone with HTTPS" and "Use SSH", along with a copy link and a "Download ZIP" button.

angela-devicente / **git-it-electron**  
forked from [jlord/git-it-electron](#)

Code Pull requests 0 Actions Projects 0 Wiki Security 0 Insights Settings

Git-it is a (Mac, Win, Linux) Desktop App for Learning Git and GitHub Edit

Manage topics

776 commits 5 branches 0 packages 14 releases 52 contributors BSD-2-Clause

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

This branch is even with jlord:master.

**jlord** Merge pull request [jlord#257](#) from jotoeri/add-guilink ...

assets	make lang button wider
lib	Set es_ES for es alias
menus	Add extra space jlord#220
resources	Adding a link to the GUI overview

Clone with HTTPS Use SSH  
Use Git or checkout with SVN using the web URL.  
<https://github.com/angela-devicente/git-it-electron>

Download ZIP

In [11]:

```
%%bash

cd ../Demos/Repos

# USE YOUR forked repository, not mine!
#
# git clone https://github.com/angel-devicente/git-it-electron.git
# fatal: destination path 'git-it-electron' already exists and is not an empty d
irectory.

if [ ! -d git-it-cpl ] ; then
    git clone https://github.com/angel-devicente/git-it-electron.git git-it-cpl
fi
```

In [12]:

```
%%bash

cd ../Demos/Repos/git-it-cpl

git remote -v
```

```
origin https://github.com/angel-devicente/git-it-electron.git (fetch)
origin https://github.com/angel-devicente/git-it-electron.git (push)
```

## Collaborating in GitHub

- The commands `fetch/merge` and `pull` are useful when collaborating with 'others'
  - another collaborator has pushed changes to the repository that you want to get in your local copy
  - you want to synchronize your own changes in different computers
  - you want to get changes from another repository branch (*we will see this later on*)

# Granting collaborators access in GitHub

angela-devicente / [git-it-electron](#) Watch 0 Star 0

forked from [jlord/git-it-electron](#)

[Code](#) [Pull requests 0](#) [Actions](#) [Projects 0](#) [Wiki](#) [Security 0](#) [Insights](#) [Settings](#)

**Who has access**

PUBLIC REPOSITORY	DIRECT ACCESS
This repository is public and visible to anyone.	0 collaborators have access to this repository. Only you can contribute to this repository.

**Manage access**

You haven't invited any collaborators yet

[Invite a collaborator](#)

**Options**

**Manage access**

- Branches
- Webhooks
- Notifications
- Integrations
- Deploy keys
- Secrets
- Actions

**Moderation**

**Interaction limits**

## **Let's simplify and collaborate with ourselves**

- Clone a second copy of `git-it-electron`, now to directory `git-it-cp2`

## Pull and fetch/merge demo

[demo-1-4.cast \(https://asciinema.org/a/9UTZj3TVwXm6jpftZ2ZcOdk5d?autoplay=1&cols=180&rows=40\)](https://asciinema.org/a/9UTZj3TVwXm6jpftZ2ZcOdk5d?autoplay=1&cols=180&rows=40)

- As *cp1* I make some changes and push to GitHub. Later, as *cp2* we pull those changes.
- Then, as *cp2* I make some changes and push to GitHub. Later, as *cp1* I:
  - first fetch the changes, so I can review them
  - when happy with introducing those changes I can merge them to my repository. (merge in this simple case is basically just absorb all the changes)
- Did you notice *origin/master?* *master* is tracking *origin/master*. Confusing? It will become clear later when talking about branches. Hold on!

# Create a new repository in GitHub (same idea for GitLab, Bitbucket, etc.)

- Create an empty repository in GitHub ...

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere?  
[Import a repository.](#)

---

Owner      Repository name \*

 angel-devicente /  

Great repository names are short and descriptive. [git-it-test](#) is available. Need inspiration? How about [expert-potato](#)?

Description (optional)

 **Public**  
Anyone can see this repository. You choose who can commit.

 **Private**  
You choose who can see and commit to this repository.

Skip this step if you're importing an existing repository.

**Initialize this repository with a README**  
This will let you immediately clone the repository to your computer.

---

Add .gitignore: **None** ▾    Add a license: **None** ▾ 

---

**Create repository**

## **4. Branches**

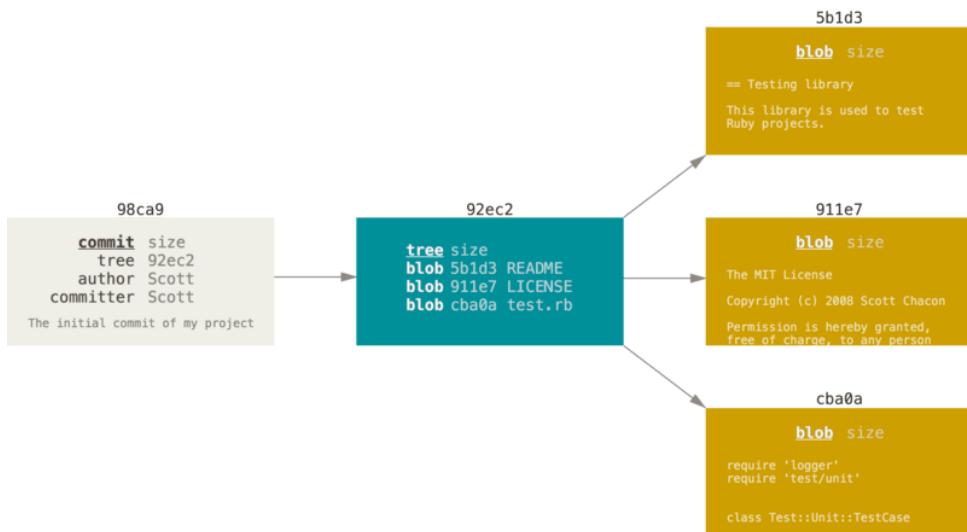
## Branches

- A lot of the power (and some confusion) in Git comes from branches
- Branches are the *killer* feature of Git, very lightweight compared to other VCS
- This encourages workflows that create branches and merge them very often.

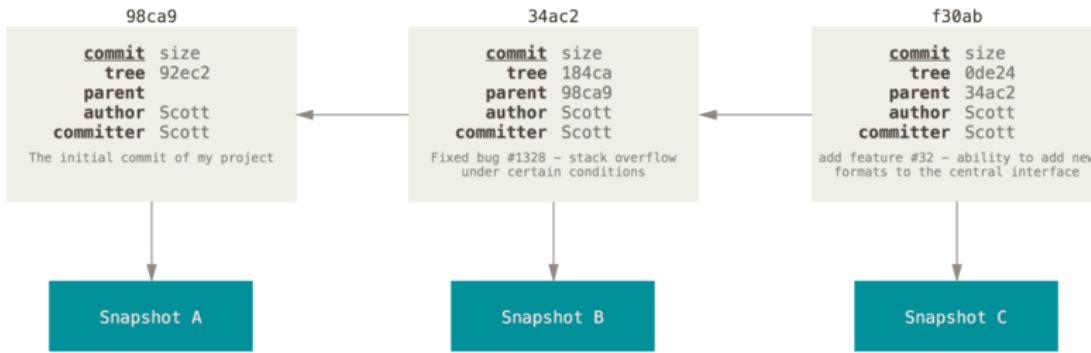
*So, let's try and see some of the Git internals to understand what branches are...*

(see <https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell> (<https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell>))

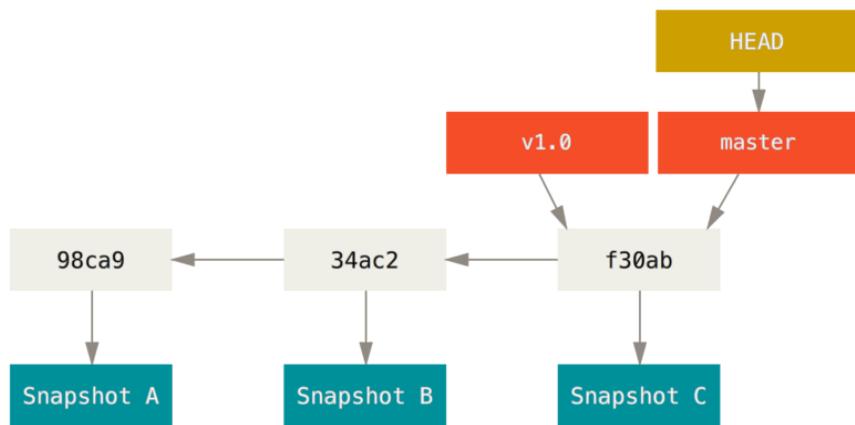
- A *commit* points to a *tree* object, and this to *blobs*
  - For us the important part is just the *commit*, which points to a *snapshot* of our work



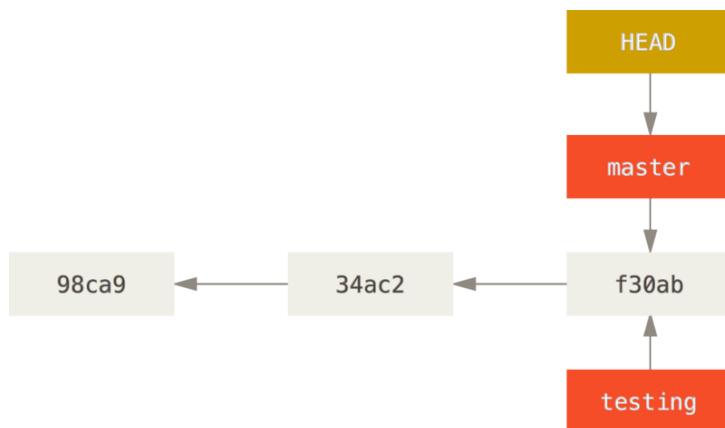
- Creating more *commits*, keeps a linked list
  - (which is basically what you see when you do `git log`)



- A *branch* is just a pointer to one of these *commits*
  - *master* is created when you do `git init`, but it is no special
  - *HEAD* points to the branch we are working on



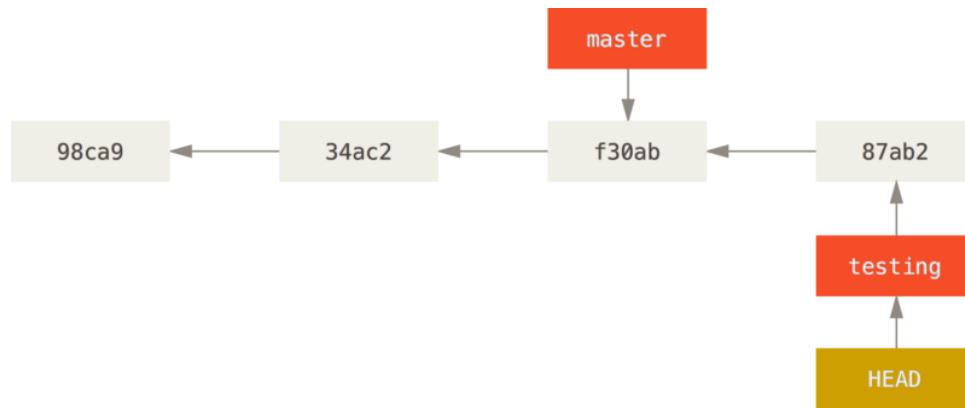
- Creating a branch
  - If we are working in *master* and we do `git branch testing`
    - *testing* branch is created, but we are still working on *master*
    - so you can see *HEAD* is still pointing to *master*



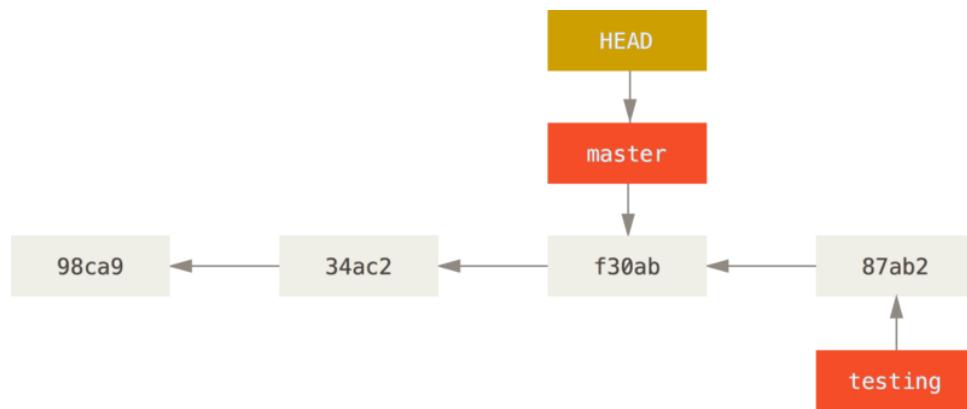
- To work with another branch
  - `git checkout testing` makes `HEAD` point to `testing`
    - when you switch branches in Git:
      - **files in your working directory will change.**
      - **if Git cannot do it cleanly, it will not let you switch at all.**



- Committing to *testing*
  - makes *testing* branch move forward
  - but *master* branch still points to the same commit



- Changing branches
  - `git checkout master` (if *working tree* is *dirty* we won't be able to change branches)



## Branching demo

- In `git-it-cp1`:
  - create a new branch `test_branch1` and change to it
  - add a line to a file and commit changes
  - change back to branch `master` and verify the added line is not present in this branch
  - use `git log` to view the history of all branches (should be able to see in first position both branches: `master` and `test_branch1`, plus `HEAD`)

[demo-1-5.cast \(https://asciinema.org/a/0du1PPZviaW6EoQ7DSATEiAqe?autoplay=1&cols=180&rows=40\)](https://asciinema.org/a/0du1PPZviaW6EoQ7DSATEiAqe?autoplay=1&cols=180&rows=40)

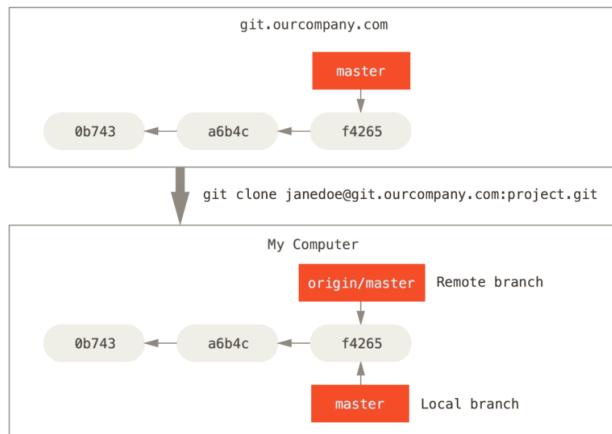
## Remote Branches

If you read <https://git-scm.com/book/en/v2/Git-Branching-Remote-Banches> (<https://git-scm.com/book/en/v2/Git-Branching-Remote-Banches>), you get all the details, but it can be a bit confusing. Most of the time, you just need three concepts:

- Remote branch: a branch in a **remote** repository, for example:
  - GitHub,
  - a repository in another PC of yours,
  - another directory in the same machine
- Remote-tracking branch: a **local** bookmark pointing to a remote branch
  - can become out-of-date if the remote branch gets new commits
- Tracking branch: a **local** branch, linking the *local* branch to the *remote* branch.

*Let's see some examples*

- `git clone` will take the remote repository in `git.ourcompany.com` and copy it to your local repository.
  - it will create remote-tracking branches (here `origin/master`)
  - it will create and checkout a *local* tracking branch named `master`, (Only one local branch is created, linked to the currently active branch in the remote repository).



In [14]:

```
%%bash

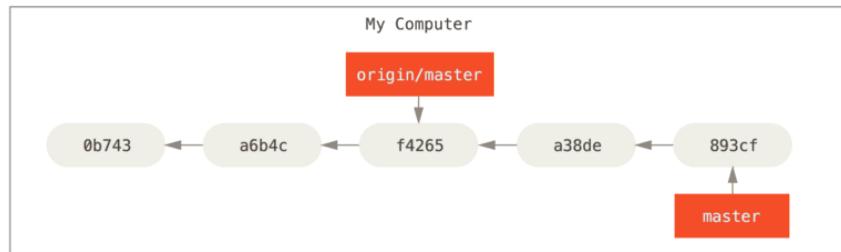
cd ../Demos/Repos/git-it-cp2
git remote -vv
echo
git branch --all
echo
git branch -vv

origin  https://github.com/angel-devicente/git-it-electron.git (fetch)
origin  https://github.com/angel-devicente/git-it-electron.git (push)

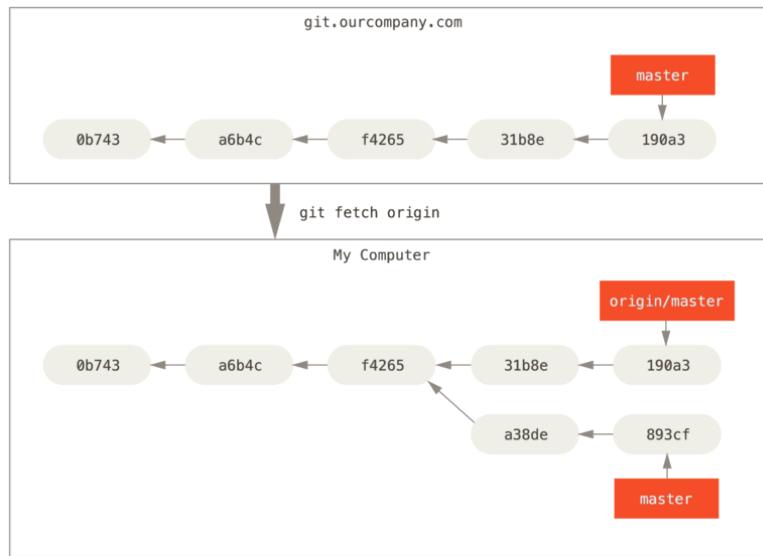
* master
  remotes/origin/HEAD -> origin/master
  remotes/origin/IUTInfoAix-M2105-french-translation
  remotes/origin/dependabot/npm_and_yarn/handlebars-4.3.0
  remotes/origin/fix-menu
  remotes/origin/lang-tweaks
  remotes/origin/master

* master b992ccb [origin/master] Added FORKED-COPY.md
```

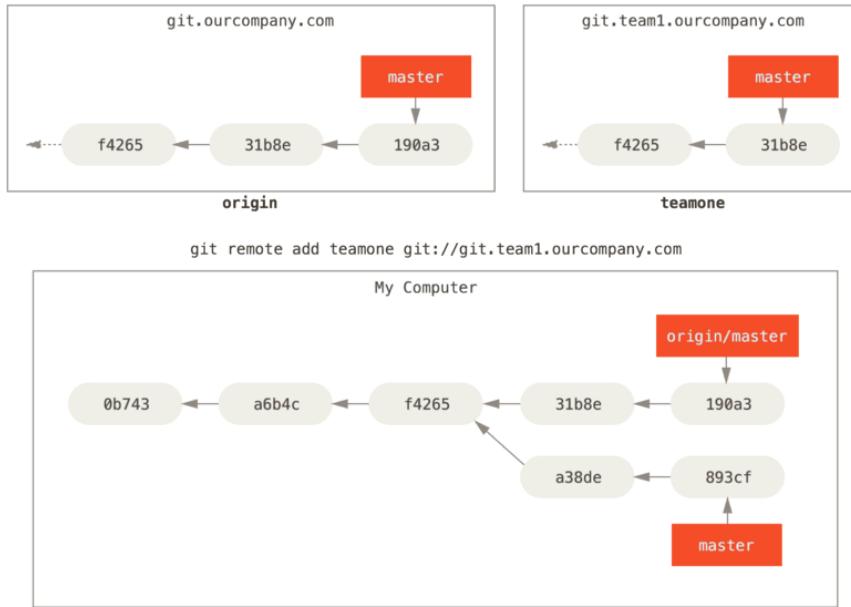
- Remote and local repositories can become unsynchronized
  - Here there were commits in the local branch and the remote branch
  - Note that the remote-tracking branch `origin/master` doesn't move (until you use `git fetch` your view of the remote repository is out-of-date)



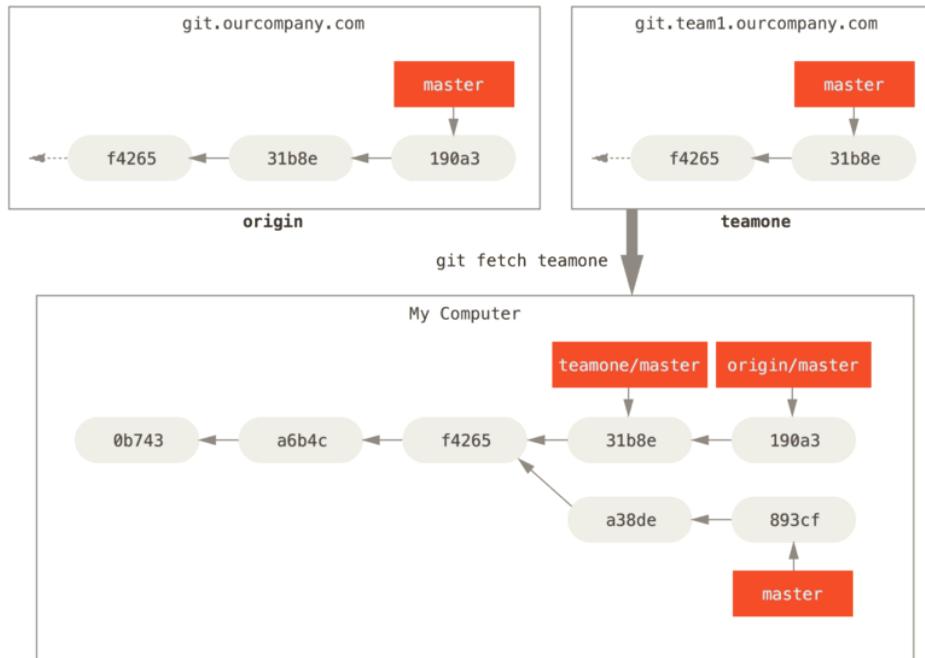
- `git fetch` will synchronize your remote-tracking branch
  - at this point `origin/master` and `master` have diverged (will have to use `git merge` or similar)



- You can have many remotes



- ...`git fetch` to synchronize their state to your repository



- You can `fetch`/`merge` or `pull` from remote-tracking branches without creating local branches
- If you want to work (and perhaps contribute) to a remote branch, you can create a local branch out of it:
  - `git checkout <branch>` (if no name conflict, will just create a tracking branch)
  - `git checkout -b <branch> --track <remote/branch>` (specific, in case name there are name conflicts)

## Demo

[demo-1-6.cast \(https://asciinema.org/a/wlBMwwJXF24xYPqzfi3QS91SG?autoplay=1&cols=180&rows=40\)](https://asciinema.org/a/wlBMwwJXF24xYPqzfi3QS91SG?autoplay=1&cols=180&rows=40) (tracking branches, and "local" remotes)

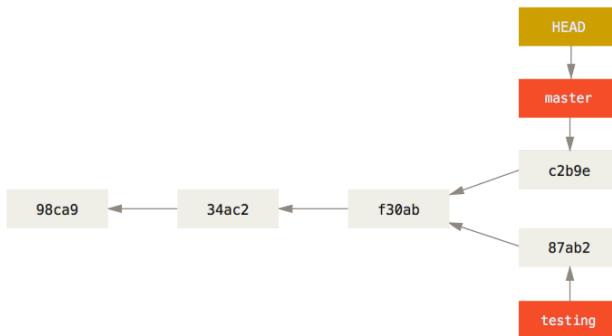
## Most of the time, just one remote (probably GitHub)

- GitHub is just another remote, very useful for collaboration, but just another remote
- Creating remotes in other machines or even in different directories as we did can be very useful. I use it regularly:
  - one PC has required GUI software,
  - the other PC required documentation software.
  - I need to push changes quickly from one to the other without making many commits in the common repository

*Now you understand branches (local and remotes) properly, but with several branches trouble will inevitably knock on your door at some point...*

## The problem: divergent branches (applies to local and remote branches)

- Earlier we created a *testing* branch and committed some changes.
- Later we went back to the *master* branch.
- At that point, if you commit to master, you end up with divergent branches.
- If you want to incorporate the changes in *testing* to *master*, you might have conflicts (maybe both commits updated the same function).



## Two possible solutions to divergent branches

### **git merge**

Incorporates changes from the named commits (since the time their histories diverged from the current branch) into the current branch.

### **git rebase (*we will not see this one today*)**

Apply all changes made in the current branch on top of another branch. (*This is a more advanced command, and you have to be careful with it*)

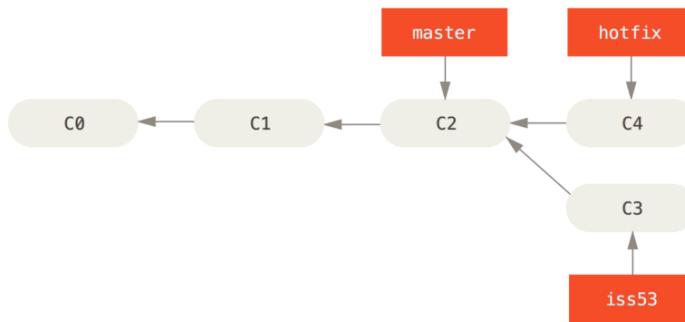
## **5. Merging**

## Merging

- Fast-forward merges
  - the simplest, we already saw an example of this in *demo-1-4.cast*
- No-conflict merges
  - very simple, and very common if working on your own
- Merges with conflicts
  - very usual when collaborating with others, specially if long time between commits

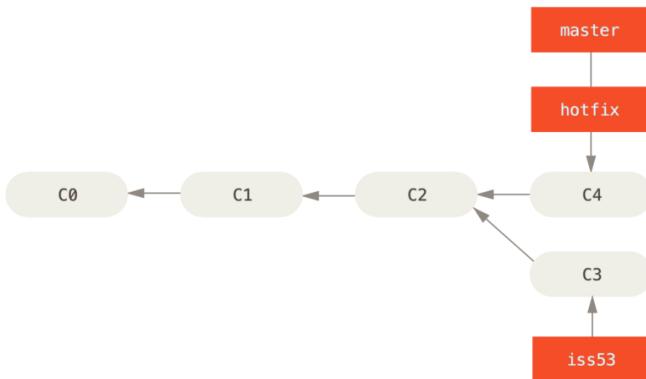
## Fast-forward "merge"

- Imagine you are in a situation like this, where:
  - you have two branches, started off the *master* branch, where you made one commit to each.
  - you want now to incorporate to *master* the changes done in branch *hotfix*



## Fast-forward "merge" (2)

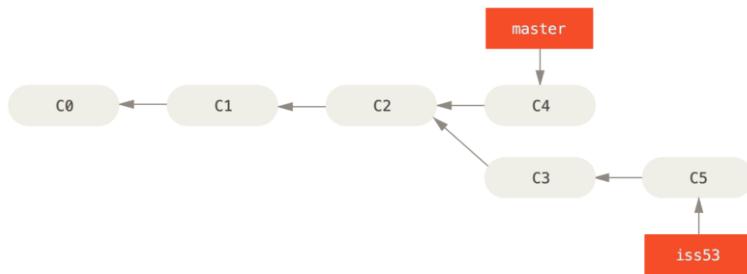
- Check out the master branch and merge the hotfix branch
  - This is a fast-forward merge (i.e. nothing really to merge, not divergent branch)



[demo-1-7.cast \(\[https://asciinema.org/a/ZbFhHkGvVaIGWF3HiF9ZiSyUG?  
autoplay=1&cols=180&rows=40\]\(https://asciinema.org/a/ZbFhHkGvVaIGWF3HiF9ZiSyUG?autoplay=1&cols=180&rows=40\)\)](https://asciinema.org/a/ZbFhHkGvVaIGWF3HiF9ZiSyUG?autoplay=1&cols=180&rows=40) [Note the *behind*, *ahead* comments when doing `git  
branch -vv` ]

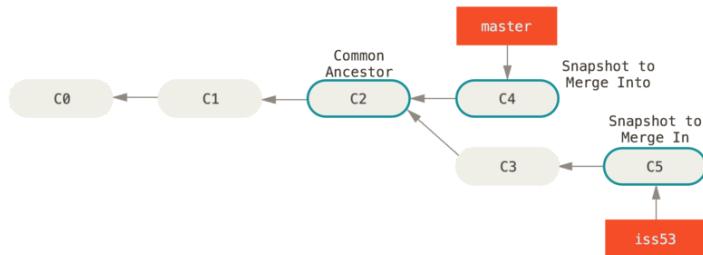
## No-conflicts merge

- Following from the previous situation:
  - you now have a branch *iss53* that has diverged from the *master* branch
  - a "fast-forward" is not possible when incorporating those changes to *master*



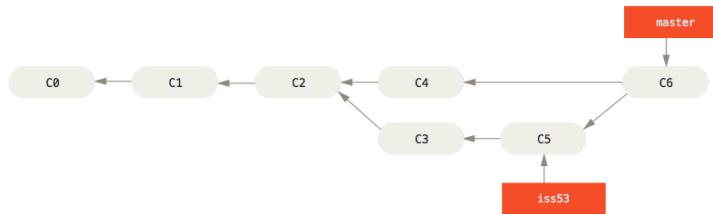
## No-conflicts merge (2)

- A git merge command will have to work harder now to find:
  - which commit is common to both lines
  - what changes were made in each branch so as to collect all changes
  - if changes in, e.g. different files/functions, merge possible w.o. conflicts



## No-conflicts merge (3)

- And we end up with a new "merge" commit in *master*:



**Note:** remember that `git merge iss53` means: merge branch `iss53` to my current branch (i.e. you want to issue the merge command with the destination branch checked-out)

[demo-1-8.cast \(https://asciinema.org/a/4Slf3zhP3K0ds9I5SdmHXUe8d?autoplay=1&cols=180&rows=40\)](https://asciinema.org/a/4Slf3zhP3K0ds9I5SdmHXUe8d?autoplay=1&cols=180&rows=40)

## Merge with conflicts

**When git cannot merge automatically, it will tell you:**

```
$ git merge iss53
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
Automatic merge failed; fix conflicts and then commit the result.
```

**The conflicting files will have conflict-resolution markers:**

```
<<<<< HEAD:index.html
<div id="footer">contact : email.support@github.com</div>
=====
<div id="footer">contact us at support@github.com</div>
>>>>> iss53:index.html
```

## Merge with conflicts (2)

- To resolve the conflict by hand:
  - just edit the file with your usual text editor, leaving the correct resolution (and no markers)
  - run `git commit` when all conflicts have been resolved

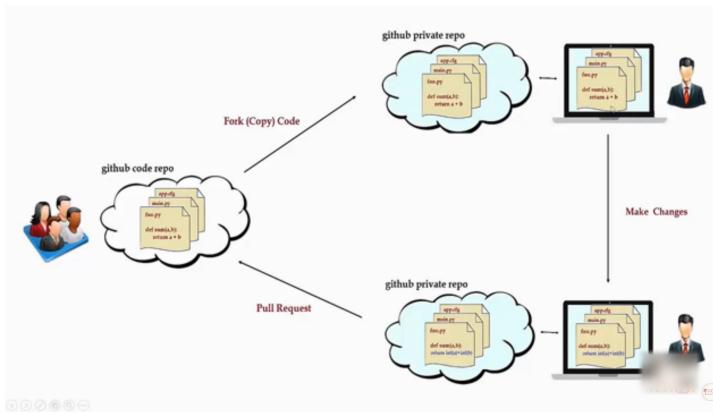
[demo-1-9.cast](#)

([https://asciinema.org/a/MxjPsupBk3toHCQsOuEQBKbMF?  
autoplay=1&cols=180&rows=40](https://asciinema.org/a/MxjPsupBk3toHCQsOuEQBKbMF?autoplay=1&cols=180&rows=40))

- But if collaborating regularly you should really learn how to use an external tool
  - `git mergetool` will tell you options and how to configure git
  - GitAhead demo: <https://www.youtube.com/watch?v=W-FHwUwE84M>  
(<https://www.youtube.com/watch?v=W-FHwUwE84M>).
  - Emacs + Magit + Ediff demo: [https://youtu.be/S86xsx\\_NzHc](https://youtu.be/S86xsx_NzHc)  
([https://youtu.be/S86xsx\\_NzHc](https://youtu.be/S86xsx_NzHc)).

## Pull/merge requests (GitHub)

- Very useful when contributing code when not a collaborator
- (but it can also be used across branches with collaborators)



From <https://www.youtube.com/watch?v=e3bjQX9jIBk>  
(<https://www.youtube.com/watch?v=e3bjQX9jIBk>),

## 6. Other useful Git configuration/commands/tools

- `.gitignore` (patterns of files that Git will simply ignore)
- `rebase` (apply changes of a branch onto some other branch)
- `reset` (discard commits in a private branch or throw away uncommitted changes)
- `cherry-pick` (apply the changes introduced by some existing commits)
- `bisect` (use binary search to find the commit that introduced a bug)
- `blame` (show what revision and author last modified each line of a file)
- `workflows` (git is very (too?) flexible. Need to decide on a sane workflow)
- `submodules`

## References

- [Git main page \(https://git-scm.com/\)](https://git-scm.com/),
- [Pro Git book \(https://git-scm.com/book/en/v2\)](https://git-scm.com/book/en/v2),
- [Git cheatsheet \(https://ndpsoftware.com/git-cheatsheet.html\)](https://ndpsoftware.com/git-cheatsheet.html).