

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*

May 12-13, SUNDIAL



Index

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

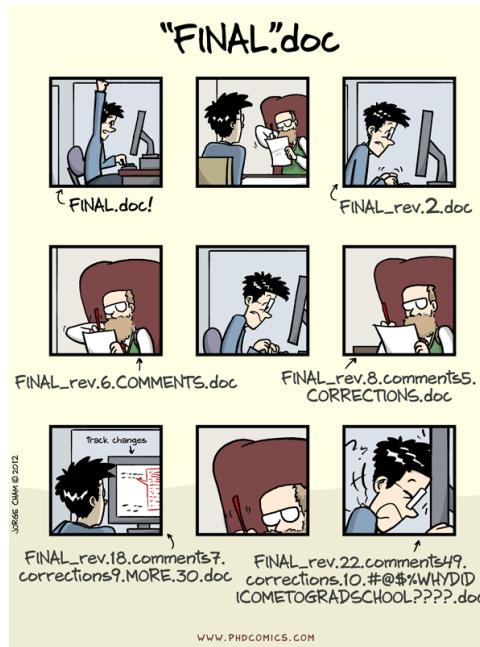
1. Git basics

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

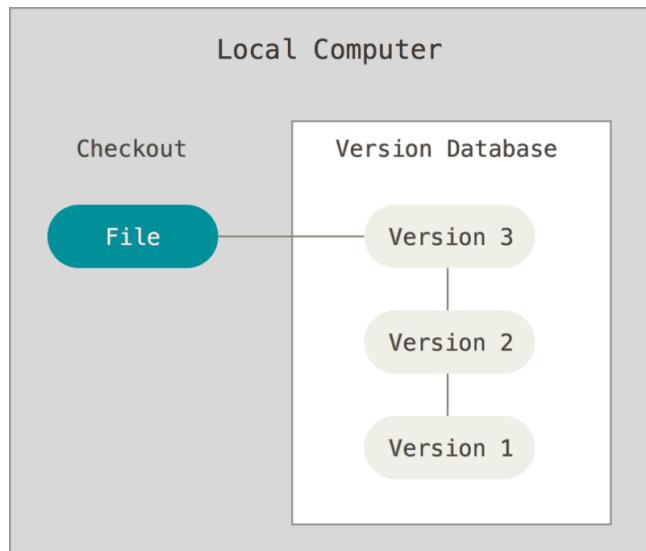
- What is version control?
- What do we version control (text vs. binary files)
- What is a version control system
- You have used a version control system directly or indirectly (e.g. Dropbox, Overleaf, etc.)
- Version control ideas: record/playback, branching, merging
- Benefits of version control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

Types of Version Control Systems

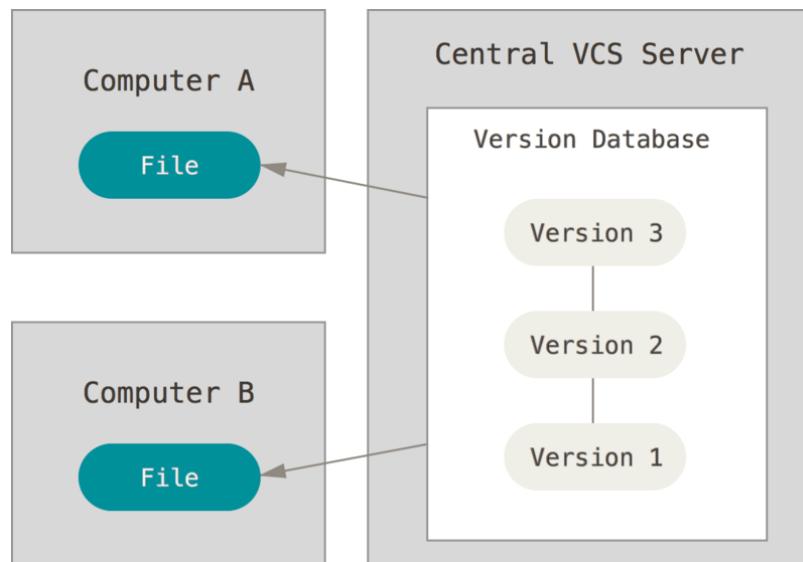
- No VCS (*don't be the one doing this!*)



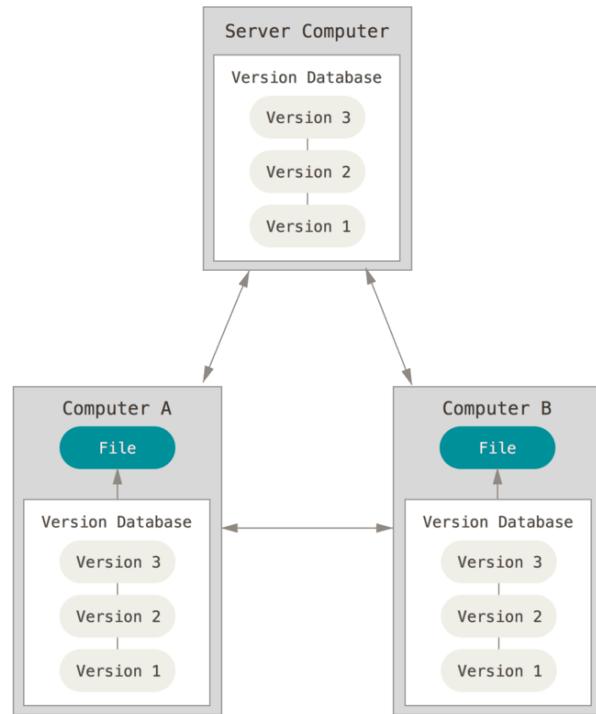
- Local VCS (e.g. RCS)



- Centralized VCS (e.g. Subversion)



- Distributed VCS (e.g. Git)



I'm assuming that:

- You are *familiar* with Git
- About Git: <https://git-scm.com/> (<https://git-scm.com/>) (downloads, documentation, etc.)
- You have *git* installed in your computer
- You have a *GitHub* account (<https://github.com/> (<https://github.com/>))
- You are using GNU/Linux (small variations if using Mac OS X or Windows)
- Much more than what we'll do today:
 - Pro Git book: <https://git-scm.com/book/en/v2> (<https://git-scm.com/book/en/v2>) (most images taken from here)

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"
```

Sample settings

```
In [16]: ! git config --list ## --show-origin
```

```
user.email=angel.de.vicente@iac.es
user.name=Angel de Vicente
color.ui=true
color.status=auto
color.branch=auto
credential.helper=cache --timeout=28800
diff.jupyter notebook.command=git-nbdiffdriver diff
merge.jupyter notebook.driver=git-nbmergedriver merge %0 %A %B %L %P
merge.jupyter notebook.name=jupyter notebook merge driver
difftool.nbdime.cmd=git-nbdifftool diff "$LOCAL" "$REMOTE" "$BASE"
difftool.prompt=false
mergetool.nbdime.cmd=git-nbmergetool merge "$BASE" "$LOCAL" "$REMOTE" "$MERGE
D"
mergetool.prompt=false
core.repositoryformatversion=0
core.filemode=true
core.bare=false
core.logallrefupdates=true
remote.origin.url=https://github.com/angel-devicente/git-workshop.git
remote.origin.fetch=+refs/heads/*:refs/remotes/origin/*
remote.pushdefault=origin
branch.master.remote=origin
branch.master.merge=refs/heads/master
```

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

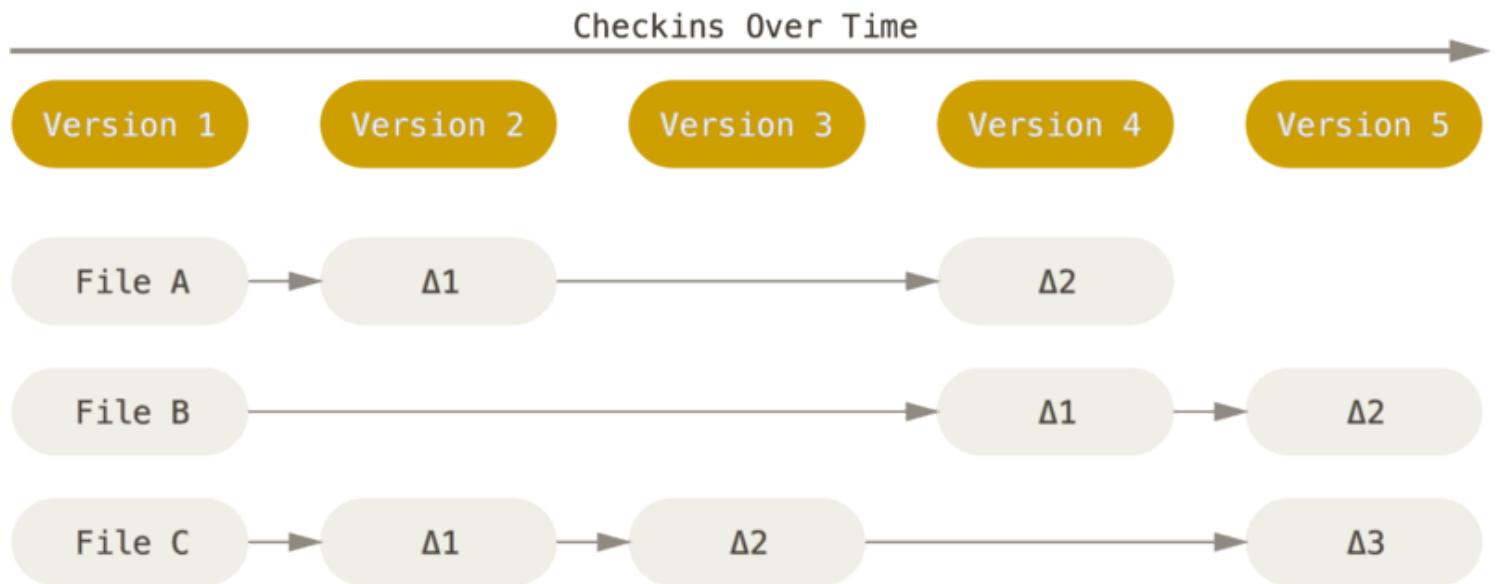
Show the branch and tracking info even in short-format.



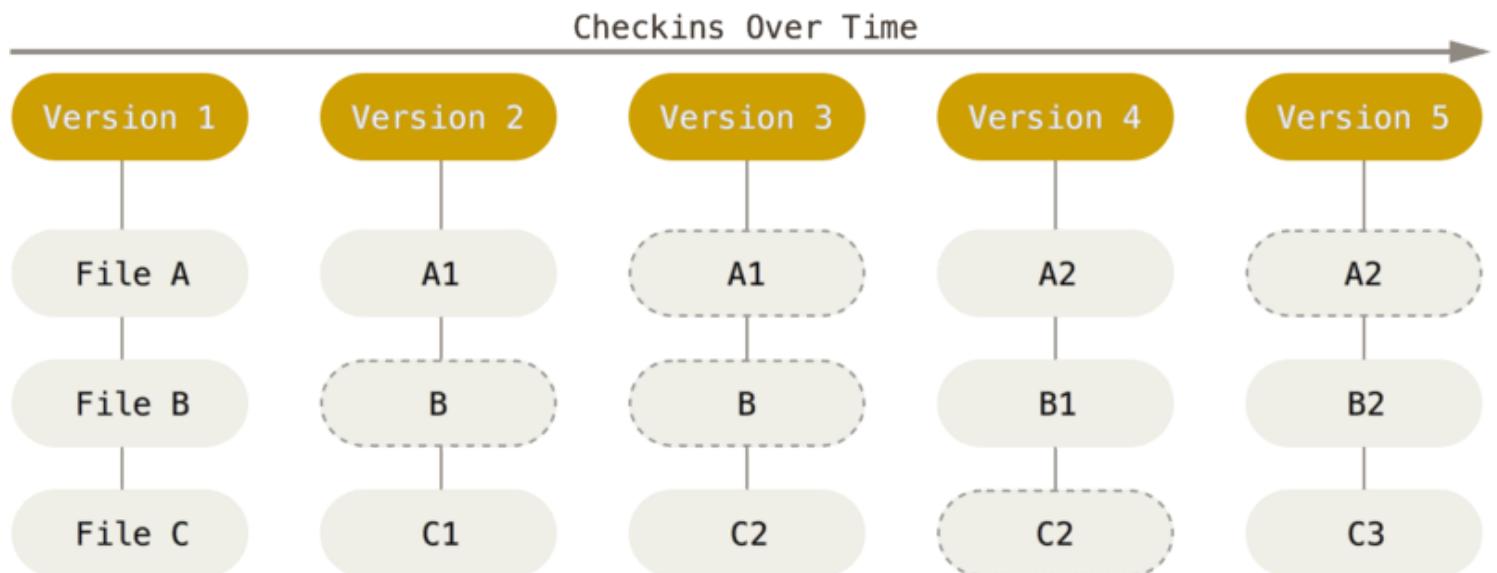
Main ideas in Git

- 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

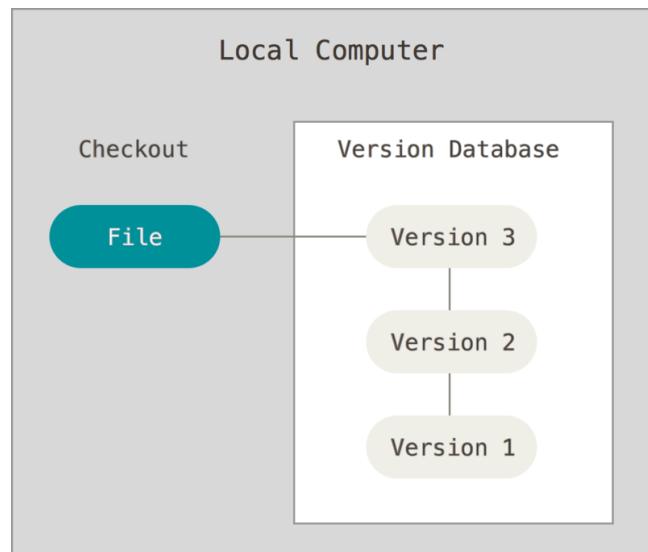


- Snapshots (Git)



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]:

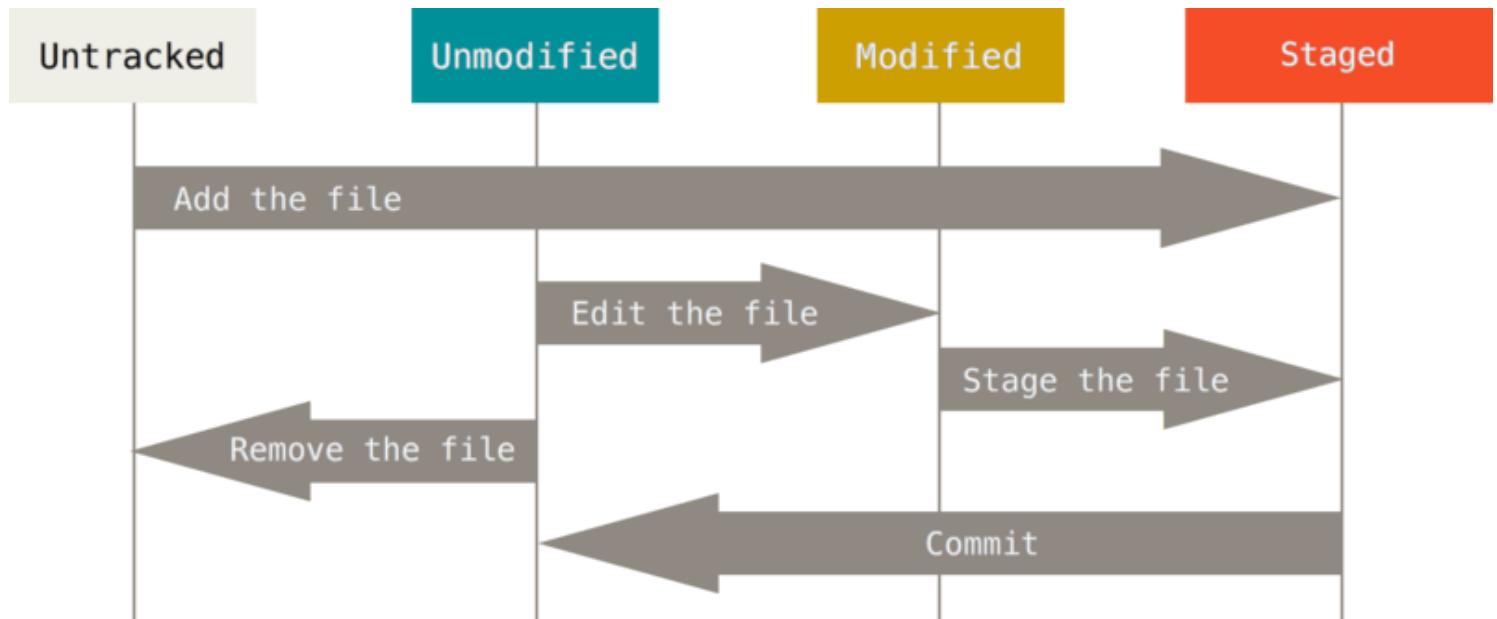
```
%%bash  
  
cd ../../Demos/Repos  
ls First_Demo/.git
```

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

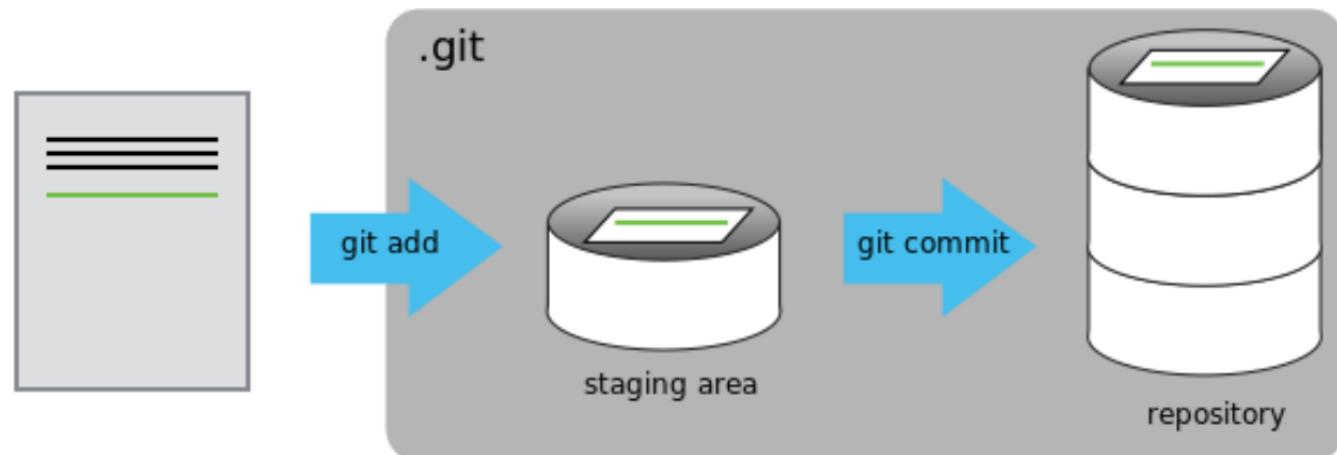
Recording changes

- 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)

- Commit messages are important (*don't do this*)

```
ae69e17 * last
e6c4dc1 * last
6ee6e4d * last
0df6fcf * last
eacd473 * thesis
56b21d2 * last
c17ac01 * thesis LAST INDEEDgit statusgit statusgit statusgit status sent to referees this version
42e4c33 * last tt
e2b0b24 * last thesisgit statusgit status
e32e4e6 * tesis
aef739c * last
0481ed3 * last hopefully
961b606 * last version BEFORE last round
e04d68b * t
904d9e7 * crap thesis last
f1bcf71 * last
305a89a * th
e644f77 * th
fe20449 * th
93ada46 * added png
5bac6ff * cap 6
6f14455 * finalslopegit status
19a6ba7 * th
1431e0c * d
9cbebef * merge
  \
54ec53b | * hh
3657531 * | th
  \
ff37a06 * th
550a99d * th
997d824 * mergmergee
  \
e7f685d | * last
e5d1f8d * | last
  \
ec0ef72 * added app zdif
b8844e0 * th
3b8c792 * t
1-UUU:@%%--F21 magit-log: finalpaper Top (1,0)      (Magit Log ivy Projectile yas) 11:22 0.35 Mail
```

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](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) ([https://git-scm.com/book/en/v2/Git-Basics-Undoing-
Things](https://git-scm.com/book/en/v2/Git-Basics-Undoing-Things))))

Practice time



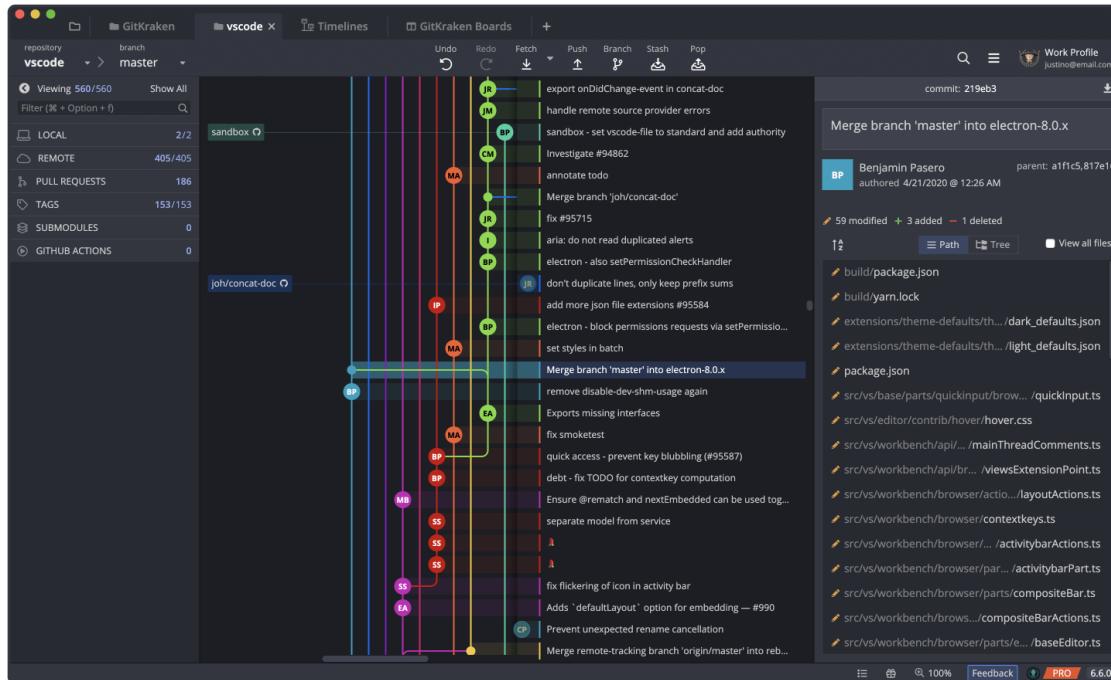
- Get some repository with plenty of history (we'll see `git clone` later)
 - `git clone https://github.com/jlord/git-it-electron.git`
 - `cd git-it-electron`
- How many commits in total are made by Brendan Forster?
- Find the date and the hash of the commit by Nikolai Plath where "GitHub" is mentioned in the commit message.
- Undo the change to the `README.md` file made by him.
 - *Hints:*
 - to find what was changed in a commit you can use `git show`. Get help for this command.
 - to revert changes do it manually or check the command `git revert`
- Add files `BUGS.md` and `FIXES.md` and commit with message "Fixes"
- Add some content to `BUGS.md` and amend the commit with message "Fixes and bugs"
- Rename `BUGS.md` to `BUGS_FIXES.md` and delete `FIXES.md` and commit again (amending)



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 GitHub desktop application interface. The main window title is "gitahead - master - behind: 2 (origin/master)". The left sidebar displays a timeline of commits from Jason Haslam and Shane Gramlich, with the current commit being "Make a really great change." by Jason Haslam on 2/6/17. The right side shows the "Commit Message" field containing "Make a really great change." and a list of staged files: "conf/editor.lua", "src/app/GitHead.cpp", and "src/app/GitAhead.cpp". The "src/app/GitAhead.cpp" file is expanded, showing code changes related to font family and size. The bottom status bar provides information about the repository's state and recent activity.

```

gitahead - master - behind: 2 (origin/master)

Branch: master
Commit Message: Make a really great change.
1 of 3 files staged, 1 file partially staged
Stage All
Unstage All
Commit

Jason Haslam c02358e Merge branch 'ExternalTools'
Jason Haslam 1d8e824 v1.1.4 Changed version to 1.1.4.
Jason Haslam c054b29 Updated libgit2 to tip of master.
Jason Haslam 6998810 Simplify and tweak toolbar button style.
Jason Haslam cea72bd Set toolbar button colors through the palette and use the button s...
Shane Gramlich 112e8ac Merge branch 'master'
Shane Gramlich f347b5f Remove failed on start error

12:38 PM - Reset - master to 6a82e89
12:38 PM - Pull - master from origin
Fetch - origin
  Everything up-to-date.
Fast-forward - master to origin/master
  Unable to fast-forward 'master' - 1 conflict prevents checkout
    src/cli/installer_mac.mm
      You may be able to reconcile your changes with the conflicting files by stash 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/>)

```

e800d9c * master origin/master Merge pull request #257 from jotoeri/add-guilink
7934d09 \| * Adding a link to the GUI overview
jotoeri      1 year
es749bd \| Merge pull request #252 from 0x15F9/master
jessica.lord 1 year
fb309c4 \| * Fixed slight typo
0x15F9      1 year
fc64b22 \| Merge pull request #242 from franciosi/patch-2
jessica.lord 1 year
9eba724 \| * Update README.md
franciosi    1 year
eac0502 \| Merge pull request #230 from juanaruiz/patch-1
jessica.lord 2 years
67d5dee \| * Cambiando "and" por "y"
juanantonio.ruiz. 2 years
aede755 \| * Asegurante - Asegurate
jessica.lord 2 years
be2594f \| Merge pull request #234 from diegocasillas/spanish-translati
jessica.lord 2 years
\\
0d353b0 \| Fix typos in 11_merge_tada.html
diegocasillas 2 years
3d627cb \| Fix typos in 10_requesting_your_pull_request.html
diegocasillas 2 years
1b3256b \| Fix typos in 9_pull_never_out_of_date.html
diegocasillas 2 years
2966499 \| Fix typos in 7_branches_arent_just_for_birds.html
diegocasillas 2 years
7090000 \| Fix typos in 6_forks_and_clones.html
diegocasillas 2 years
1c04891 \| Fix typos in 5_remote_control.html
diegocasillas 2 years
bdca4bc \| Fix typos in 4_githubbin.html
diegocasillas 2 years
\\
0d38029 \| 4.4.0 Merge pull request #231 from mberasategi/es_ES
jessica.lord 2 years
\\
\\
\\
\\
v1c59e3 \| Add es_ES to README
mberasategi 2 years
9f5145c \| Set es_ES for es alias
mberasategi 2 years
e78316a \| Add new locale to menus
mberasategi 2 years
3f7970a \| Translate 11-pages
mberasategi 2 years
1a1c1e6 \| Translate 11-commands
mberasategi 2 years
696c24d \| Translate 11-merge
mberasategi 2 years
1b70c0d \| Translate 10-requesting-pull-request
mberasategi 2 years
0b458bc \| Translate 9-pull
mberasategi 2 years
ebc7795 \| Translate 8-small-world
mberasategi 2 years
3f00000 \| Translate 7-clones
mberasategi 2 years
e1f50515 \| Translate 6-forks-clones
mberasategi 2 years
10f50908 \| Translate 5-remote-control
mberasategi 2 years
05c4baee \| Translate 4-githubbin
mberasategi 2 years
47061d0 \| Translate 3-commit-to-it
mberasategi 2 years
06822d0 \| Change 3-commit-to-it to helamundo
mberasategi 2 years
0303005 \| Change Git Shall to Git CMD for Windows
mberasategi 2 years
0ff2317 \| Begin translating 3-commit-to-it
miren.berasategi 2 years
e16927f \| Translate 2-repository
miren.berasategi 2 years
6d9a218 \| Translate 1_get-git
miren.berasategi 2 years
c14966d \| Duplicate es_CO to jumpstart es_ES
miren.berasategi 2 years
\\
7aaea5c \| Add list of supported languages to readme
jessica.lord 2 years
573d0000 \| 4.3.0 version 4.3.0
jessica.lord 2 years
01f9070 \| make lang button wider to fit language names
jessica.lord 2 years
cfc0031 \| Merge pull request #217 from node-co/feature/spanish-translati
jessica.lord 2 years
\\
1-UUU:@%--F21 magit-logi git-it-electron 1op (5,0) (Magit Log ivy Projectile yas) 06:34 0.61 --- 99-UUU:@%--F21 magit-revision git-it-electron All (3,0) (Magit Rev ivy Projectile yas) 06:34 0.61
1-UUU:@%--F21 magit-logi git-it-electron 1op (5,0) (Magit Log ivy Projectile yas) 06:34 0.61 --- 99-UUU:@%--F21 magit-revision git-it-electron All (3,0) (Magit Rev ivy Projectile yas) 06:34 0.61
23.566% 2020-05-06 06:34:51

```

Live demo with Magit & GitAhead

- (see usefulness of hunks)

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

```
In [10]: %%bash
```

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

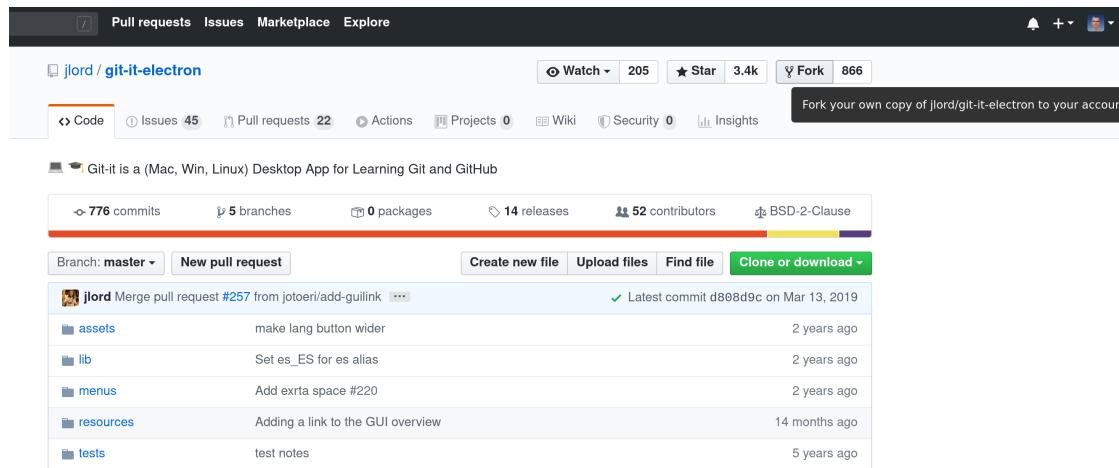
* remote origin
  Fetch URL: https://github.com/jllord/git-it-electron.git
  Push URL: https://github.com/jllord/git-it-electron.git
  HEAD branch: master
  Remote branches:
    IUTInfoAix-M2105-french-translation      tracked
    dependabot/npm_and_yarn/handlebars-4.3.0  tracked
    fix-menu                                    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`. The page includes standard GitHub navigation like Watch, Star, Fork, and Insights. Below the header, there's a brief description: "Git-it is a (Mac, Win, Linux) Desktop App for Learning Git and GitHub". The repository statistics show 776 commits, 5 branches, 0 packages, 14 releases, 52 contributors, and a BSD-2-Clause license. A pull request from `jlord` is visible. On the right, a modal window is open over the repository content, specifically focusing on the "Clone or download" section. This modal contains two cloning options: "Clone with HTTPS" (selected) and "Use SSH", along with the URL `https://github.com/angel-devicente/git-it-electron` and a "Download ZIP" link.

```
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 directory.  
  
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)
```

- Create a new file in the git-it-cp1 repository and push it to GitHub

[angel-devicente / git-it-electron](#)

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

[Watch](#) 0 [Star](#) 0 [Fork](#) 867

[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

Branch: master [New pull request](#) [Create new file](#) [Upload files](#) [Find file](#) [Clone or download](#)

This branch is 1 commit ahead of jlord:master. [Pull request](#) [Compare](#)

Angel de Vicente FORKED-COPY.md file added	Latest commit 501cc77 18 seconds ago
assets make lang button wider	2 years ago
lib Set es_ES for es alias	2 years ago
menus Add extra space jlord#220	2 years ago
resources Adding a link to the GUI overview	14 months ago
tests test notes	5 years ago
.gitattributes make PortableGit to be ignored in language stats	3 years ago
.gitignore Update .gitignore	4 years ago
.travis.yml update travis configuration to cache and use a later version of node	3 years ago
CONTRIBUTING.md Massive, messy merge. Fingers crossed.	3 years ago
FORKED-COPY.md FORKED-COPY.md file added	18 seconds ago



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

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

Watch 0 Star 0

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

Who has access

PUBLIC REPOSITORY This repository is public and visible to anyone.

DIRECT ACCESS 0 collaborators have access to this repository. Only you can contribute to this repository.

Manage

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

In [13]:

```
%%bash  
  
cd ../Demos/Repos  
if [ ! -d git-it-cp2 ] ; then  
    git clone https://github.com/angel-devicente/git-it-electron.git git-it-cp2  
fi
```

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  angel-devicente / Repository name * 

Great repository names are short, [git-it-test](#) is available. Get 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

- ... and then follow the instructions given by GitHub (reproduced below).
 - all steps should be familiar
 - but why *git push -u origin master*?
 - what happens if you just use *git push* as in the previous demo?
 - ... we need to learn more about branches

```
echo "# git-it-test" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/angel-devicente/git-it-test.git
git push -u origin master
```



4. Branches

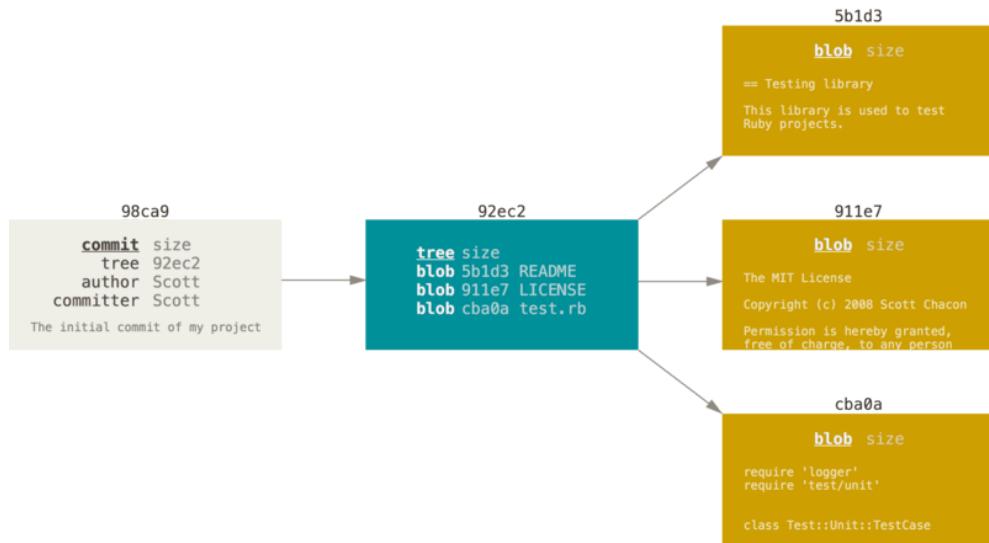
Branches

- 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.

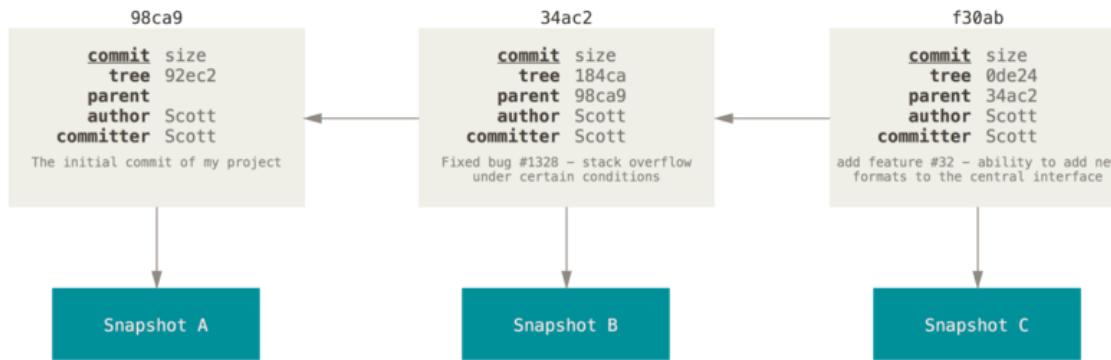
So, let's try and see some of the Git internals to understand what branches are...

(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>))

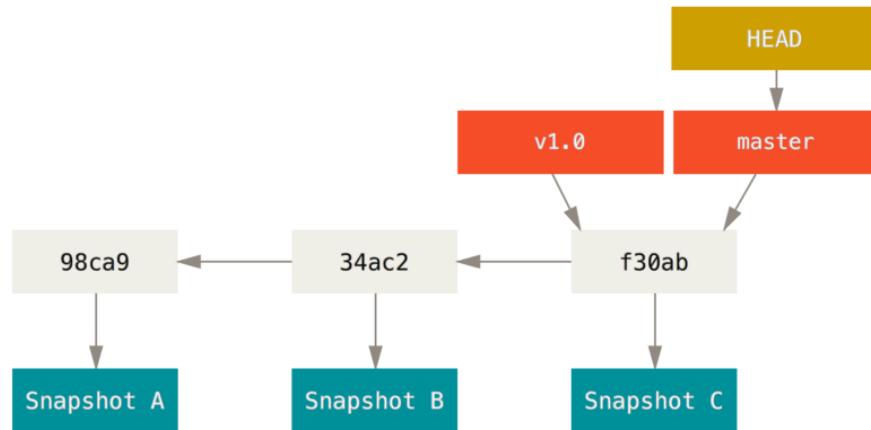
- A *commit* points to a *tree* object, and this to *blobs*
 - For us the important part is just the *commit*, which points to a *snapshot* of our work



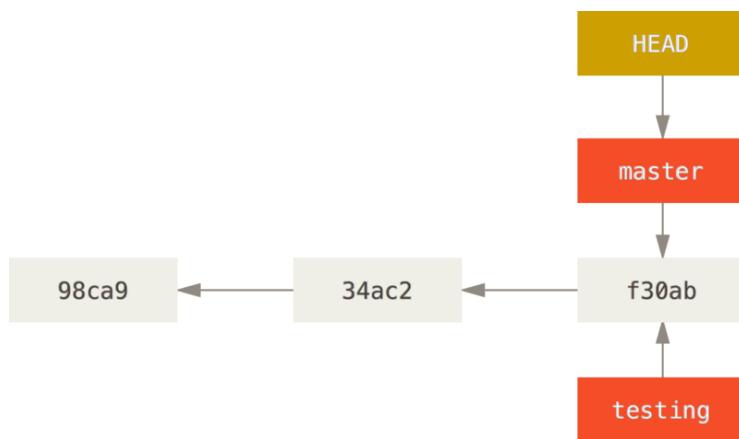
- Creating more *commits*, keeps a linked list
 - (which is basically what you see when you do `git log`)



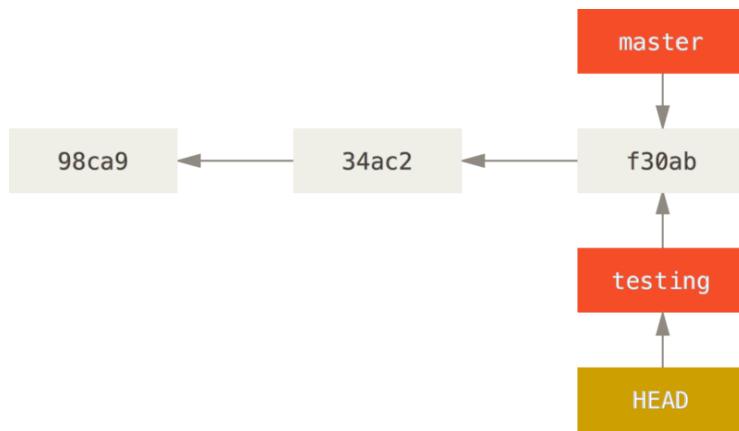
- A *branch* is just a pointer to one of these *commits*
 - *master* is created when you do `git init`, but it is no special
 - *HEAD* points to the branch we are working on



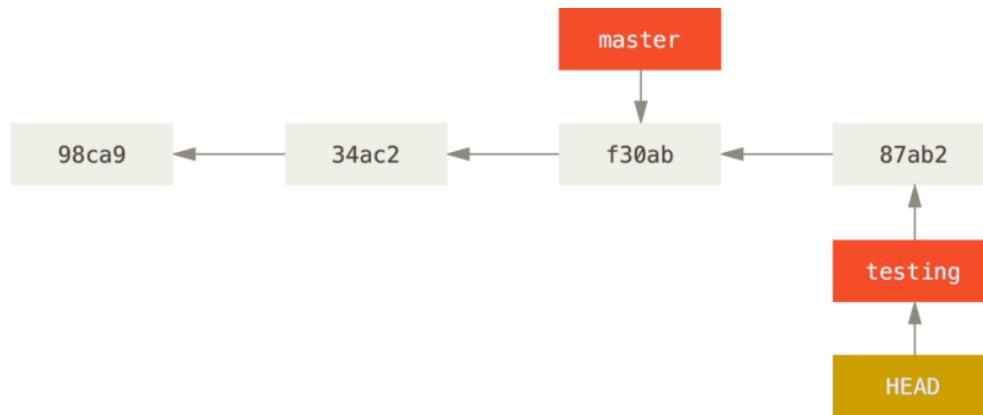
- Creating a branch
 - If we are working in *master* and we do `git branch testing`
 - *testing* branch is created, but we are still working on *master*
 - so you can see *HEAD* is still pointing to *master*



- 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.**

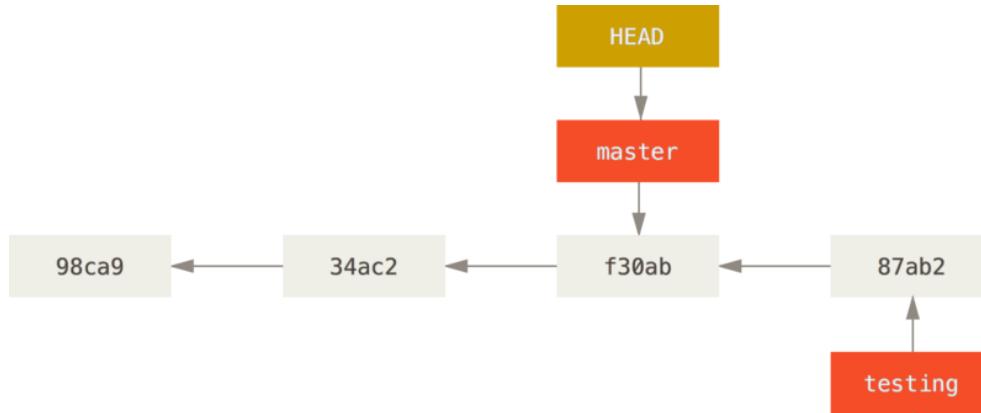


- Committing to *testing*
 - makes *testing* branch move forward
 - but *master* branch still points to the same commit



- Changing branches

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



Practice time



- 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)



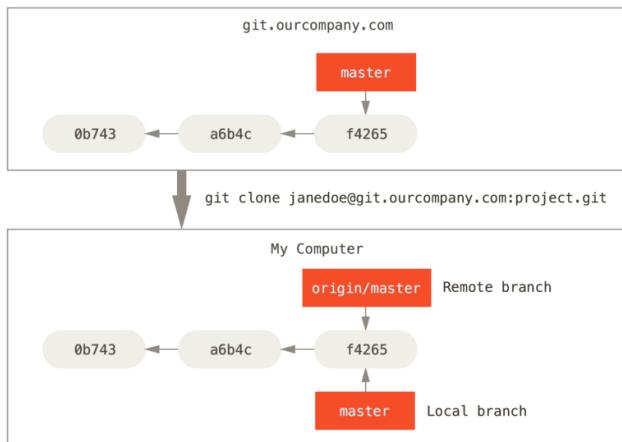
Remote Branches

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

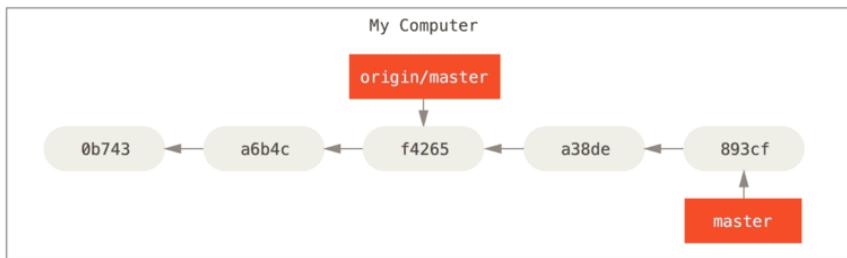
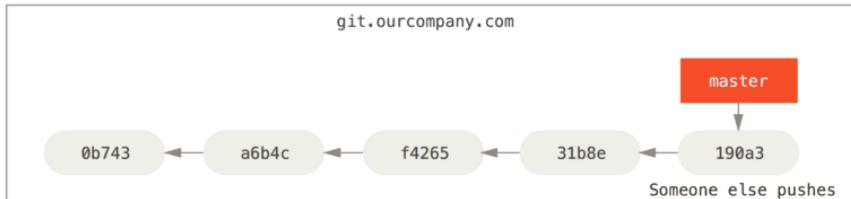
```
cd ../Demos/Repos/git-it-cp2
git remote -vv
echo
git branch --all
echo
git branch -vv
```

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

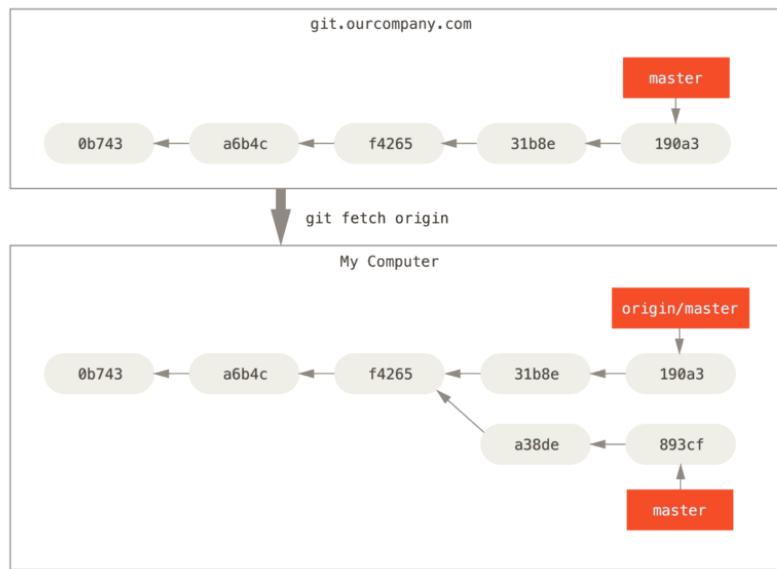
```
* master
  remotes/origin/HEAD -> origin/master
  remotes/origin/IUTInfoAix-M2105-french-translation
  remotes/origin/dependabot/npm_and_yarn/handlebars-4.3.0
  remotes/origin/fix-menu
  remotes/origin/lang-tweaks
  remotes/origin/master
```

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

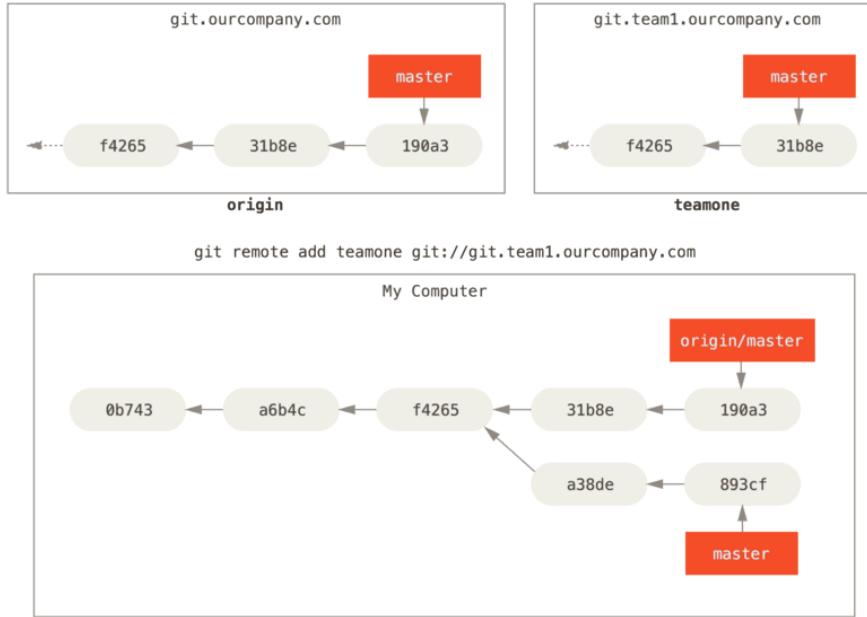
- Remote and local repositories can become unsynchronized
 - Here there were commits in the local branch and the remote branch
 - Note that the remote-tracking branch `origin/master` doesn't move (until you use `git fetch` your view of the remote repository is out-of-date)



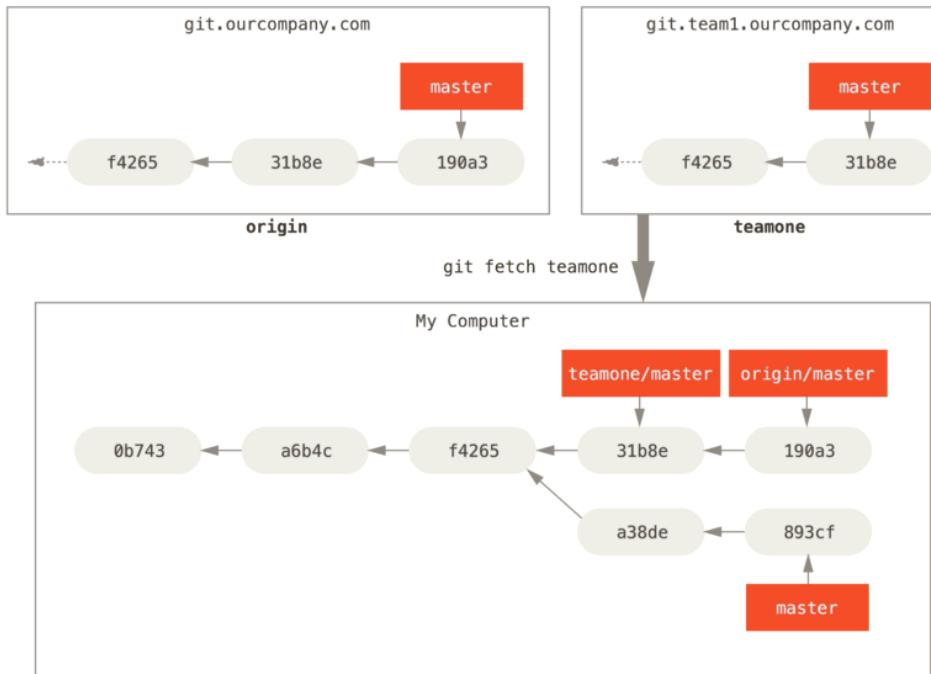
- `git fetch` will synchronize your remote-tracking branch
 - at this point `origin/master` and `master` have diverged (will have to use `git merge` or similar)



- 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)

Let's see a 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)



Practice time



- In `git-it-cp2` follow similar steps to the previous demo, so that:
 - you set the local repository `git-it-cp1` as a remote
 - checkout the branch `test_branch1`
 - commit some change to it
 - push it to the repository `git-it-cp1`
 - if you just do `git push` it should work, (do you understand why?)
 - when you are not in a tracking branch you can also push it to a remote branch, (do you understand how?)



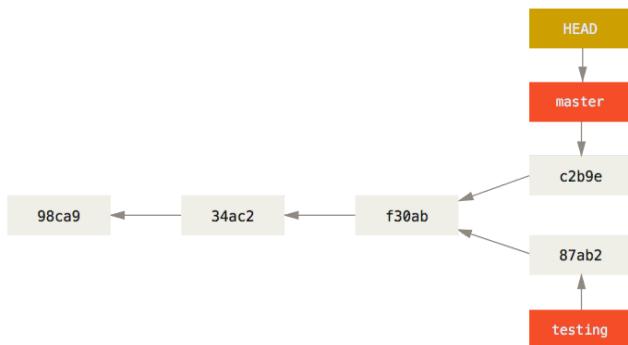
Most of the time, just one remote (probably GitHub)

- 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)

- Before 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` (*see Section 7*)

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*)

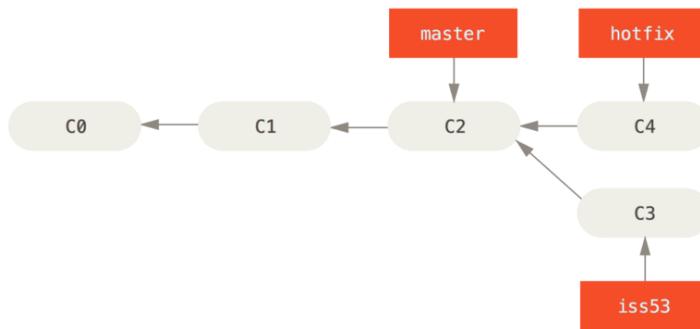
5. Merging

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

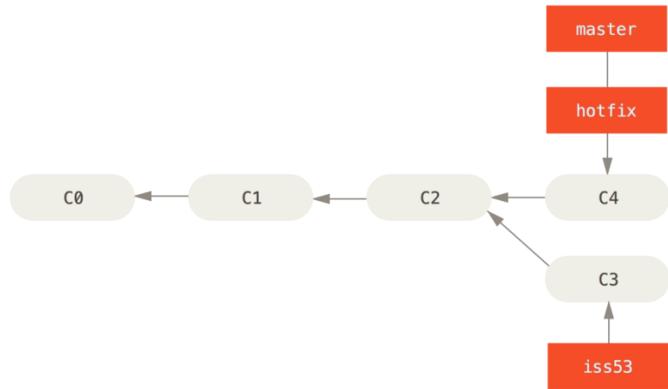
Fast-forward "merge"

- 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)

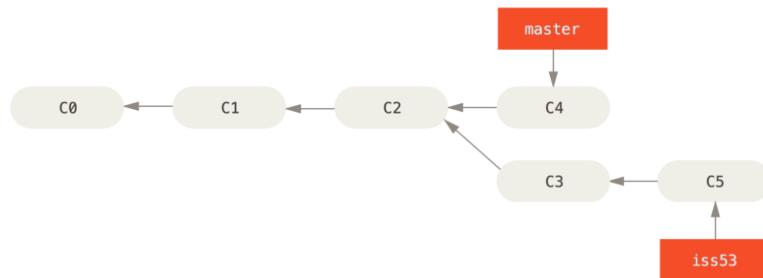
- Check out the master branch and merge the hotfix branch
 - This is a fast-forward merge (i.e. nothing really to merge, not divergent branch)



[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) [Note the *behind*, *ahead* comments when doing `git branch -vv`]

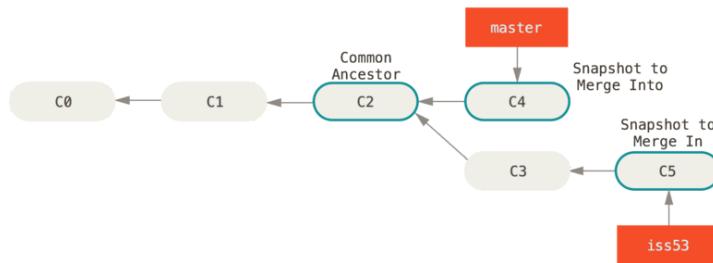
No-conflicts merge

- 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*



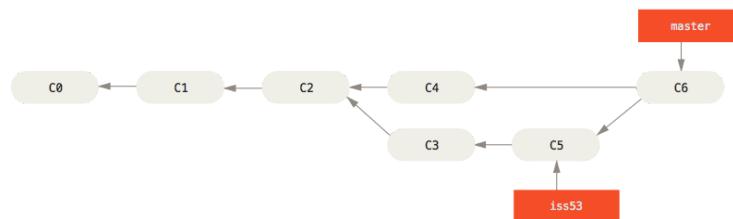
No-conflicts merge (2)

- 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 \(https://asciinema.org/a/4SIf3zhP3K0ds9I5SdmHXUe8d?autoplay=1&cols=180&rows=40\)](https://asciinema.org/a/4SIf3zhP3K0ds9I5SdmHXUe8d?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
- 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>) (many more GitAhead videos at https://www.youtube.com/playlist?list=PLkhgTa3ULplz_DgalwtORMviEJeHy07EB (https://www.youtube.com/playlist?list=PLkhgTa3ULplz_DgalwtORMviEJeHy07EB))
 - Emacs + Magit + Ediff demo: https://youtu.be/S86xsx_NzHc (https://youtu.be/S86xsx_NzHc).

Practice time



- First make sure you synchronize the *master* branch in *git-it-cp1* and *git-it-cp2* with GitHub
 - If you followed the examples above,
 - *git-it-cp1* is synchronized with *origin/master*
 - *git-it-cp2* has un-pushed commits. Push those ones to GitHub
 - then pull them from *git-it-cp1*
 - In any case, make sure that *master* in *git-it-cp1*, *git-it-cp2*, and *origin/master* are in the same state.
- Make a commit (adding one line to a file) in *git-it-cp1* and push to GitHub
- In *git-it-cp2* make a commit (adding one line to the same file above) and push to GitHub
 - You won't be able to do it, since now the remote branch is ahead, so you will have to do a *fetch*
 - Then *merge origin/master*
 - And now you should get conflicts. Solve the conflicts and push to GitHub
- Back in *git-it-cp1*, pull (or fetch/merge) the changes that you just did in *git-it-cp2*



Possible solution

[demo-1-10.cast](#)

(<https://asciinema.org/a/1tHAh1ygCDswdITaG64KPuk91?autoplay=1&cols=180&rows=40>)

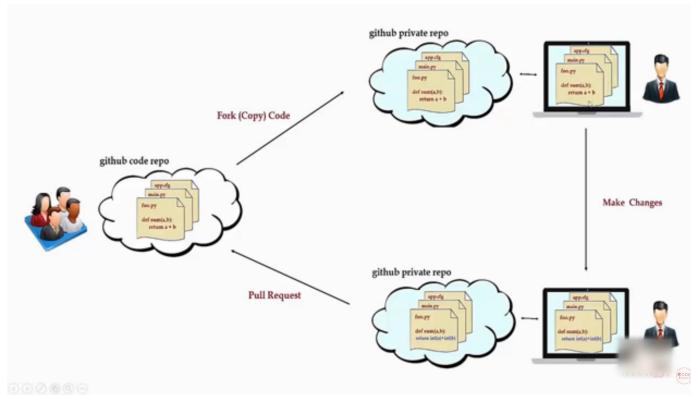
The screenshot shows a GitHub repository page for `git-it-electron`. The repository was forked from `jlord/git-it-electron`. The master branch is selected. A list of commits is shown, all made by Angel de Vicente:

- Changes by cp1 and cp2. Conflicts solved (commit `16f6c47`)
- Change by cp2 (commit `0ba3466`)
- Change by cp1 (commit `af58f62`)
- Merge branch 'iss53' after conflict solved (commit `e23c002`)

Each commit has a copy icon, a commit hash, and a diff icon.

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>).

Pull request GitHub live demo

- Pull request all inside GitHub
- Pull request via local branches for more advanced control
- Keeping your fork-ed version up-to-date with the original (*upstream*)

6. Workflows

Workflows

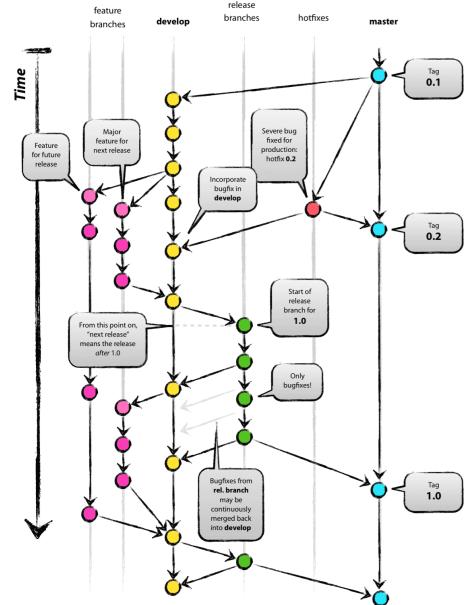
- Branches are a *killer* Git feature and you should use them often
 - create a branch to fix bugs
 - create a branch to test new ideas
 - etc.
- But you need to get organized with workflows or things will get messy quickly. Useful commands:
 - `git branch --merged`
 - `git branch --no-merged`
 - `git branch -d <branch>`

Messy history

- Our own project, only two/three active developers

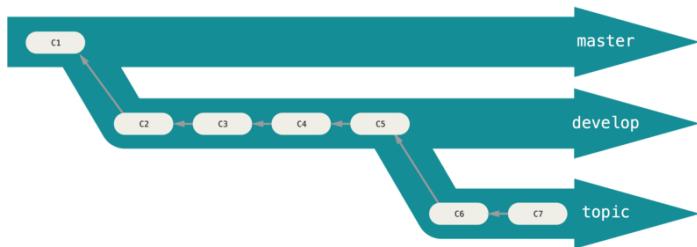
```
cb303c0 | /| Fix:
c0dbc39 | * | Merge branch 'multilevel-zeeman' into public_porta
| |\ \
0feb0d7 | | * | Feature and some fixes:
842fc66 * | | | Issues #36 and #42. Changes and bugfixes in PORTA GUI
| / /
c5f887f | * | | Issue #36: HDF5 routines for the continuum module implemented.
d16de05 * | | Removed code for biquadratic interpolation.
704e1da * | | Removed commands + params related to MultiGrid
d29c288 * | | Small fixes to reading commands
9e7c356 * | | Issue #32
| / /
4bf3b20 * | Merge branch 'public_porta' of https://gitlab.com/angelyv/PORTA-private into public_porta
| \ \
a7e0d7e | * | Fix issue #38 to avoid skipping last character of command argument
0e41d90 * | | minor changes to User Manual + makefile changes for PDF manual creation
8c0b857 * | | Minor changes to documentation
| / /
5b90422 * | Documentation page for the BESSER formal solver properly formatted
82fdae6 * | Fixed issue #34
c903e2a * | Docs
e1b8413 * | Merge branch 'public_porta' of ssh://gitlab.com/angelyv/PORTA-private into public_porta
| \ \
d606b3b | * Cleaned up temp directory after merge of issue11 branch
47673eb | | Merge branch 'issue11' into public_porta
| | \
7481b0a | | * This one works nice with a Thread for I/O. It seems to work quite nicely
fd2fd74 | | * First version. Much nicer than the standard PORTA way, but it doesn't really work properly yet.
6ddcd7e * | | Docs. Also, no math in sphinx compilation right now?
| / /
```

- Example workflow (<https://nvie.com/posts/a-successful-git-branching-model/> (<https://nvie.com/posts/a-successful-git-branching-model/>))



Many types of workflows

- see, for example, <https://git-scm.com/book/en/v2/Git-Branching-Branching-Workflows> (<https://git-scm.com/book/en/v2/Git-Branching-Branching-Workflows>),
- for our development of PORTA (<https://gitlab.com/polmag/PORTA> (<https://gitlab.com/polmag/PORTA>)), we decided to have something similar to:



Clean history

```
Commits in master
eca2325 * master origin/master Merge branch 'development'
c6afdf6e | * Fixing issue #56 (some ERRORS should be moved back to WARNINGS)
da866c0 * | Merge branch 'development' 161.72.206.34:8888/notebooks/Work/POLMAG/Outreach/Pres
|\ \
2e8b12e | * Shrink in GUI fixed (#55) + small DEVS changes (ready to v1.0.2)
e51d4b9 * | Merge branch 'development'
|\ \
14f59a4 | * Small changes, ready to bump up to version 1.0.1
c4f2602 | * Added information on compiling/running in several supercomputers
41b96fc | * Changes to makefiles/configure + fixed bug to use PGI/Cray compilers
06af922 | * Matika.c slimming operation + minimal changes to PORTA_DEVS_ONLY
af8887b * | Merge branch 'development'
|\ \
74213d4 | * Small corrections when making public
b6b9f93 | * Updated information on how to make public version
d6ef60f * | Merge branch 'development'
|\ \
1a6930d | * Stable version: final touches before merging to master
c6fdbed | * Merge branch 'issue-52-bug-in-contpol-source-function' into development
| | \
534e096 | | * issue-52-bug-in-contpol-source-function Bug in source function (issue #52)
| | /
1d7a3ed | * Doc change:
8d3e84e | * Added parallel_debugger directory
```

Our workflow for PORTA

- Three long-lived branches: *release*, *master*, *development*, plus short-lived topic branches.
- After merged to *development* we delete the topic branch.
- Small issues directly to *development*. Larger issues have a topic branch, and we follow it up via an *issue* (in GitLab).
- Developers in the team have direct access only to *development*
- The repository maintainer (i.e. me) takes care of merging into *master* (stable versions) and *release* (ready to become public versions)
- To keep a clean history, we make sure that topic branches (**rebasing**):
 - can be merged as fast-forward
 - but we merge it as no fast-forward to keep a more readable history

Issues in GitLab (very similar in GitHub)

Angel de Vicente > PORTA-private > Issues > #55

Closed | Opened 1 month ago by Angel de Vicente | Reopen issue | New issue

Bug in GUI - X axis labels not changed when "shrink" is chosen

For XY and slit plots, when the "Shrink" option is chosen, the x labels should shrink accordingly, but at present the labels are not modified.

Need to change in all branches (and make public version v1.0.2), plus also change the figures in the documentation (see, for example https://polmag.gitlab.io/PORTA/Notes_Implementation/index.html#long-characteristics-formal-solution)

Oldest first | Show all activity

Discussion 1 | Designs 0

Angel de Vicente @anglev changed due date to March 26, 2020 1 month ago

Angel de Vicente @anglev added label 1 month ago

Angel de Vicente @anglev mentioned in commit 2e8b12e 1 month ago

Angel de Vicente @anglev closed via commit 2e8b12e 1 month ago

Angel de Vicente @anglev - 1 month ago

Maintainer

Fixed in all branches (and made it to public version v1.0.2)

With the previous version, regardless of the Shrink option in the GUI, the XY plots would never shrink the labels in the X axis, and the slit plots would always shrink them. All the plots below are generated by creating two subplots, unchecking "Shrink" in the first one, and checking "shrink" in the second one.

$I [\text{erg Hz}^{-1} \text{cm}^{-2} \text{s}^{-1} \text{sr}^{-1}]$

6
5

0.000048
0.000042

Relate issues with logs

```
commit 0a8b1201781a219b0c0b08033d08b590bb880
Author: Angel de Vicente <angel.de.vicente@iac.es>
AuthorDate: Sun Mar 22 09:44:12 2020 +0000
Commit: Angel de Vicente <angel.de.vicente@iac.es>
CommitDate: Thu Mar 26 11:11:56 2020 +0000

Parent: 14f59a4 Small changes, ready to bump up to version 1.0.1
Merged: issue-27-automated-tests issue-50_automatic-formatting-src-clang-format
        issue-52-bug-in-contpol-source-function issue-53-contpol-multilevel-3D-tests
Contained: development master release
Precedes: v1.0.2 (19)

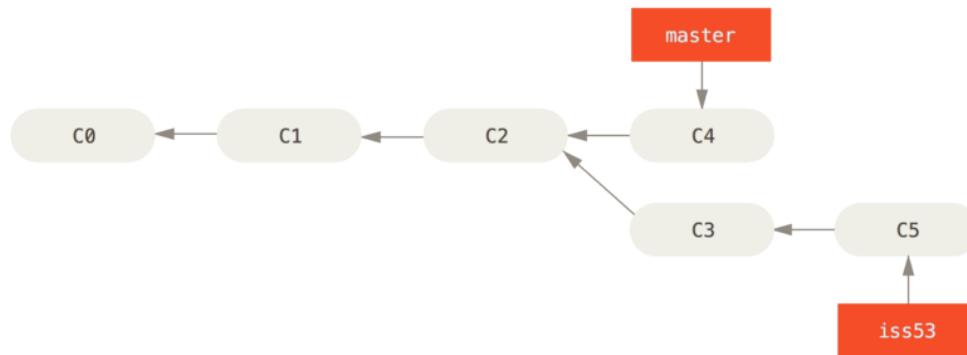
Shrink in GUI fixed (#55) + small DEVS changes (ready to v1.0.2)

+ Fixed issue #55
+ Modified (almost) all figures in visualization chapter to include active cue border
+ Modified figure in Long Characteristics section to include fixed labels in shrunk axis
+ Minimal changes to documentation on making PORTA public
+ Added basic scripts to convert from PMDv1 to PMDv2 for twolevel
```

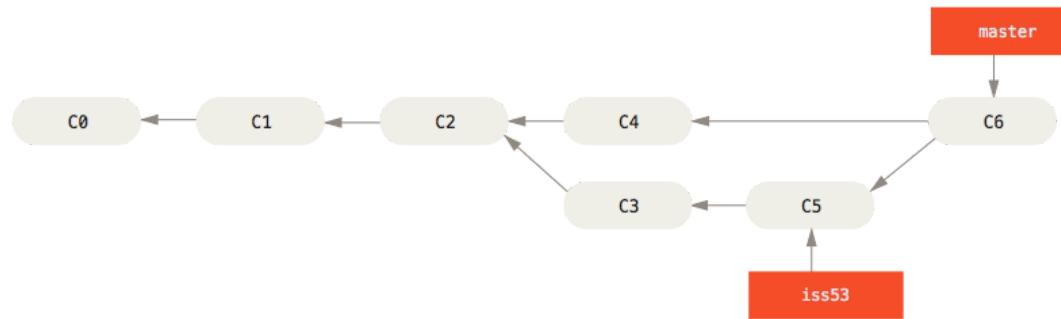
7. Rebasing

Why do we want to avoid regular merges?

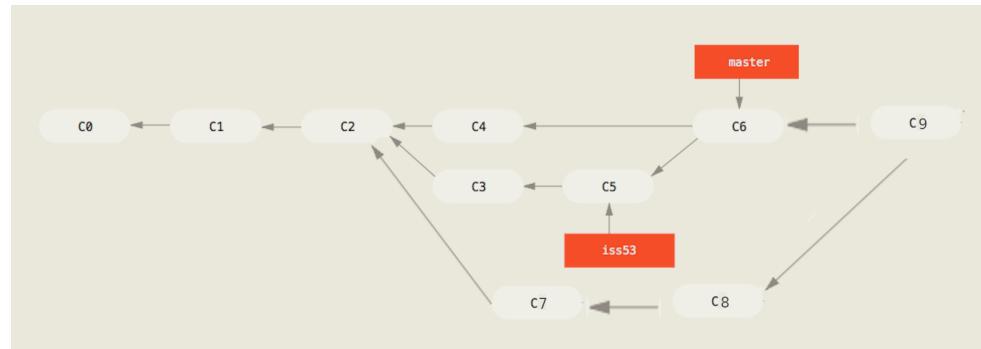
- Remember how we do merges of divergent branches

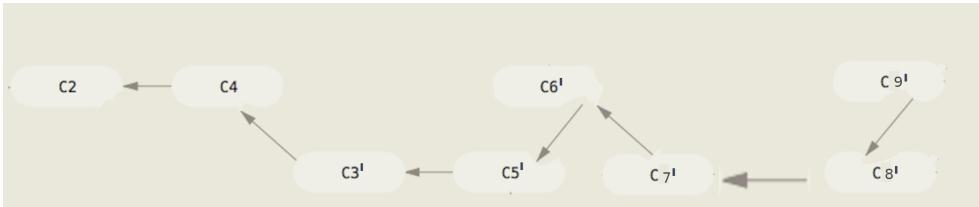


- Regular merge
 - this could be two developers, each working in a branch



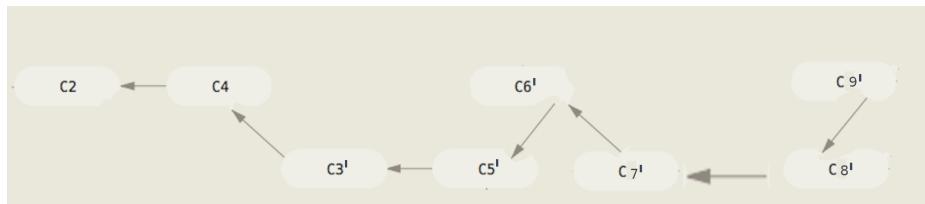
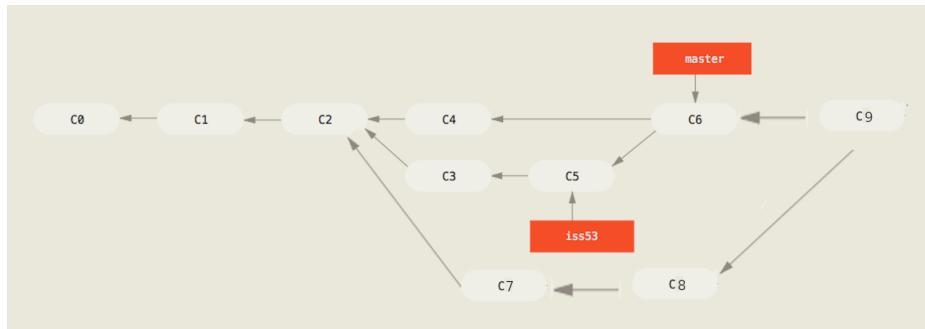
- Regular merge (2)
 - ... things start to get messy if several branches/developers want to merge



- Rebasing
 - Replay the commits in this branch, but starting from some other branch
- 

Merge vs. rebase

Let's just see them side by side, to understand the differences



Rebasing to keep a clean history (as we do in PORTA)

- We can have as many branches as we want
- Before merging to *development* branch we make sure that:
 - the topic branch could *fast-forward* merge to *development* by rebasing the topic branch on *development*
- We merge the topic branch to *development*, with the *--no-ff* option, to keep a clean history and leave clear indication of when we solved a particular issue.

[demo-1-11.cast](https://asciinema.org/a/Au3nBUONIIgXDrFvdPHK2IZeo?autoplay=1&cols=180&rows=40) (<https://asciinema.org/a/Au3nBUONIIgXDrFvdPHK2IZeo?autoplay=1&cols=180&rows=40>)

Rebasing to keep clean commits

- While rebasing you can decide that some commits can be *squashed*, "skipped", put in different order, etc. If you do this only in local branches, it is very useful, for example, to combine half-baked commits into a coherent one.
- But you need to have one rule in mind: **never rebase public branches** (if other people fetched your history and you rebase it, you are rewriting it, which will cause a lot of trouble for them)
- More info about rebasing: <https://git-scm.com/book/en/v2/Git-Branching-Rebasing> (<https://git-scm.com/book/en/v2/Git-Branching-Rebasing>)

8. Other useful Git configuration/commands/tools

- `.gitignore` (patterns of files that Git will simply ignore)
- `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)
- `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).