Version Control systems and Git

A brief introduction

Enrico Bassetti

Apr 19, 2018

Sapienza - University of Rome

Roadmap

1. Version control
2. Version Control systems
3. Git
4. Git: workstation setup
5. Git: daily workflow

Version control

Keeping track of versions

Either alone or in a team, one big issue of software development is *keeping track of changes* on the source code.

Why?

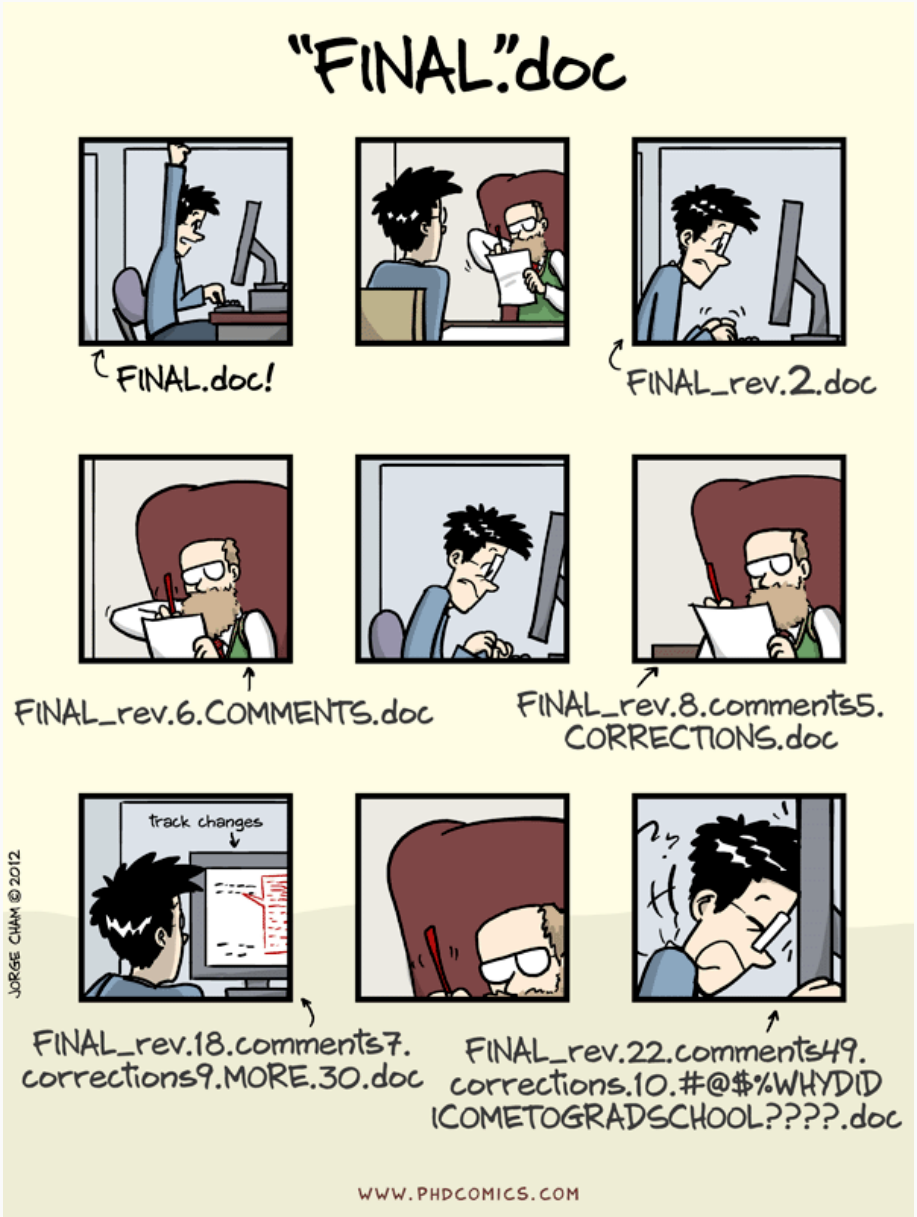
- Complex projects/documents, built up over time
- Multiple collaborators
- Multiple (parallel) versions (eg. testing a new algorithm while the main *branch* of development is focused on other things)
- Reproducibility (eg. bug hunting, etc.)

Keeping track of versions: what we need

In order to do so, we want:

- **Consistent versions**
- **Point-in-time marking** (aka tagging a released version)
- Multiple developers works on the **same** code-base
- Simple way to **merge** different *branches* of development

How do we track the code? Manual copies!



How do we track the code? Manual copies!

NO!

Why?

- Manual = fallible
- Labelling issue
- No metadata (dates, authors, etc.)
- No integrated/automated tool for storing/merging/managing copies

How do we track the code? Dropbox/Google Drive/OneDrive!

What about... **Dropbox/Google Drive/OneDrive**??

How do we track the code? Dropbox/Google Drive/OneDrive!

NO!

Why?

- No concept of "consistent checkpoint/commit"
- Collaboration is broken (unless you're working on very simple projects)
- No *branching/merging* capabilities
- Few metadata
- Too generic

How do we track the code? Version Control Systems!

- Consistent checkpoints, with **atomic operations**
- Written for text files (eg. source code), not for generic files
- Collaboration is well supported, eg. with file locking
- *Branching/merging* capabilities
- Metadata, a lot of

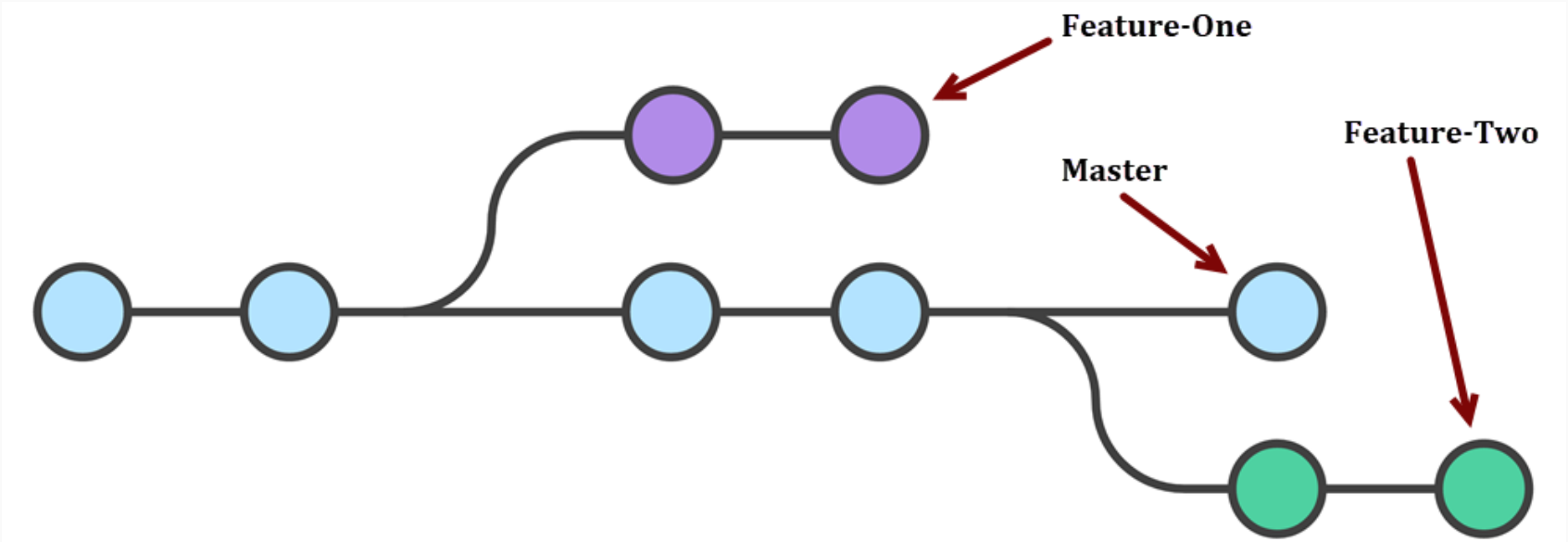
Examples of Version Control Systems: *Git*, *Subversion*, *Mercurial*.

Version Control systems

Version Control: glossary

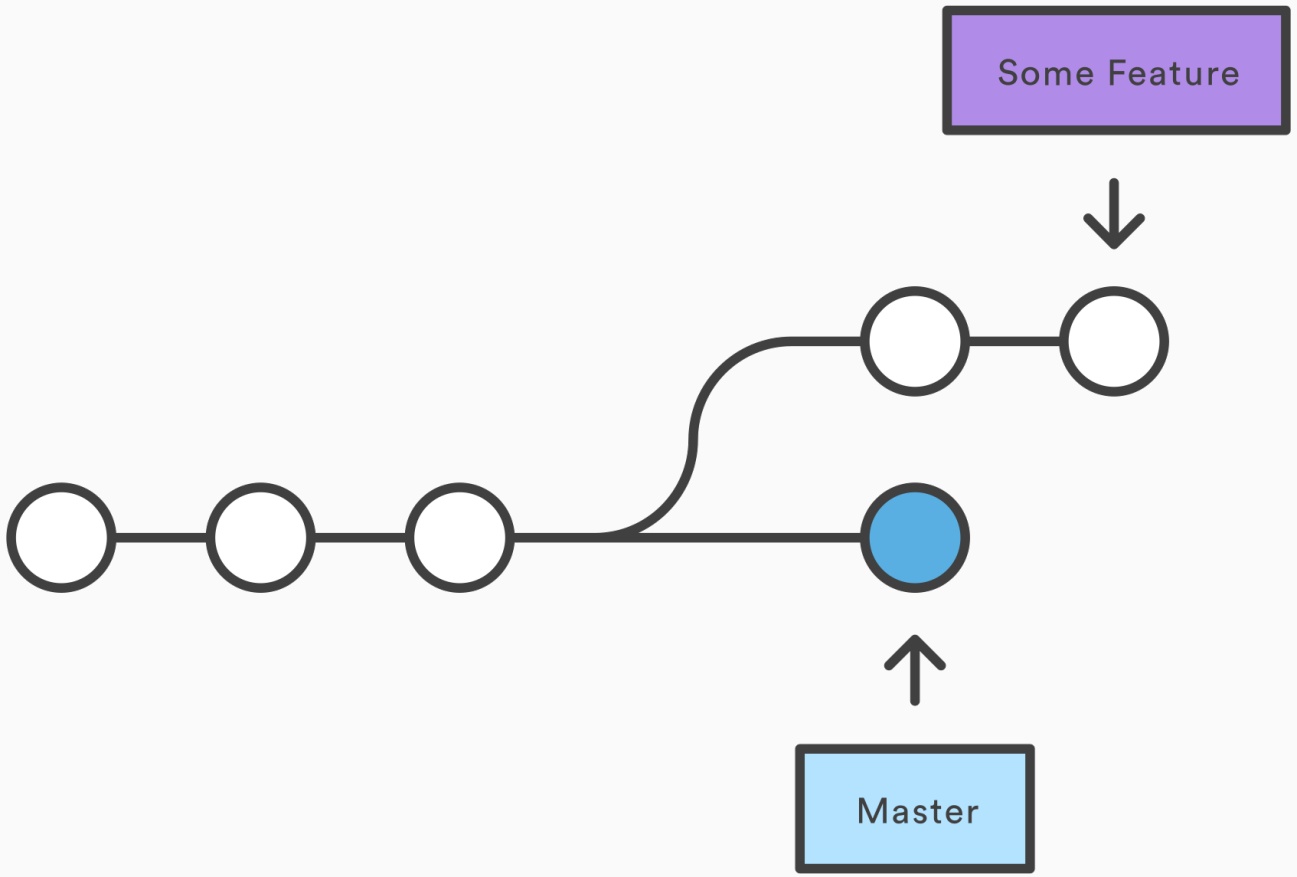
- **Commit**: a "snapshot" of the repository in a specific moment in time
- **Branch**: a parallel development
- **Merge**: the action of *fusing* two branches in one
- **Tag**: a custom *label* identifying a commit
- **Repository**: an ordered set of commits, branches and tags
- **Fork**: A copy of the entire repository
- **Pull request**: A request to merge code from a fork back to the parent repository

Version control history

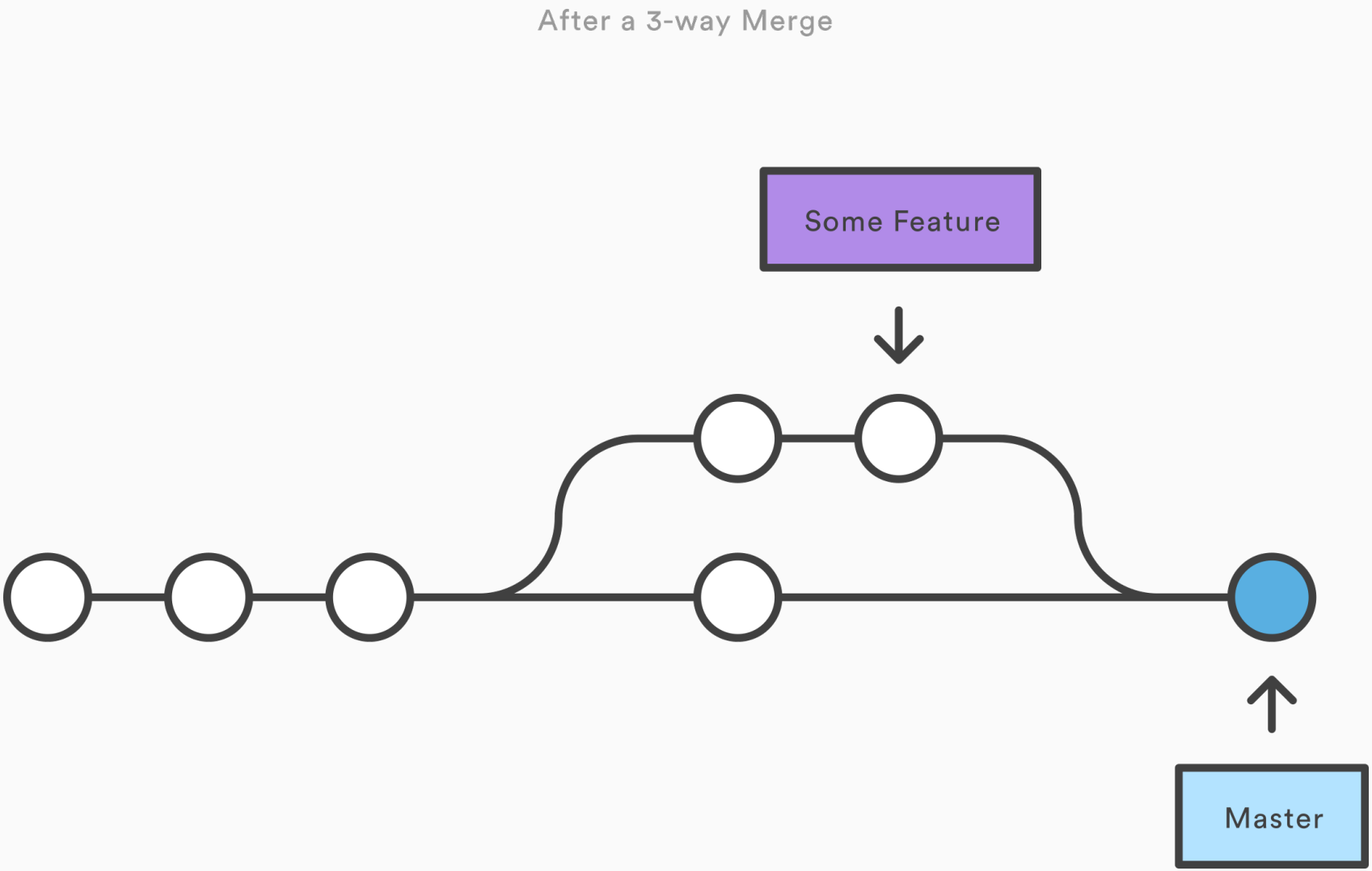


Version Control: merge - classic

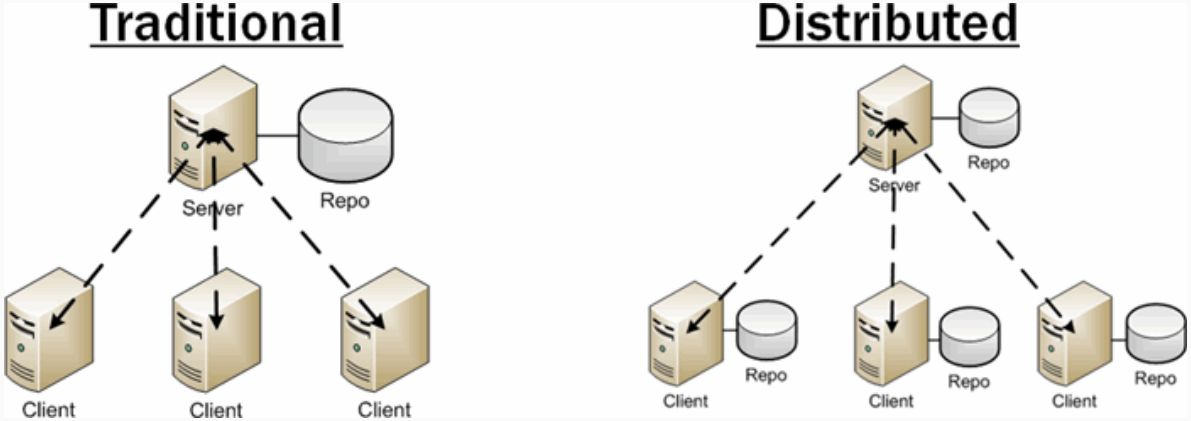
Before Merging



Version Control: merge - classic



Traditional versus Distributed Version Control



Traditional (centralized)
Everything lives in one place: the server. Easy to manage permissions, but it's not very flexible: single point of failure, clients needs to be constantly connected to the server, etc.

Distributed
Each client has an entire copy of the repository. No single point of failure, no need of persistent links.

Version control: hosting services

- Enables multiple git instances synchronization
- Provide ticketing systems (bugtracking/issue reporting system), project management, wiki for documentations
- Manage forks and pull requests
- ACL and fine granted access

Version control: well-known hosting services

- Most used public/private repository hosting platforms
- Many open source software are hosted in these services
- Mainly zero-cost for Open Source / Free Software, paid plans for private repositories
- Wiki, issues, pull requests
- Integrated Continuous integration and Continuous delivery

Git

Git introduction

Git (2005, Linus Torvalds) is a very powerful VC system.

In this presentation we'll skip description and internal working model, and we'll focus only on usage.

Initialize a git repository

If we don't have a repository, we need to create it:

```
git init
```

- Git will create a new (local) repository
- A special directory, named `.git`, is created on the root of the project
- No file will be added to the repository

Clone an existing repository

If a repository exists on a remote endpoint, we can clone it:

`git clone` *<url> [<localdir>]*

Adding a file into a repository

`git add` <*file*>

Adding a file into a repository means that Git will track the changes on file contents, attributes and other metadatas.

You need to add a file into the repository both the first time (to inform Git that you want to track that file) and before every commit where you want to "save" the file change (to inform Git that you want to store the change in this commit).

Create a commit in Git history

```
git commit
```

Note: if you don't specify the commit message with `-m` parameter, Git will open the default editor to write one.

Common operations

Create a new branch

`git branch <branchname>`

Create a new branch and switch to it

`git checkout -b <branchname>`

Merge the branch *bob* into current branch

`git merge bob`

Send commits to (default) remote endpoint

`git push`

Retrieve commits from (default) remote endpoint

`git pull`

Git: workstation setup

Before we begin

On a new computer/user profile, we need to setup our environment:

```
$ git config --global user.name "John Doe"
$ git config --global user.email john@doe.com
```

Start to develop on an existing Git repository

If you(/your team) have an existing project on Git, you can start by *cloning* the repository locally

```
$ git clone https://githosting/gitrepo gitrepodir
$ cd gitrepodir/
```

Import an existing project into a repository

If you have an existing project outside Git, you can import it with:

```
$ cd existingproject/
$ git init
$ git add .
$ git commit
$ git remote add origin https://githosting/gitrepo
$ git push -u origin master
```

Git: daily workflow

Pulling remote changes

At the beginning of the day, or when someone push changes to server, you need to update your local copy:

```
$ cd existingproject/
$ git pull
```

Creating branch

When needed, you can create a new branch, starting from the current commit:

```
$ cd existingproject/
$ git branch new-feature-1
$ git checkout new-feature-1
$
```

Merging branch

Let's suppose that we're on `master` branch and we want to merge `new-feature-1`

```
$ cd existingproject/
$ git merge new-feature-1
$ git push
$
```

Commit and push after work

At the end of the day, or when needed, we can create a commit by issuing:

```
$ cd existingproject/
$ git add modifiedfile1 addedfile2 removedfile3
$ git commit
$ git push
```

Useful readings

- https://www.atlassian.com/git/tutorials
- https://git-scm.com/book/en/v2
- http://thepilcrow.net/explaining-basic-concepts-git-and-github/

Questions?