Platforms for Networked Communities

Introduction to Git

Boni García

http://bonigarcia.github.io/
boni.garcia@uc3m.es

Telematic Engineering Department
School of Engineering

2020/2021

uc3m | Universidad Carlos III de Madrid

Table of contents

Table of contents

1. Introduction - What is Git?

- According to Wikipedia:

  > “ ***Git*** *is a* ***distributed version-control system*** *for tracking changes in* ***source code*** *during* ***software development*** ”

- Git was created by Linus Torvalds in 2005 for development of the Linux kernel
- Git is Free and Open Source Software (FOSS), licensed under the terms of the GNU General Public License version 2 (GPLv2)

https://git.kernel.org/pub/scm/git/git.git/

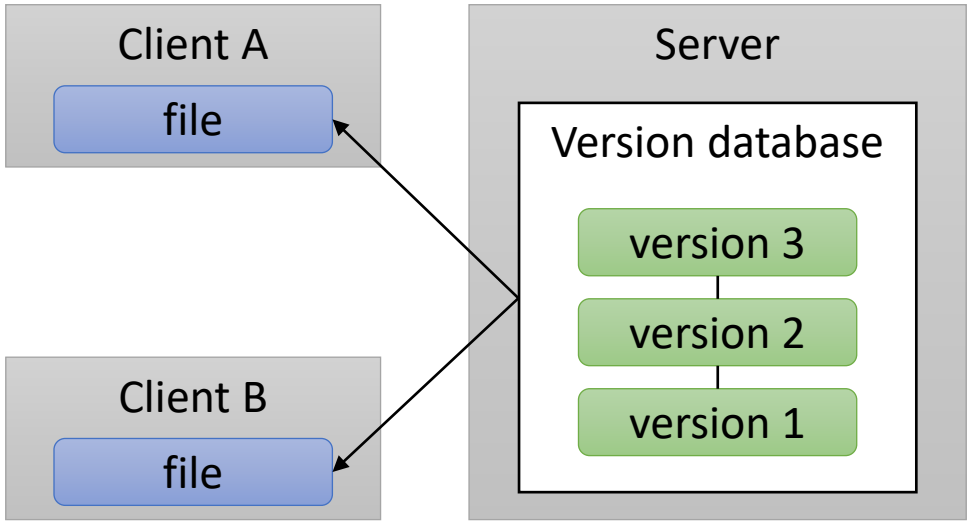
1. Introduction - Version control systems

- **Version control** is the name given to the process of **management of changes** to a collection of information (e.g. source code, documents, websites, etc.). It allows:
  - Keep track who/when made each change (sometimes referred as patch)
  - Combine changes made by different people
  - Revert contents to a previous state or version
- A **Version Control System** (VCS) is a **tool** for managing the version control process.
  - A VCS provides a timestamped and annotated history of changes to the project, which allows monitoring the changes and facilitate collaboration
  - Examples of VCS: Git, CVS, Subversion (SVN), Mercurial

1. Introduction - Version control systems

- The are two main types of VCSs:

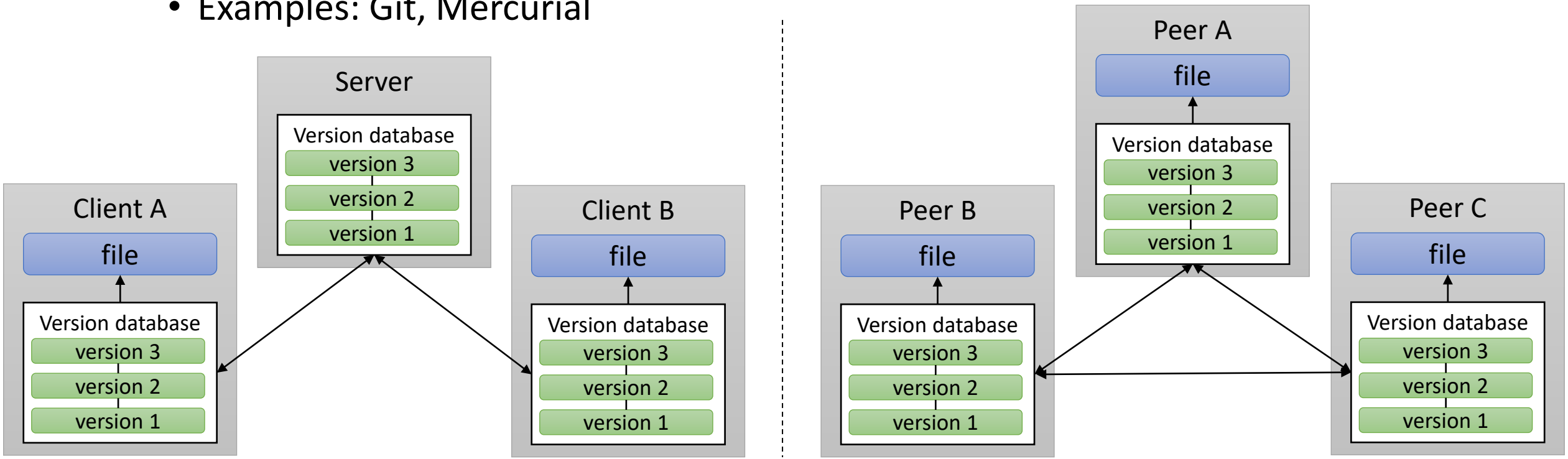
1. **Centralized**
   - Client-server architecture
   - Clients check out the latest snapshot of the resources from a central server
   - Examples: CVS, SVN



1. Introduction - Version control systems

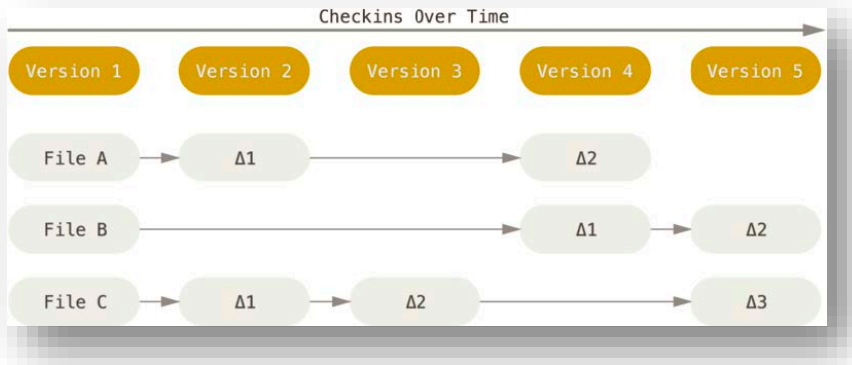
2. Distributed

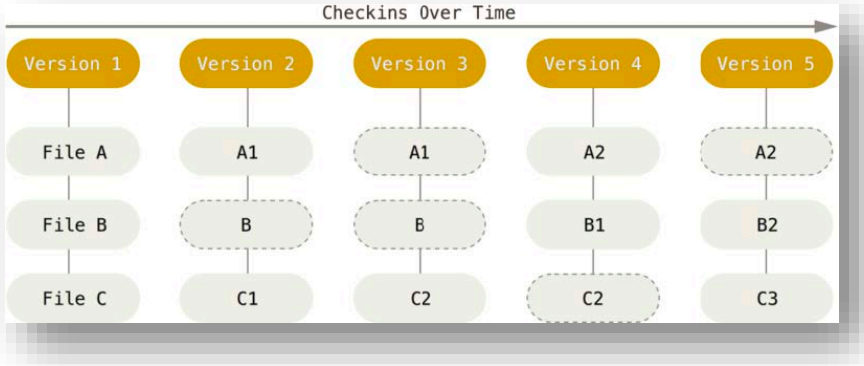
- Users mirror the repository, including its **full history**
- Flexible peer-to-peer architecture (each host can contribute to other repositories and maintain public repositories in which other can contribute)
- Examples: Git, Mercurial



1. Introduction - Version control systems

- Centralized VCS are **delta-based**
  - Data is stored as changes to a base version of each file
- Git stores data as data a **stream of snapshots**
  - Git stores the state of each file with each **commit** (if a files is not changed, Git just a link to the previous file already stored)
  - Each snapshot is stored internally in Git using a **key-value map** in which the value is the state of each file in a commit and the key is a **SHA-1** hash value (40-character string composed of hexadecimal characters)
  - Advantage: fast (network only required for specific commands)





1. Introduction - Development platforms

- **Development platforms** are collaborative cloud (web-based) repository hosting services for creating and sharing software
  - These platforms are sometimes referred as **code hosting platforms** or **forges**
  - These platforms typically provide a **social environment** for developers
- Some of the most relevant platforms using **Git** as VCS are:

https://github.com/

GitLab

https://about.gitlab.com/

Bitbucket

https://bitbucket.org/

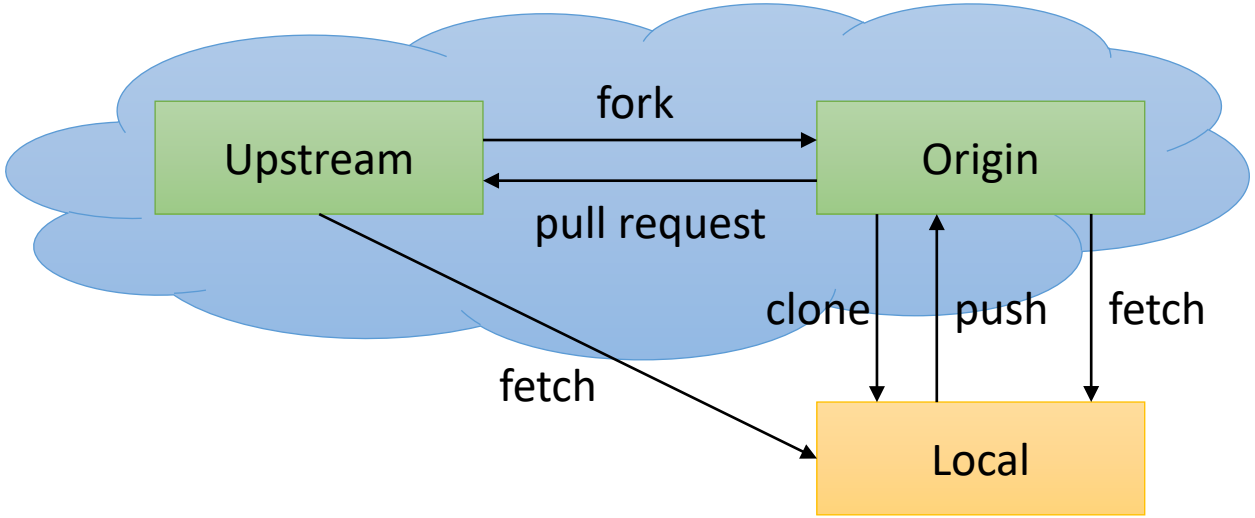
1. Introduction - References

- Official Git documentation
  - https://git-scm.com/doc
  - https://git.wiki.kernel.org/index.php/GitDocumentation
- Pro Git (2nd edition), Scott Chacon and Bend Straub, Apress, 2014
  - https://www.git-scm.com/book/en/v2
- GitHub guides
  - https://guides.github.com/
- Atlassian Git tutorials
  - https://www.atlassian.com/git/tutorials
- Git man pages
  - http://man7.org/linux/man-pages/man1/git.1.html

Table of contents

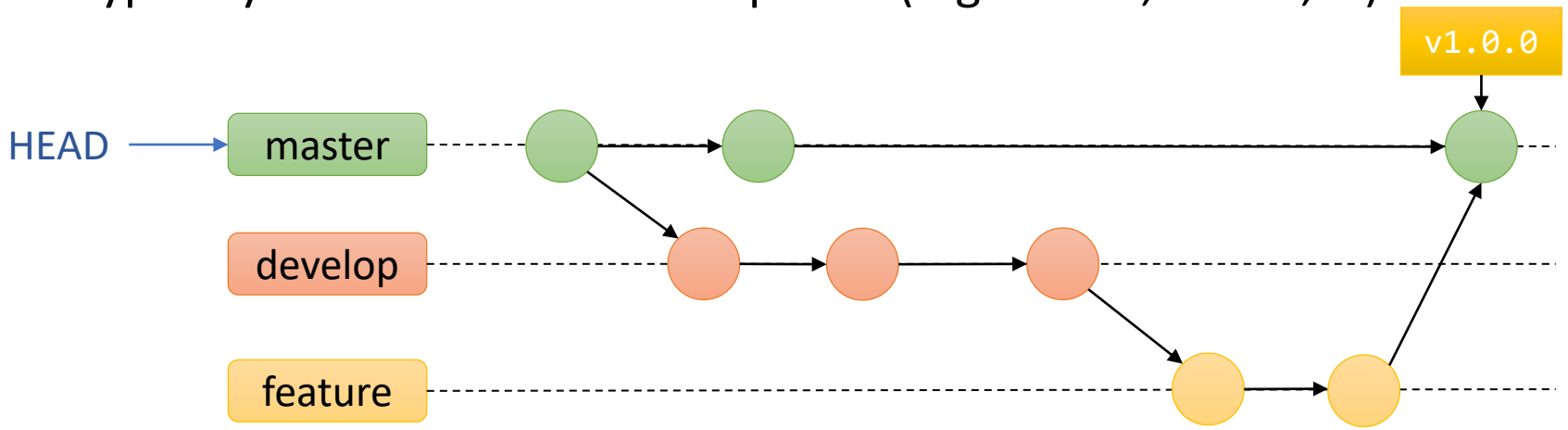
2. Git - Terminology

- **Repository** (or *repo*) is a collection of files tracked together by Git
  - A repo is **remote** when is hosted in a server (e.g. GitHub), and it is **local** when is stored directly in a user machine
- **Clone** is to download all files from a remote repository to a local machine
- **Fork** is a direct copy of a remote repository from a different owner
- **Origin** is the standard naming convention for a remote repository
- **Upstream** is the standard naming convention for the original remote repository of a fork



2. Git - Terminology

- A **commit** is an individual change to a file(s) in the repository. It is identified with a unique identifier (a hash code generated with SHA-1)
- A **branch** is a movable pointer to one of these commits
  - The **master** branch is the default branch when creating a Git repository
  - Different branches can appoint to divergent path from the main development line (master) an evolve in parallel
  - Git maintains an pointer called **HEAD** which points to a given branch
- A **tag** is a label which points to an specific commit
  - Typically used to mark release points (e.g. v1.0.0, v1.0.1, …)



2. Git - Install and initial setup

- First, we need to **install** Git in our machine:

```
sudo apt-get install git-all
```

```
sudo dfn install git-all
```

https://git-scm.com/download/win

http://git-scm.com/download/mac

- Then, at least we need to configure our **user name** and **email**:

```
git config --global user.name "My Name"
git config --global user.email myemail@email.com
```

We can use the command line to run Git

2. Git - Create local repository

- We can create a new repository using the command **`git init`**
- This command creates a hidden folder (`.git`) containing all the internal data of the repository

```
boni@ubuntu:~/dev$ mkdir hello-world-git
boni@ubuntu:~/dev$ cd hello-world-git
boni@ubuntu:~/dev/hello-world-git$ git init
Initialized empty Git repository in /home/boni/dev/hello-world-git/.git/
```

- It is very common to use **remote repositories** hosted in development platforms (e.g. GitHub, GitLab)
  - In the next examples, we will use **GitHub**

2. Git - Cloning a remote repository

- We can clone a remote GitHub repository using HTTPS or SSH
  - With HTTPS, our credentials (or a token) are required to make commits
  - The use of credentials is going to be deprecated
  - The use of **SSH keys** is recommended

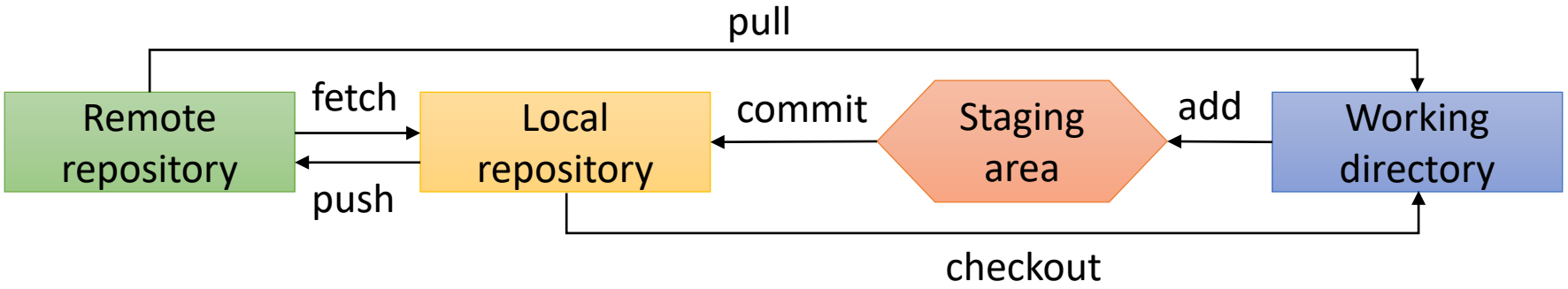
We use the command **`git clone`** to get a complete copy of the remote repository

```
boni@ubuntu:~/dev$ git clone git@github.com:bonigarcia/git-intro.git
Cloning into 'git-intro'...
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (5/5), 5.32 KiB | 5.32 MiB/s, done.
boni@ubuntu:~/dev$ cd git-intro
boni@ubuntu:~/dev/git-intro$ git remote -v
origin git@github.com:bonigarcia/git-intro.git (fetch)
origin git@github.com:bonigarcia/git-intro.git (push)
```

We use the command **`git remote`** to see the information about the remote repository

2. Git - Track changes

- Once we have cloned a repository, we **checkout** a given commit, updating the state of the repository in the **working directory**
- Git has three states that our files can reside in:
  - Committed: data is safely stored in the local repository
  - Modified: some file has changed but still is not in the local repository
  - Staged: modified file is marked to go into the next commit. Files in this state are placed in **staging area** (also known as **index**)



2. Git - Track changes

- The commands we typically use to track changes are:
  - **`git status`** : Show the state (staged, modified, untracked) of the changes
  - **`git add`** : mark changes to be committed (staging area)
  - **`git commit -m "message"`** : confirms changes and stores them in the local repository
  - **`git log`** : Show the info of the HEAD commit
    - **`git log --oneline --decorate --graph --all --color`** : Show tree of commits (this can be added as alias using the command: `git config --global alias.tree "log --oneline --decorate --graph --all --color"`)
  - **`git checkout <branch>`** : Move the position of the HEAD (to the top of a branch or a given commit, using its hash code)
  - **`git push`** : Update changes from the local to the remote repository
  - **`git fetch`** : Update changes from the remote to the local repository
  - **`git pull`** : Make a fetch from the remote repository and checkout to HEAD

2. Git - Track changes

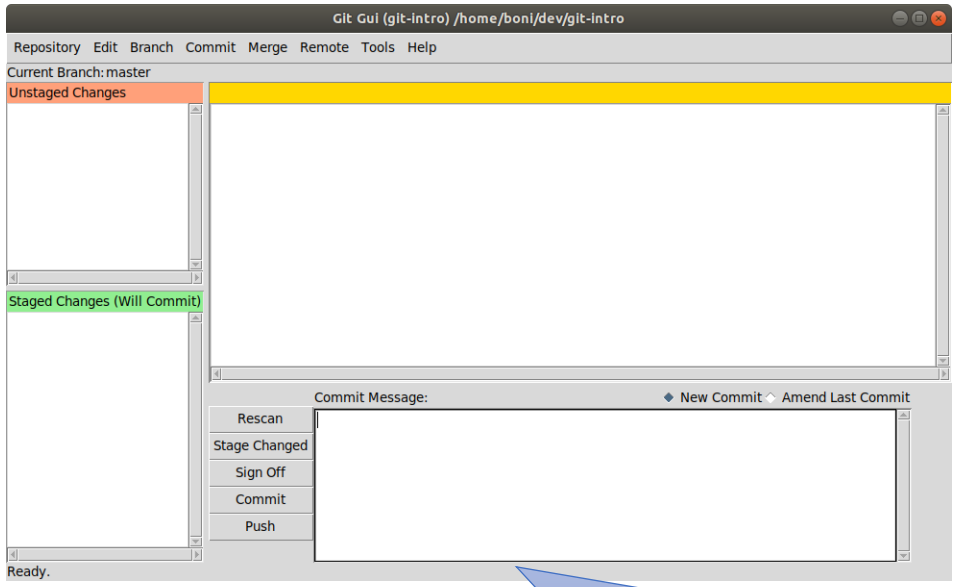
- Complete example using the command line:

```
boni@ubuntu:~/dev/git-intro$ git status
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean
boni@ubuntu:~/dev/git-intro$ nano README.md
boni@ubuntu:~/dev/git-intro$ git add README.md
boni@ubuntu:~/dev/git-intro$ git commit -m "Update README"
[master 8ffd759] Update README
 1 file changed, 1 insertion(+)
boni@ubuntu:~/dev/git-intro$ git log
commit 8ffd759d05aa857e58dc98cceaa254b45e1cc017 (HEAD ->
master)
Author: Boni García <boni.garcia@uc3m.es>
Date:   Wed Feb 19 17:29:27 2020 +0100

    Update README

commit ea24af81f4d98782181e3f84ee26556c534e0e0c (origin/master,
origin/HEAD)
Author: Boni García <bgarcia@gsyc.es>
Date:   Wed Feb 19 13:58:36 2020 +0100

    Initial commit
```

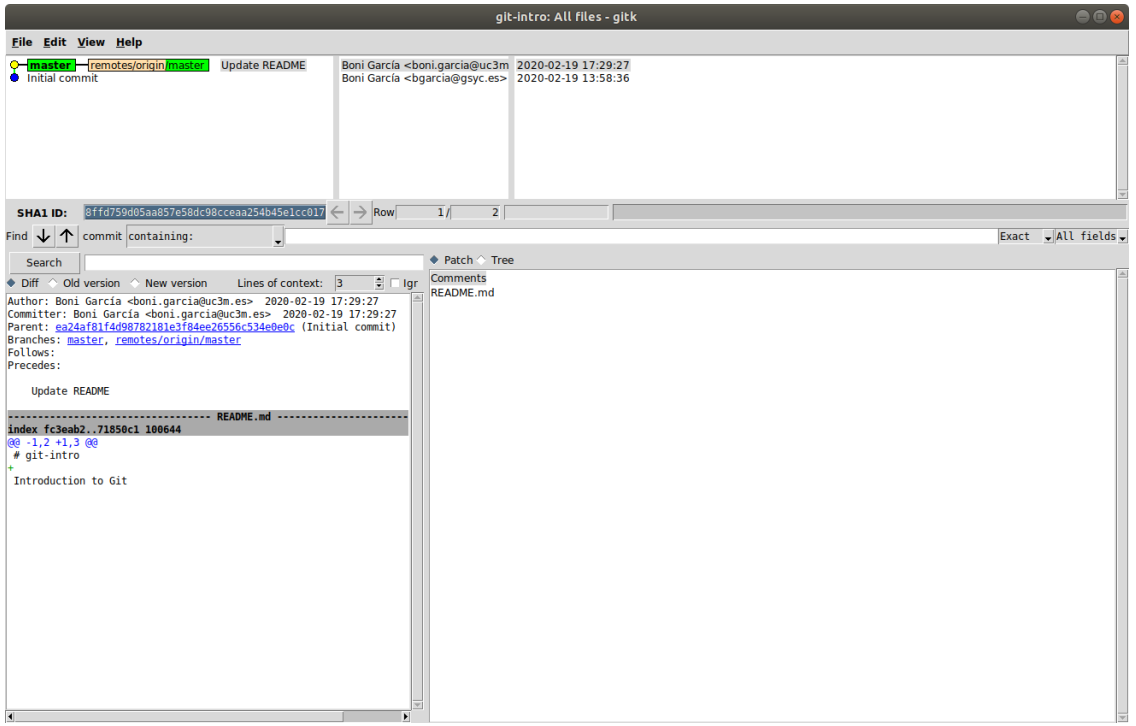
```
boni@ubuntu:~/dev/git-intro$ git push
Counting objects: 3, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 293 bytes | 293.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local
object.
To github.com:bonigarcia/git-intro.git
   ea24af8..8ffd759  master -> master
boni@ubuntu:~/dev/git-intro$ git log
commit 8ffd759d05aa857e58dc98cceaa254b45e1cc017 (HEAD ->
master, origin/master, origin/HEAD)
Author: Boni García <boni.garcia@uc3m.es>
Date:   Wed Feb 19 17:29:27 2020 +0100

    Update README

commit ea24af81f4d98782181e3f84ee26556c534e0e0c
Author: Boni García <bgarcia@gsyc.es>
Date:   Wed Feb 19 13:58:36 2020 +0100

    Initial commit
boni@ubuntu:~/dev/git-intro$
```

2. Git - Track changes

- The same actions can be done using GUI tools:
  - **`git gui`** : Graphical user interface to trace changes with Git
  - **`gitk`** : Graphical commit viewer for Git



We can define the spell checking dictionary using the command: `git config --global gui.spellingdictionary "en"`



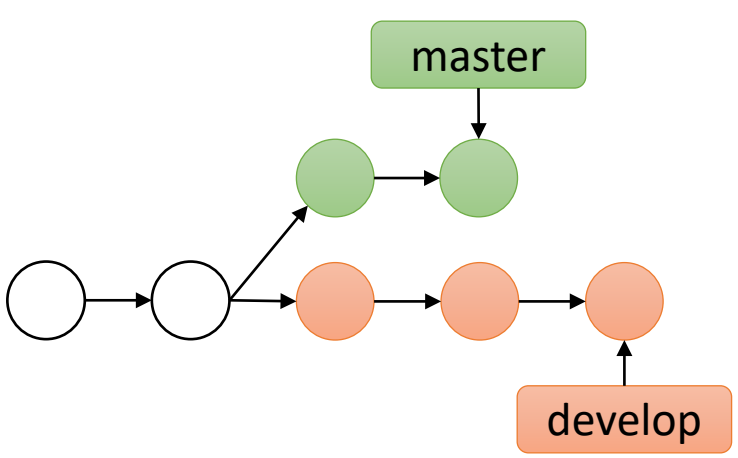
2. Git - Other commands

- Other typical Git commands are:
  - **`git rm`** : Removes file from the tracked files and working directory
  - **`git remote add <name> <url>`** : Adds a remote named *<name>* for the repository at *<url>*
  - **`git branch <new_branch>`** : Creates a new local branch
  - **`git checkout -b <new_branch>`** : Creates and checkouts new branch
  - **`git reset`** : Restore commit and staged files
  - **`git reset --hard`** : Restore everything (commit, stage, and working tree)
  - **`git tag <tagname>`** : Create a tag in the current commit
  - **`git blame <file>`** : Show revision/author last modified each line of a file
  - **`git diff`** : Shows the differences between the working version of files and the version of these files in a particular commit

https://git-scm.com/docs

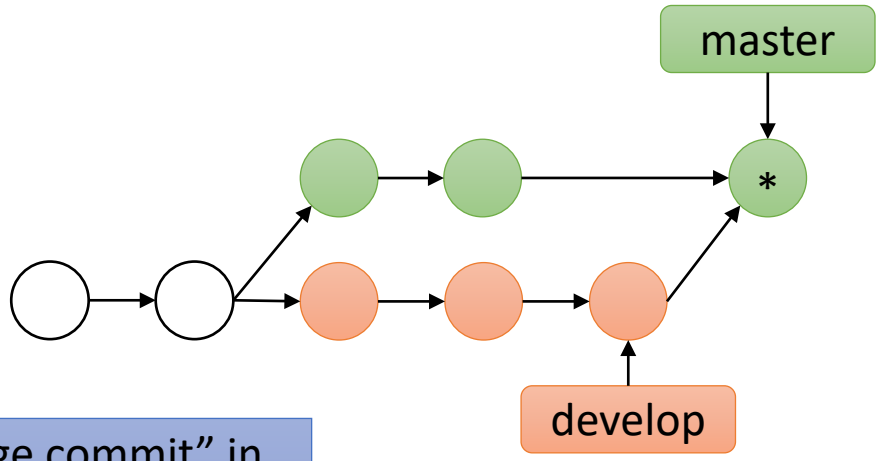
2. Git - Merging and rebasing

- Git use **merging** and **rebasing** to integrate changes from different branches
- The command **`git merge`** allows to incorporate commits from one to another branch



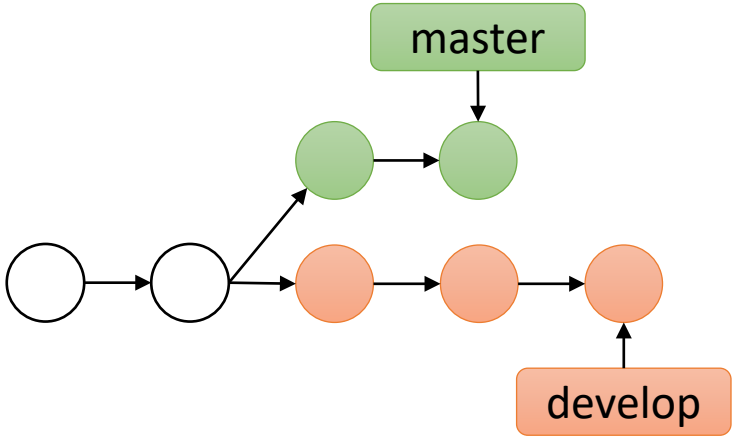

```
git checkout master
git merge develop
```

This creates a new “merge commit” in the master branch that ties together the histories of both branches



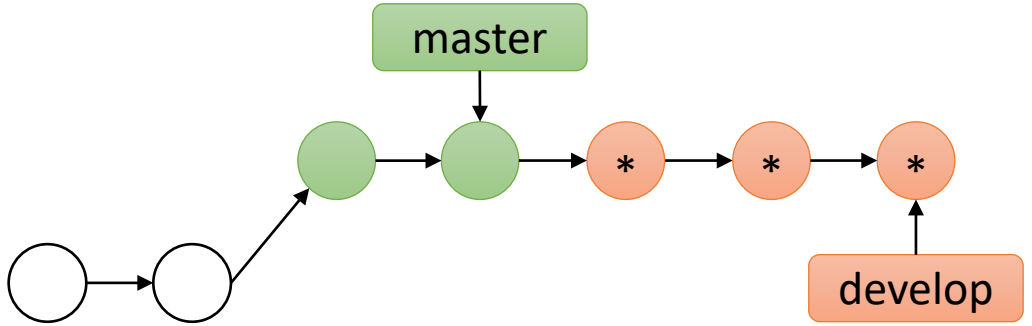
2. Git - Merging and rebasing

- The command **`git rebase`** moves an entire branch to begin on the end of other branch



This re-writes the project history by creating new commits for each commit in the original branch

```
git checkout develop
git rebase master
```



2. Git - Resolve conflicts

- Centralized VCS (such as CVS or SVN) use **locking** to avoid conflicts
- Git is more flexible: does not lock files (users can modify in parallel) and conflicts can happen
- Merge operations (pulling and rebasing) can provoke conflicts
- Conflicts are solved manually by adding the right part of the conflicting files

```
<<<<<<< HEAD
this is some content to mess with
content to append
=======
totally different content to merge later
>>>>>>> branch_to_merge
```

Table of contents

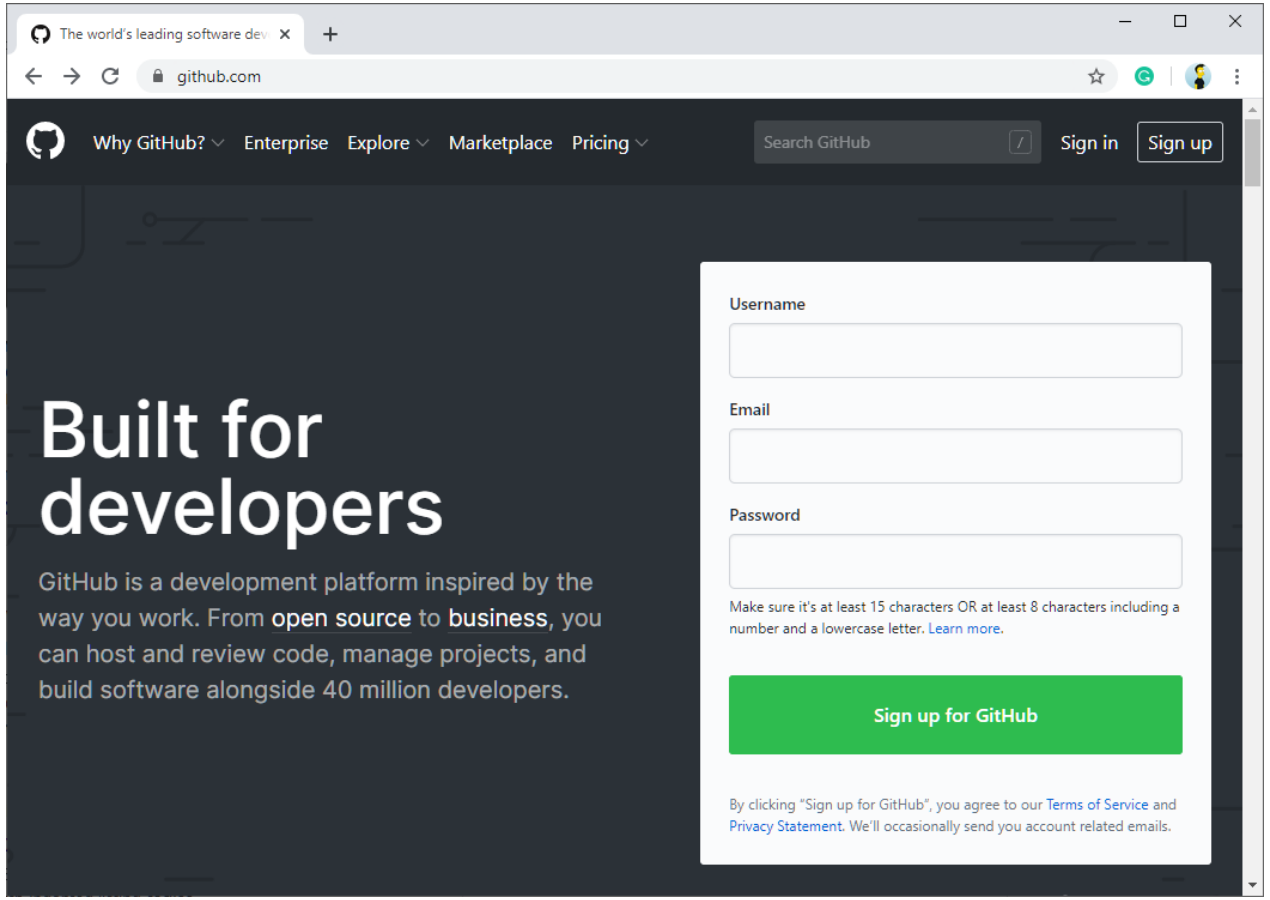
3. GitHub

- GitHub is the largest collaborative development platforms nowadays
  - As of January 2020, GitHub reports having over 40 million users and more than 100 million repositories (including 28 million public repositories)
- In addition to code hosting, GitHub supports other features:
  - Documentation, including wikis and README using Markdown formats
  - Issue tracking with labels, milestones, assignees and a search engine
  - Pull requests with code review and comments
  - Email notifications (e.g. notifications by @ mentioning them)
  - GitHub Pages: small websites made from public repositories
  - GitHub Actions: CI/CD (continuous integration and deployment)
  - …

https://octodex.github.com/

3. GitHub - First steps

- First, we need to create a GitHub account:



https://github.com/

3. GitHub - First steps

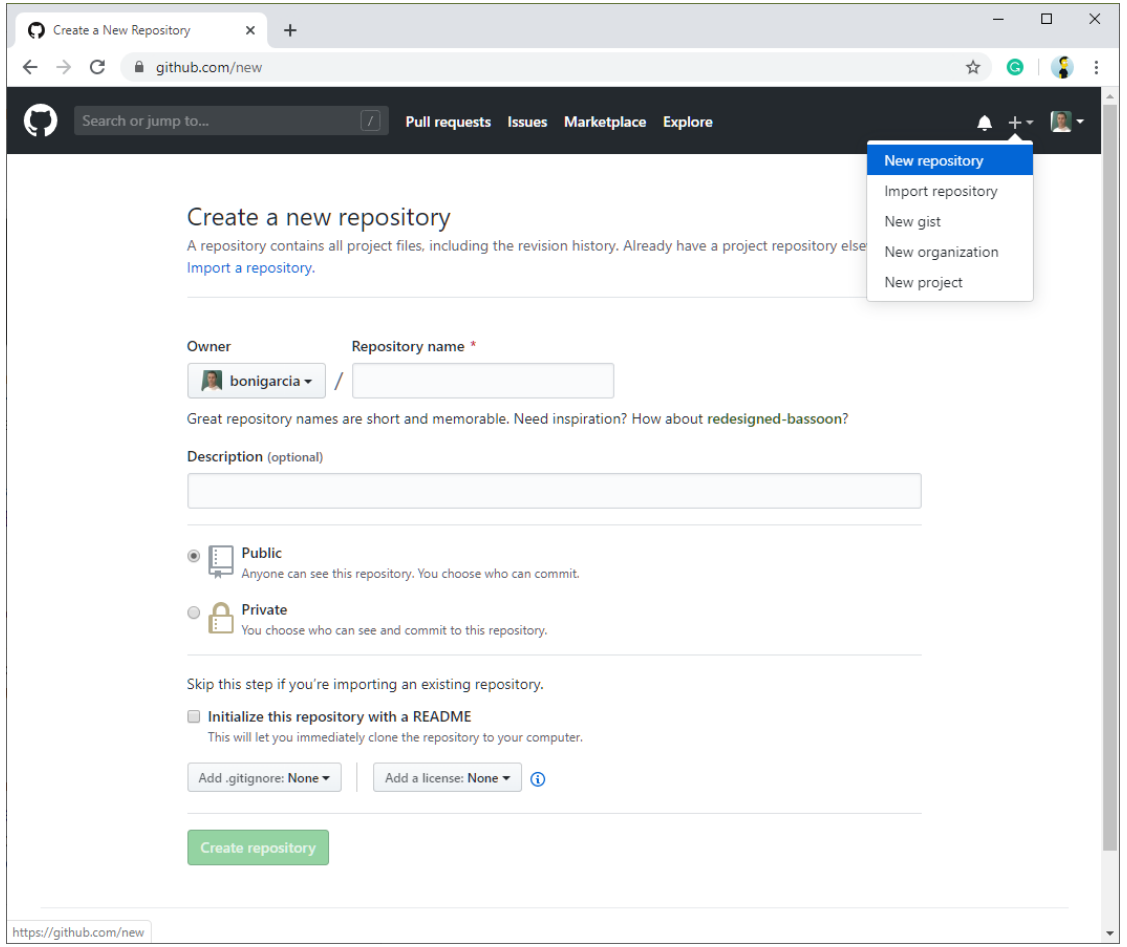
- Although not mandatory, it is recommended to include a **SSH key** in the GitHub account setup
  - This way, the process of commit new changes to the repository will be easier
  - If we don't have a pair of private-public keys, it can generated as follows:

```
boni@ubuntu:~/dev$ ssh-keygen -C boni.garcia@uc3m.es
Generating public/private rsa key pair.
Enter file in which to save the key (/home/boni/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/boni/.ssh/id_rsa.
Your public key has been saved in /home/boni/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:e+nWMTreQ8yniIWHoXVncTCLd5/KwJnOIB1Fsxxpz4I boni.garcia@uc3m.es
```

  - Then, we need to copy the content of the public key (`~/.ssh/id_rsa.pub`) to https://github.com/settings/keys

3. GitHub - Create new repository

- We use the button **+** and the option "**New repository**" to create a new repository in our GitHub account
- This page allows to include several typical files in the new repository:
  - `README.md` (documentation in **markdown** format). More info on https://guides.github.com/features/mastering-markdown/
  - `LICENSE` (legal guidelines for the use and distribution of software)
  - `.gitignore` (files not tracked by Git)



3. GitHub - Create new organization

- GitHub **organizations** are shared accounts where different users can collaborate in different repositories
- Owners and administrators can manage member access to the organization's data and repositories
- Organizations are free for open-source projects

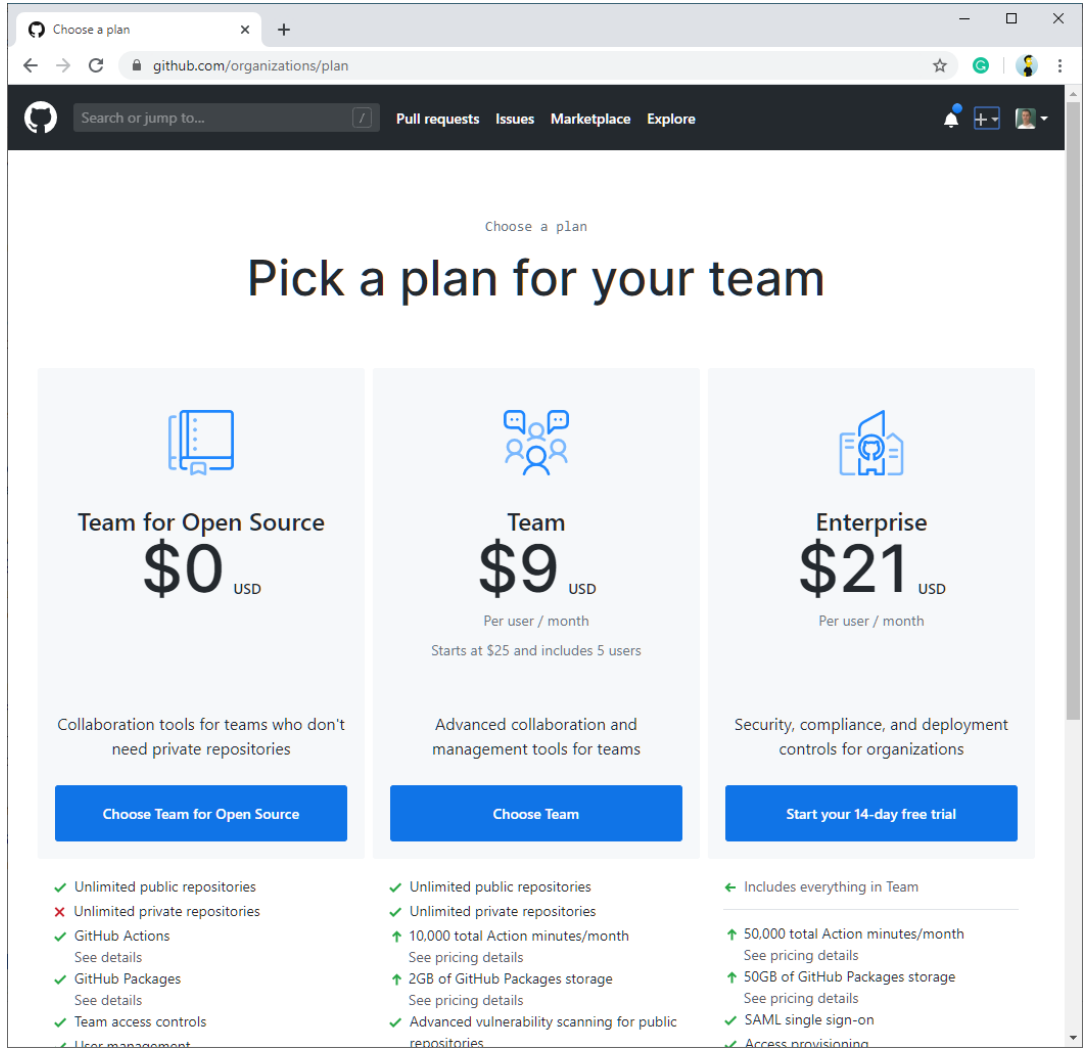


Table of contents

4. Takeaways

- Git is a popular version control system (i.e., a tool) for tracking changes in source code during the software development lifecycle
- Git repositories includes the full history (it is agile since network is only required for specific commands)
- Development platforms (e.g. GitHub) host remote repositories and provide extra features for collaborative software development
- The typical workflow of Git and GitHub is: create remote repository (origin) → clone repository → checkout branch (master) → commit changes to local repository → push to origin→ pull/fetch from origin
- Git can be used completely from the command line (alternatively there are GUI tools)