by Mari Chelo

Development environments
Block 1
Unit 7: Introduction to version control. Git tools

1.7.1. Introduction to version control

1.7.1.1. Definition

Version control systems (VCS) are tools that can register any change in any file or set of files over time, so that we can easily recover any older version. They can be used not only with source files, but also with any other file type.

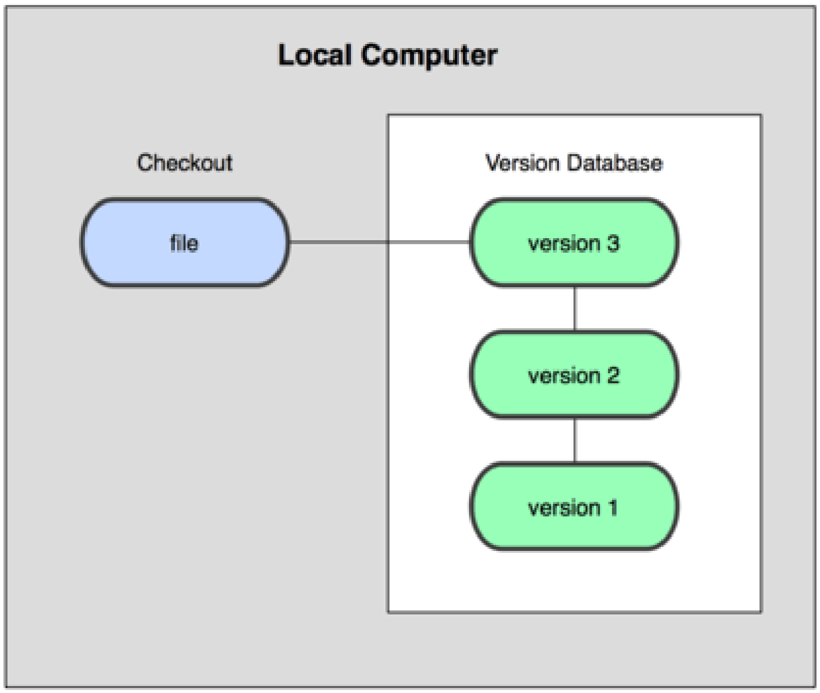
A VCS lets us revert the state of any file or even of a whole project, and compare files over time, determine who changed the file at a given timestamp and much more. Besides, if any file gets damaged or lost, we can just go back to a previous version in history and recover it again.

1.7.1.2. VCS types: Local VCS

VCS can be used either online or in local mode. This last mode is particularly useful because we can easily create a backup of a project and store it locally, so that we can restore it later if we need to (in case of an error, for instance) and go back to a stable version.

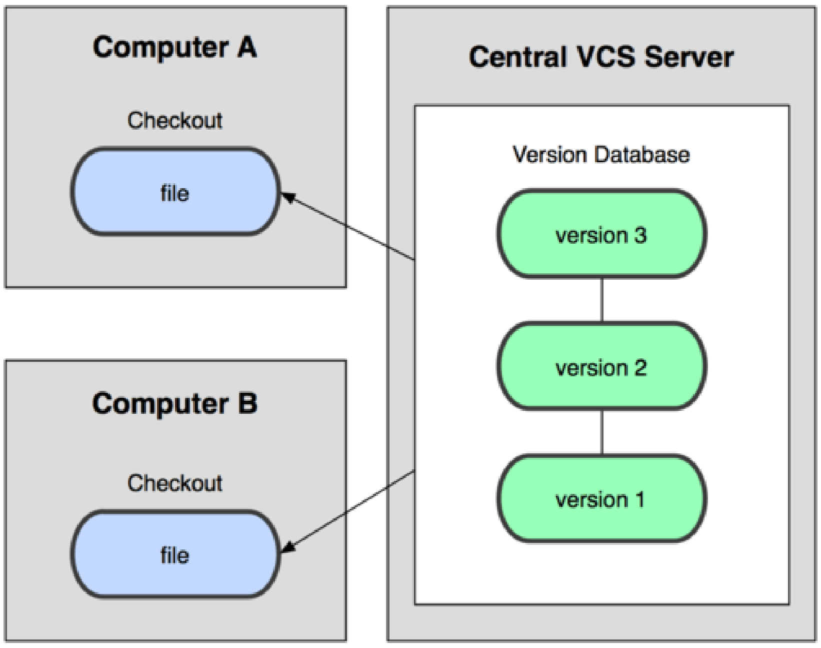
The main advantage of this is its simplicity, and the main drawback is that we must manage the version control manually, so we may make some mistakes in this process. For instance, we must forget that we are in the wrong folder, and then modify the backup file instead of the current one.

In order to face these problems, there are some interesting tools that help us manage the files and changes. One of the most popular ones is a system called *rcs*, which can still be found in many computers. This tool basically stores a set of patches or differences between files from one version to the next one. These changes are stored in a special file type, and then the system can recover any previous state of any file, by adding or substracting the corresponding patches.



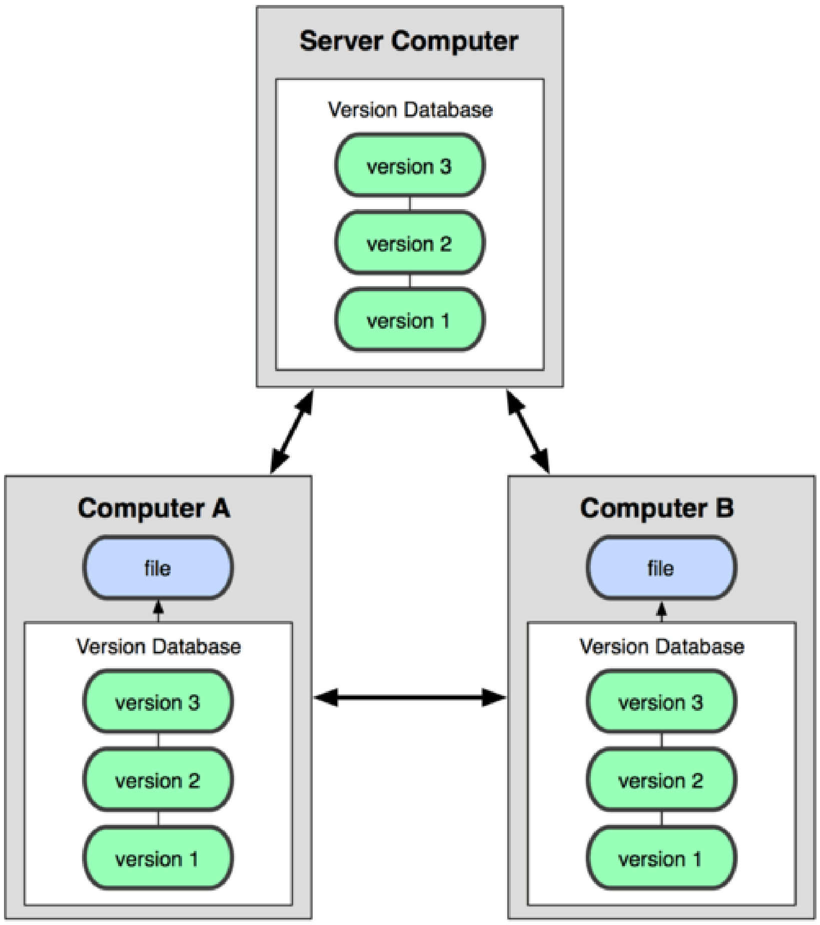
1.7.1.3. VCS types: Centralized VCS

Local VCS are not suitable when we need to collaborate with other team members. To solve this problem, there are also centralized VCS (CVCS). These systems are installed in a single server that contains all the files and their different versions. Then, many clients can connect to this server and download/upload changes to these files. This second way of controlling versions was a standard for many years, since it had a great advantages over the local CVS systems, but its main drawback is that, if server fails, we could lose the whole project.



1.7.1.4. VCS types: Distributed VCS

Distributed VCS (DVCS) emerged to solve the main drawback of CVCS. In a DVCS (such as Git, Mercurial, Bazaar or Darcs), clients not only connect to the server, but also download the whole repository. So, if a server fails, any of the local repositories of the clients can be copied to the server again, and the project can be restored. Everytime we download anything from the repository, we are in fact making a complete backup of the data.



1.7.2 Git

Git was developed by the Linux team once they broke up the relationship with *BitKeeper*, the tool that they used for version control before. From the lacks seen in this tool, they decided some of the main targets of the new system to be developed:

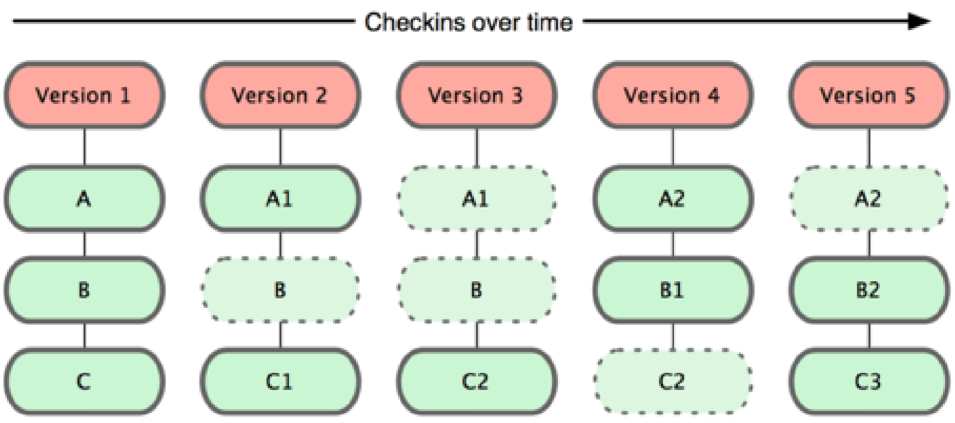
- Speed
- Easy design
- Strong support to non-lineal development (thousands of parallel branches)
- Completely distributed
- Suitable for big projects (such as Linux core)
- Efficiency (in terms of speed and data size)

From its birth in 2005, Git has evolved and it has become more and more easy to use. It is really fast and efficient with big projects, and has an outstanding branching system.

1.7.2.1 Git foundamentals

1.7.2.1.1 Data modeling

Git stores some kind of snapshot set of its file system, instead of storing a list of changes. Everytime we upload a new change, it basically takes a photo of every file at that moment, and stores a reference to this snapshot. If the file has not been modified, then Git does not save a copy of it, just a link to previous, identical version.



This is an important difference between Git and almost every other VCS, and it makes Git reconsider these aspects from previous generations of VCS. So it looks more like a small file system with some useful tools, rather than a VCS.

1.7.2.1.2 Local work

Most of Git functions just need local files and resources to work. As the project history is stored locally, many operations are immediate, and it lets us work in a project even if we are not connected to the Internet. Changes are stored locally and, as soon as we have a connection, the external repository can be updated.

1.7.2.1.3 Integrity

Git uses *hash* SHA-1 algorithm to store the information, so data is always verified and, in case it is changed, Git would notice.

1.7.2.1.4 It only adds information

Every Git operation consists in adding some information, so everything can be easily undone (information is not erased). After confirming a snapshot, information is stored safely.

1.7.2.1.5 Project states

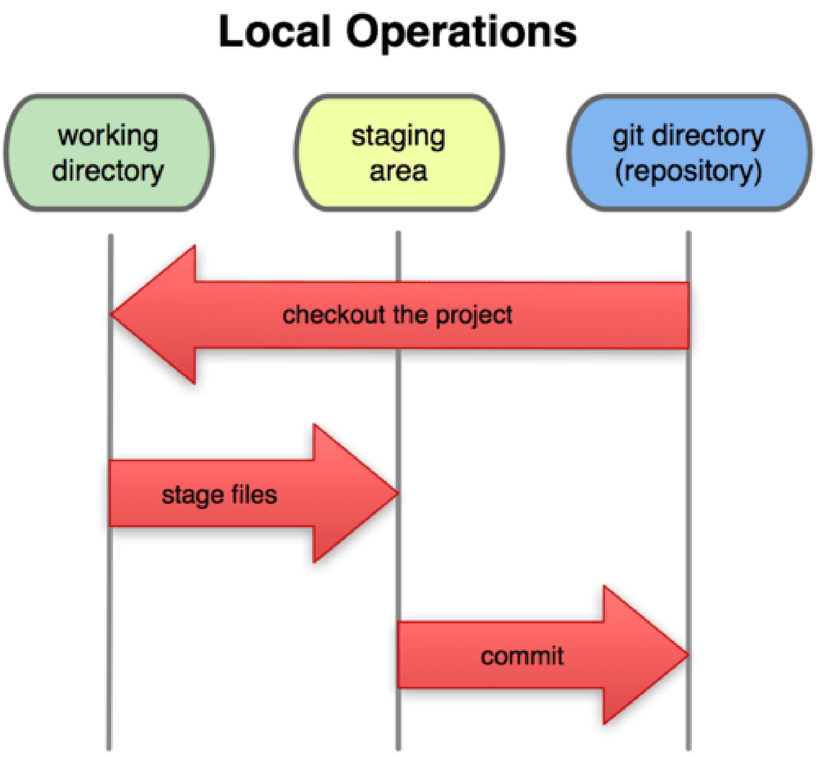
Git has three main states in which every file of a project can be:

- **Modified**: data has been changed locally, but it has not been committed yet.

- **Staged**: data has been tagged to be sent in next commit.
- **Committed**: data is safely stored in a local storage

Therefore, there are three sections in Git:

- **Git directory**: where Git stores the metadata and the database of the elements of the project. This part is what we copy when we clone the repository from other computer.
- **Working directory**: it is a copy of a project version. These files are extracted from the Git database and placed in a folder, ready to be used.
- **Staging area**: it is a simple file stored in Git directory that contains information about the files that will be sent in next commit. It is also called *index*.

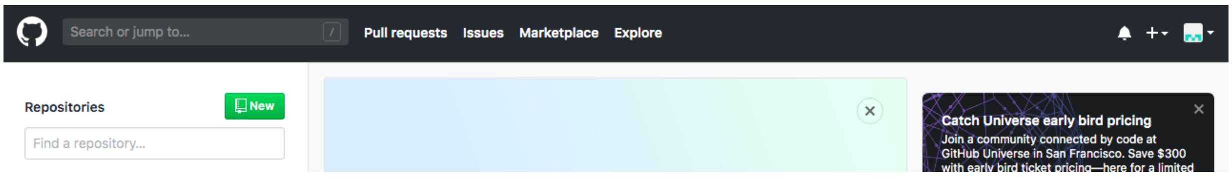


1.7.2.2 Repository

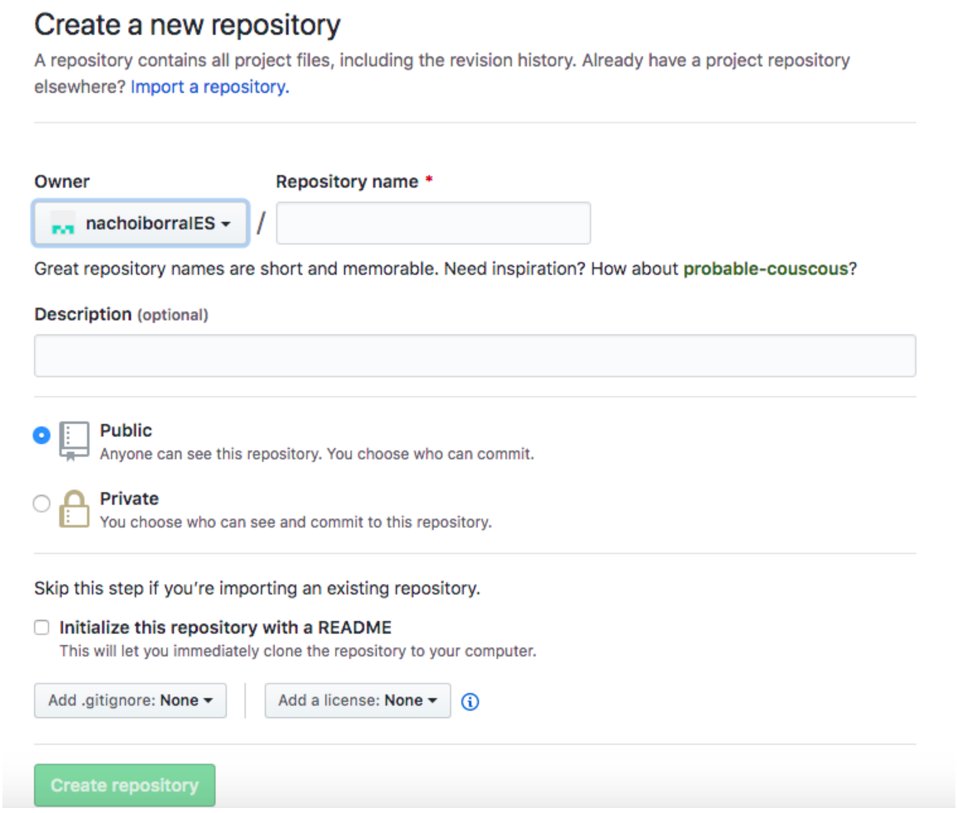
A VCS is normally used for storing projects that can be developed by many people. Either if we develop the project on our own or with other people, we may need to have a remote copy of it, so that we can restore it if there are any problems with our local copy. To do this, we need to have a repository where our remote copy will be stored.

We can create our repository in GitHub, Bitbucket or other platforms. In this case, we are going to use GitHub, which is the most popular one. Besides, it lets us create both public and private repositories.

First of all, we need to sign up in GitHub (if we don't have an account yet). This is the main page once we log in.


Search or jump to...
Pull requests
Issues
Marketplace
Explore
Repositories
New
Find a repository...
Catch Universe early bird pricing
Join a community connected by code at GitHub Universe in San Francisco. Save $300 with early bird ticket pricing—here for a limited

Then, if we want to create a GitHub repository, we must click on the *New* button on the upper left corner, and specify the repository name and some of its general settings: if we want it to be public or private, and if we want to add an initial `README` file (recommended).



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Owner: nachoiborralES / Repository name *

Great repository names are short and memorable. Need inspiration? How about probable-couscous?

Description (optional)

[✓ Public] Anyone can see this repository. You choose who can commit.

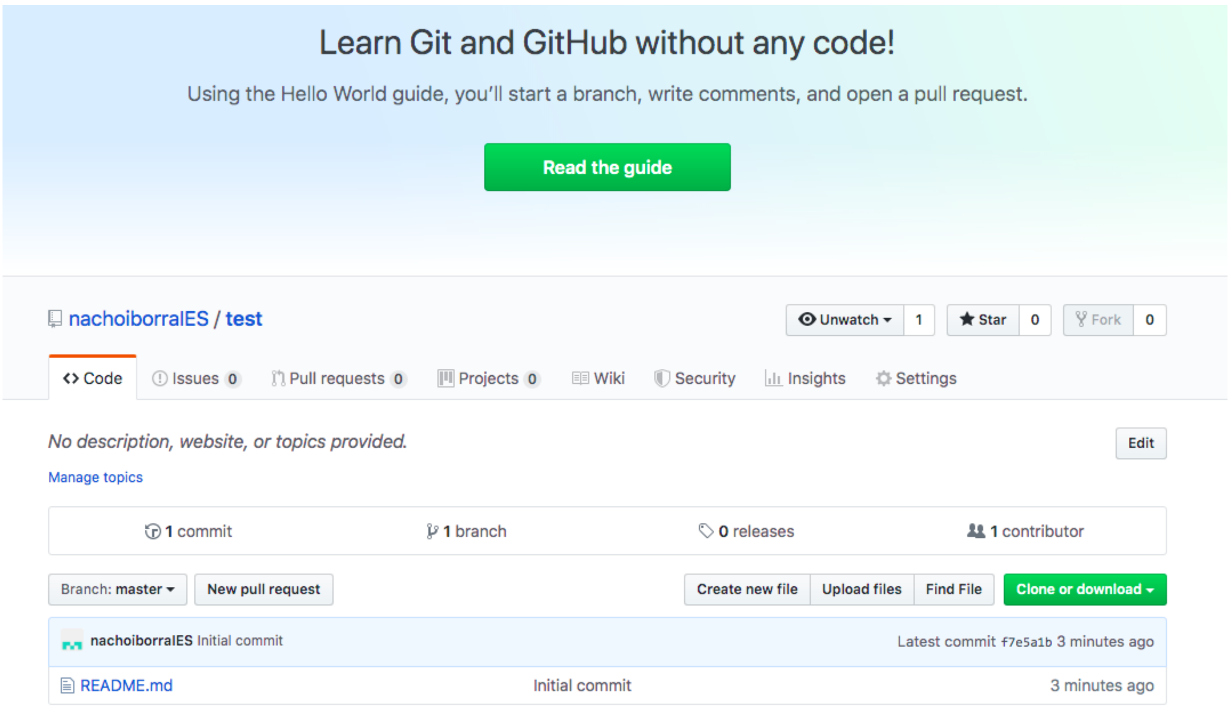
Private — You choose who can see and commit to this repository.

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

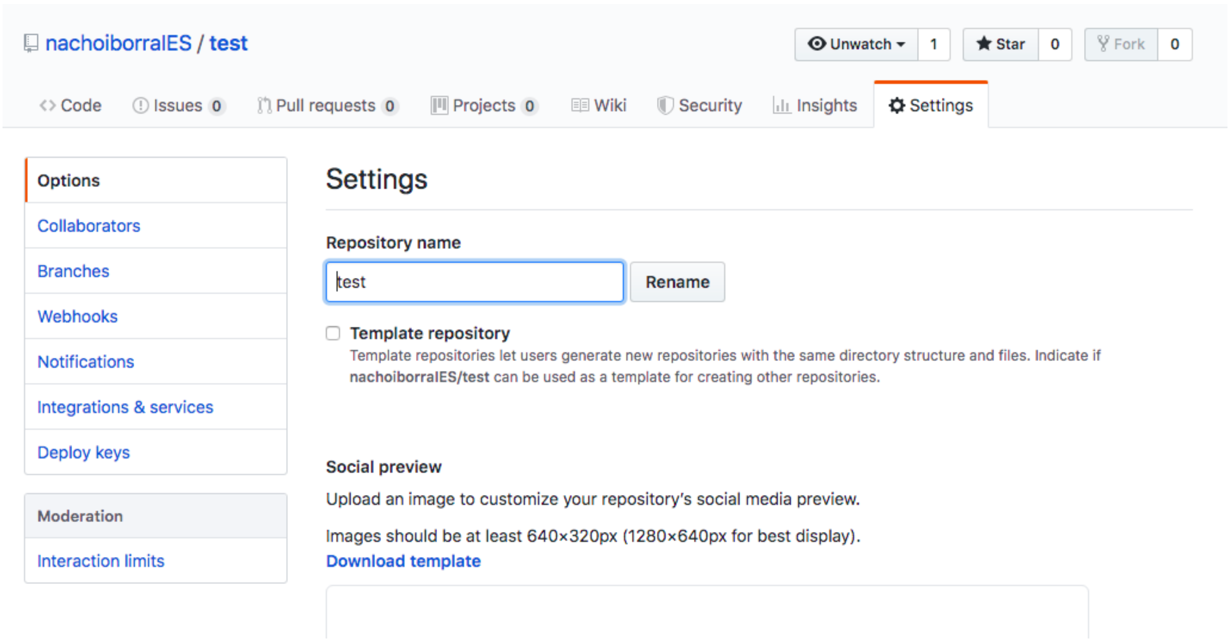
Initialize this repository with a README — This will let you immediately clone the repository to your computer.

Add .gitignore: None   Add a license: None

If we click on the repository name in the left panel of the main view, we can enter this repository. From this page we can, for instance, clone or download the repository, or see the commit history.


Learn Git and GitHub without any code!
Using the Hello World guide, you'll start a branch, write comments, and open a pull request.
Read the guide
nachoiborralES / test
Unwatch 1 Star 0 Fork 0
Code Issues 0 Pull requests 0 Projects 0 Wiki Security Insights Settings
No description, website, or topics provided.
Edit
Manage topics
1 commit 1 branch 0 releases 1 contributor
Branch: master New pull request Create new file Upload files Find File Clone or download
nachoiborralES Initial commit Latest commit f7e5a1b 3 minutes ago
README.md Initial commit 3 minutes ago

If we click on the *Settings* link, we can change some settings. From this page, we can add collaborators from the *Collaborators* menu on the left (this is, other GitHub users) to our project, so that they can also make changes on it. We can also delete the repository, or change its visibility (public/private).


nachoiborralES / test
Unwatch 1 Star 0 Fork 0
Code Issues 0 Pull requests 0 Projects 0 Wiki Security Insights Settings
Options
Collaborators
Branches
Webhooks
Notifications
Integrations & services
Deploy keys
Moderation
Interaction limits
Settings
Repository name
test
Rename
Template repository
Template repositories let users generate new repositories with the same directory structure and files. Indicate if nachoiborralES/test can be used as a template for creating other repositories.
Social preview
Upload an image to customize your repository's social media preview.
Images should be at least 640×320px (1280×640px for best display).
Download template

Before cloning the repository we need to have a Git tool installed in our computer, so that we can commit, push and pull the changes. In this case, we are going to use a tool called GitKraken.

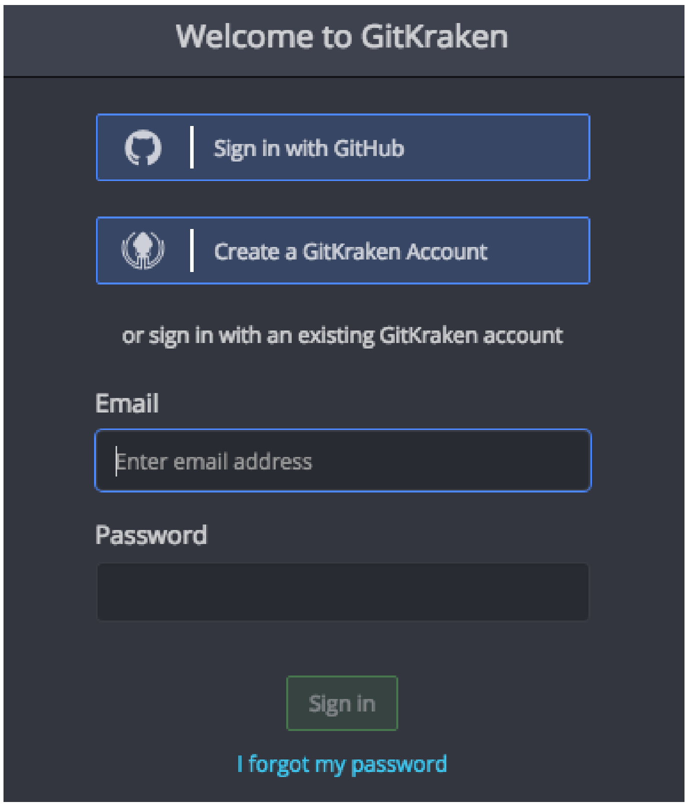
1.7.3. Git tools

Finally, in this section we are going to talk about some useful tools that we can use in order to work with Git repositories.

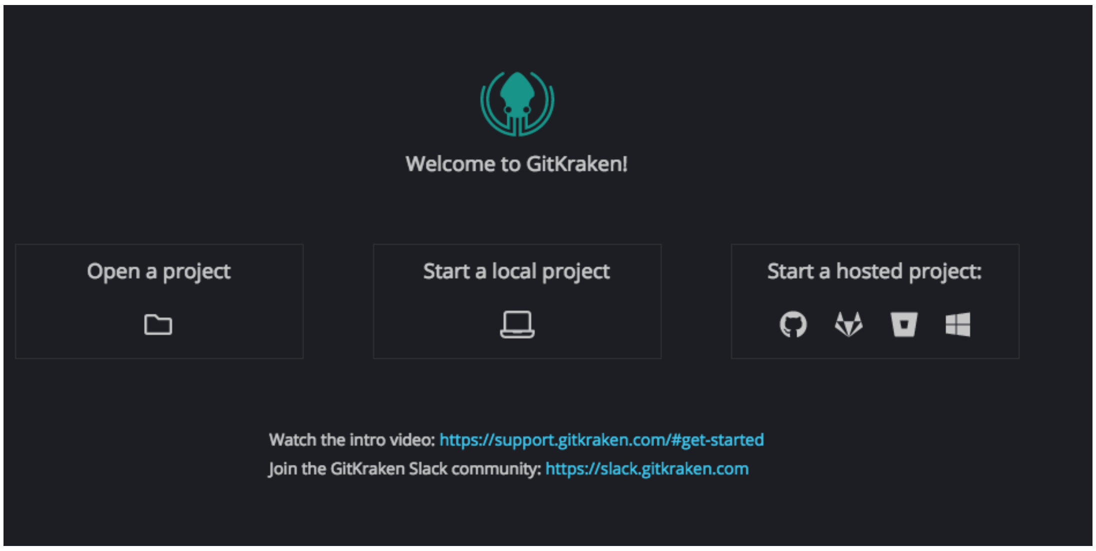
1.7.3.1. GitKraken

GitKraken is a free git tool that can be run under Windows, Linux or Mac OSX. It has also a commercial version, if we want to deal with private repositories, or we need some advanced features.

It can be downloaded from its official website. After the installation, we can start the application. The first time that we launch it, it will ask us to register, either with our GitHub account (if we already have one), or by creating our own GitKraken account. We can follow this option if we don't have any GitHub account, but if we sign up with GitHub, we can easily clone our GitHub repositories later. If we choose this option, then GitKraken will ask you to connect to GitHub from its main web site.



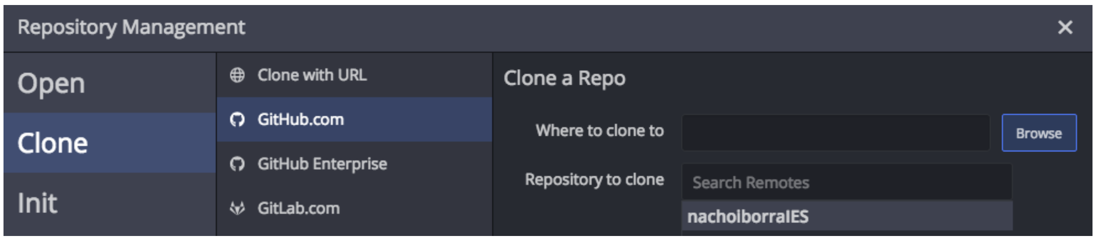
After singing up, we can see the welcome screen:



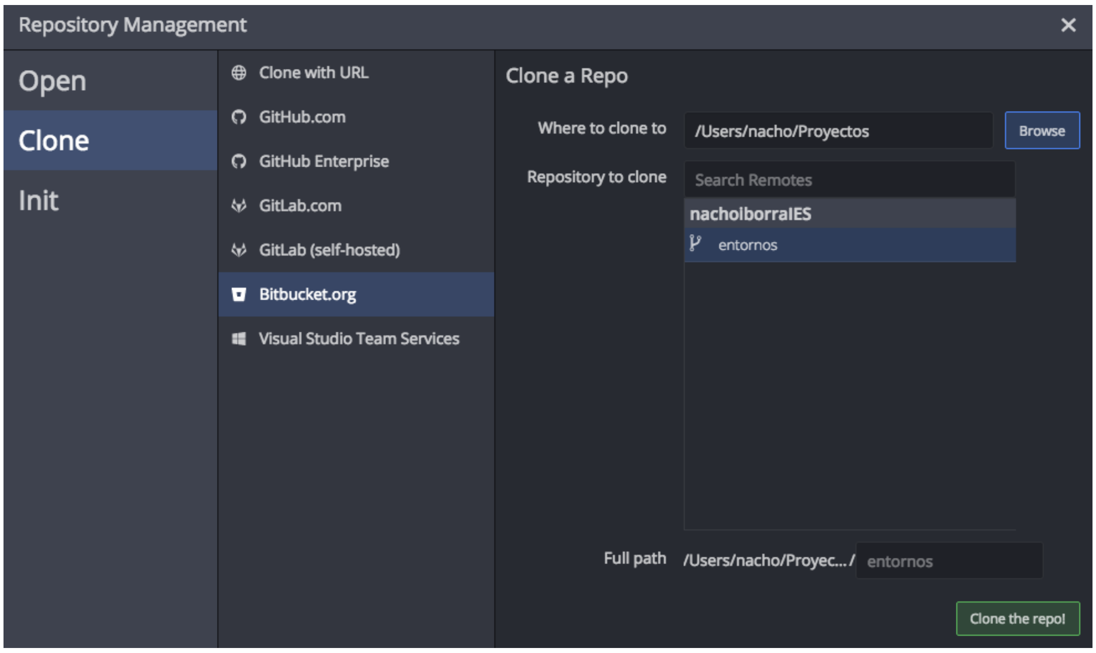
Watch the intro video: https://support.gitkraken.com/#get-started
Join the GitKraken Slack community: https://slack.gitkraken.com

- From the left option (*Open a project*) we can connect to a remote repository (from GitHub or Bitbucket, for instance), and download it.
- From the middle option (*Start a local project*) we can start a new, local project and create files on it. Then, we will be able to commit and upload the changes to a remote repository.
- From the right option (*Start a hosted project*) we can create a remote repository in one of the allowed platforms (GitHub, Bitbucket and some other), so that it connects to this repository and downloads it as well.

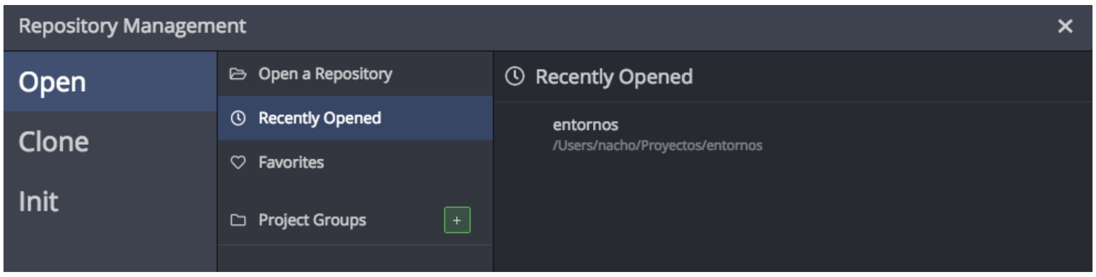
Let's assume that we have already created a remote repository, so we choose first option (*Open a project*). Then, we must choose the git platform to connect with. In our case, we choose GitHub, so we must click on the corresponding button.



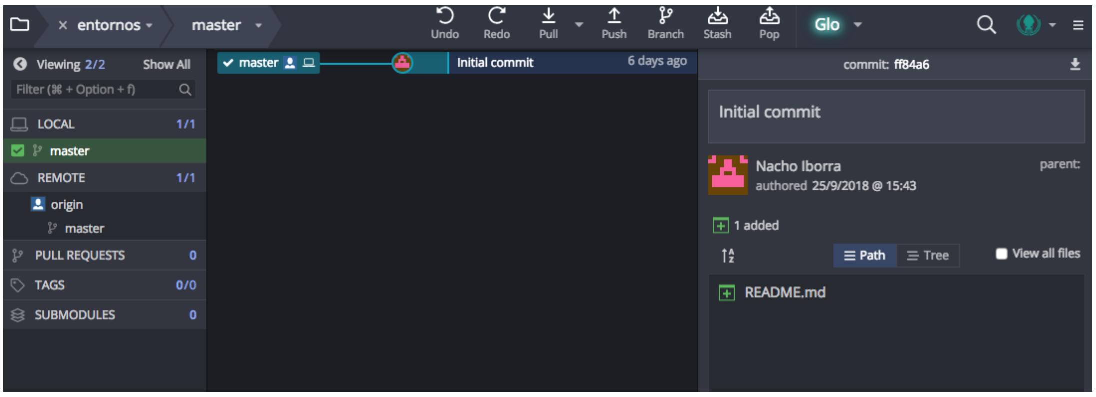
We must choose the repository to clone, and the folder where we want to download the project, in the right panel. Then, we can click on the *Clone the repo!* button in the bottom right corner.



Once our repo is cloned, we can explore it from the *Open a project* option, or by clicking on the folder icon in the upper left corner of GitKraken window.



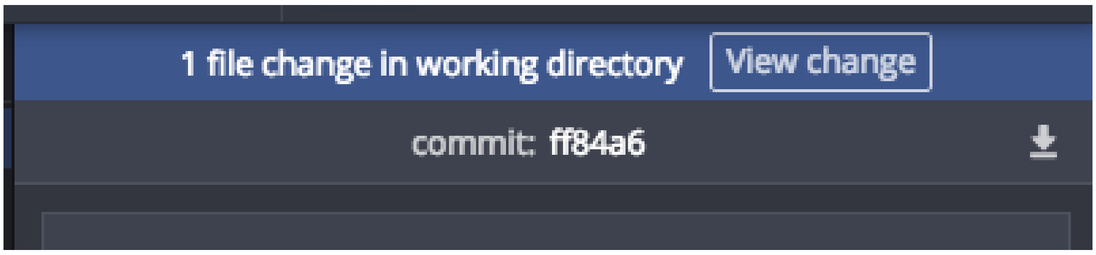
After choosing the project folder (or the project itself if we have opened it recently), we can see its contents:



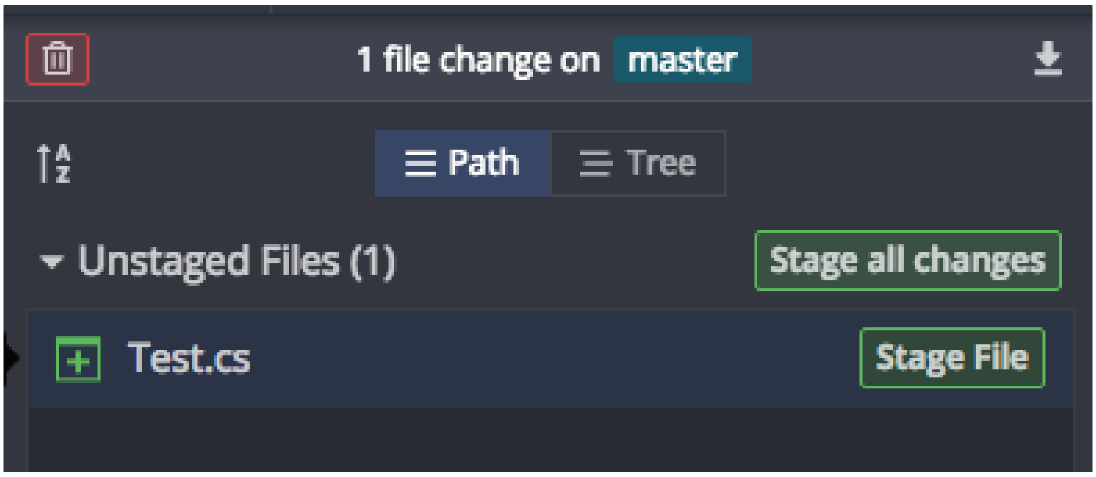
In the right panel we can see the project contents (files and folders). We can see them as paths or as a tree, by clicking on the corresponding buttons of this right part. In the middle panel we have the history of operations over the repository (commits and so on.). Finally, in the left panel we have a list of available branches.

1.7.3.1.1. Committing new changes

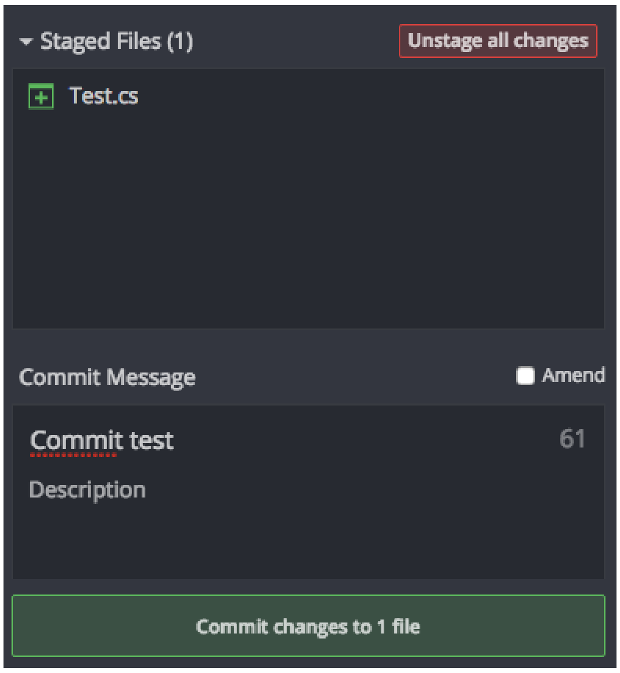
If we add new content to the project (or edit/remove existing content), then all the changes will be shown in the right panel automatically. For instance, if we add a new source file, we can see it this way:



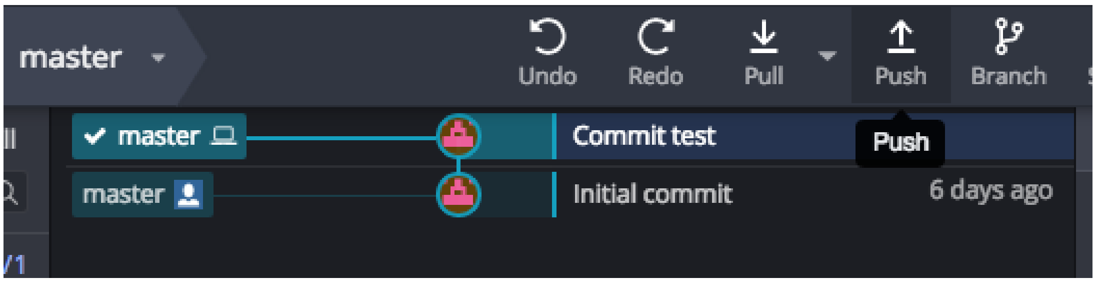
Then, we need to click on the *View Changes* button to see the file(s) that have changed. We can stage them individually (i.e. mark them to be committed in next commit operation), or stage all of them at once by clicking on the *Stage all changes* button.



Next step consists in committing the changes. We must add a commit comment (explaining what's new in this commit) and then click on the *Commit changes* files at the bottom.

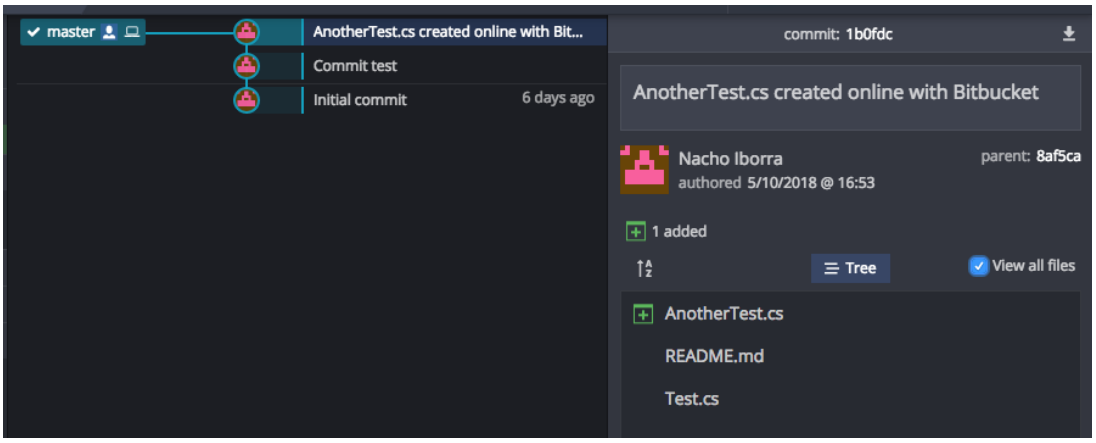


Finally, we need to push the changes to the remote repository. To do this, we go to the middle panel, choose the commit that we want to push and then click on the *Push* button in the upper toolbar.



1.7.3.1.2. Pulling new contents from the server

If we are working in a team, or if we just want to update the changes to another computer, we may need to pull the contents from the repository. If, for instance, someone has uploaded new files to the repository (or changed existing files), if we click on the *Pull* button from the upper toolbar, we will see the files involved. We can also choose the corresponding commit in the list (middle panel) and then view all files in the right panel

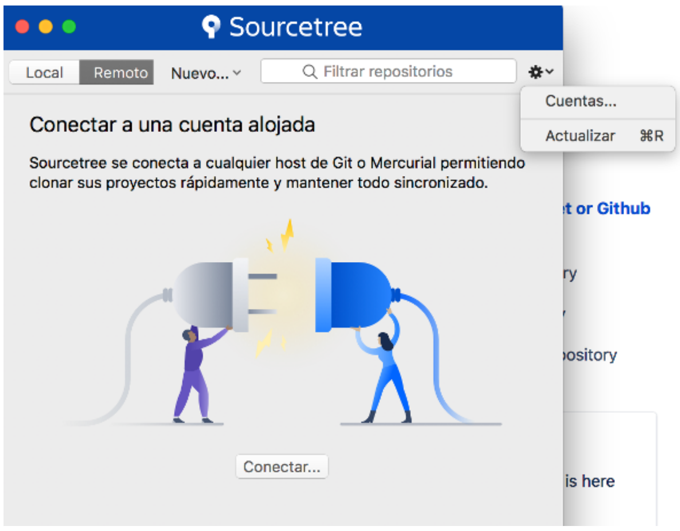


1.7.3.2. SourceTree

We can also use a tool called SourceTree in order to learn how to use Git. You can get it at is official website. This GUI simplifies the interaction with Git repositories, as GitKraken does, so that we don't need to learn every Git console command. It is available under Windows and MacOSX systems (no Linux version available).

Once we download SourceTree (choose the installer according to your operating system), we just install and run it. Then, it will ask us to enter our Atalasian account (the one that we have just created when registering in Bitbucket or GitHub). After logging in, we will see a configuration screen the first time we run the applicaton. We can omit this configuration (click on the *Omit configuration* button in this case). Bitbucket documentation about how to use SourceTree

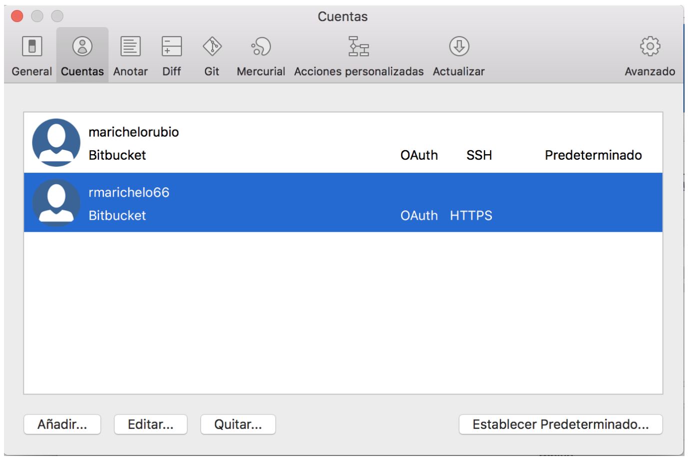
From the main SourceTree screen, we can connect to our account(s) and manage the repositories. We must choose *Remote* from the upper left tabs, and then we choose *Accounts* from the upper right menu.



Conectar a una cuenta alojada

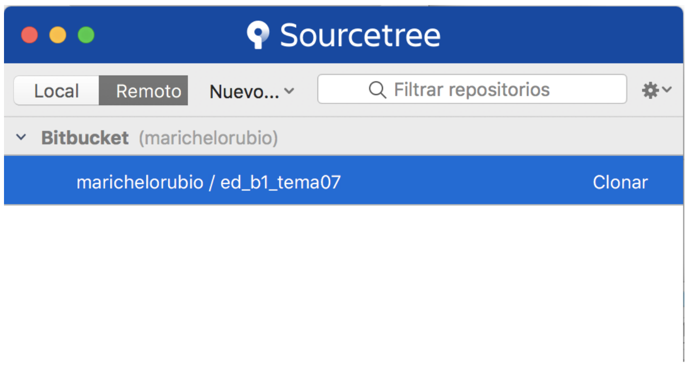
Sourcetree se conecta a cualquier host de Git o Mercurial permitiendo clonar sus proyectos rápidamente y mantener todo sincronizado.

Then, we will see an accounts screen, and we will be able to add our account by clicking on the *Add* button in the bottom left corner. After clicking on the *Connect account* button, we will see our account in the account list.



marichelorubio Bitbucket	OAuth	SSH	Predeterminado
rmarichelo66 Bitbucket	OAuth	HTTPS	

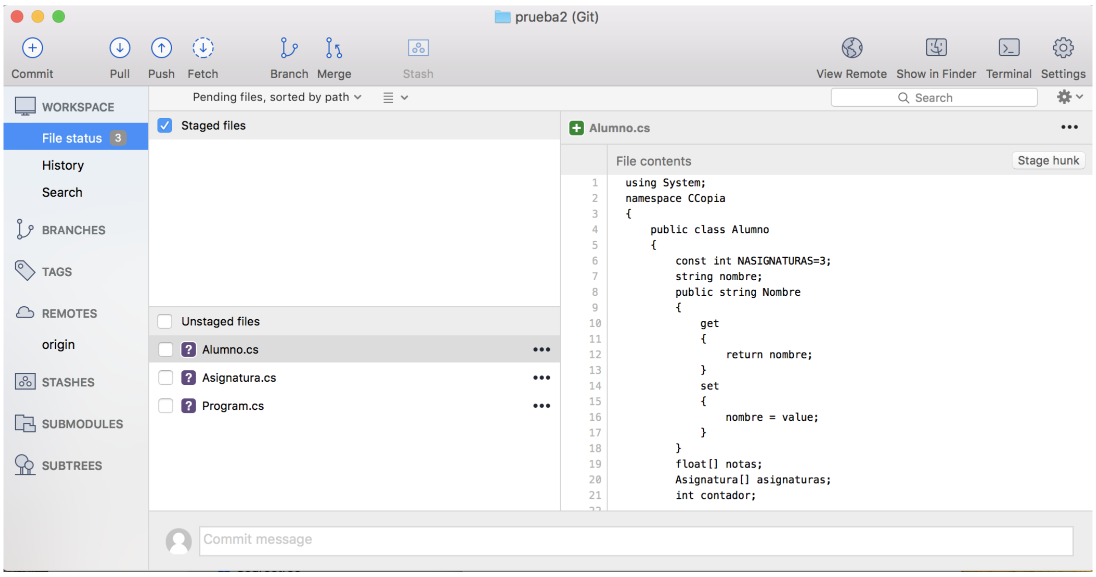
If we close the *accounts* screen and go back to main screen, we will see our remote repository and the *Clone* option:



marichelorubio / ed_b1_tema07

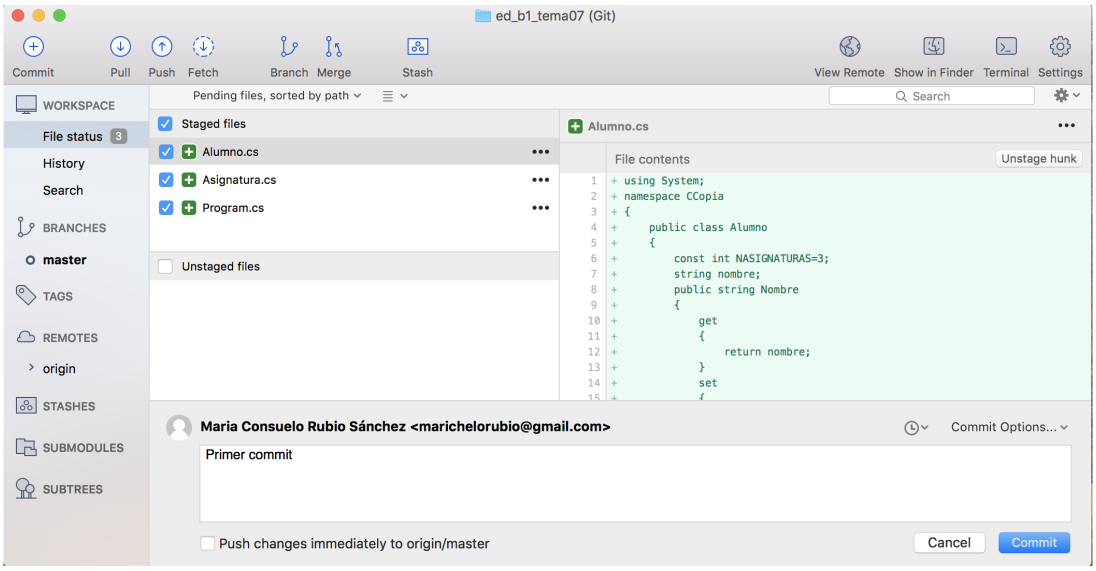
If we click on the *Clone* option, we will then choose the location to download the repository. Then, we can start working on the project using Git for version control. We just need to move to the folder where it has been downloaded.

Our first steps must be focused on sharing the project with the rest of the team. If we open SourceTree and click on the *local* button, then we see our local, cloned repositories. If we choose one of them, we will see this screen:



```
using System;
namespace CCopia
{
    public class Alumno
    {
        const int NASIGNATURAS=3;
        string nombre;
        public string Nombre
        {
            get
            {
                return nombre;
            }
            set
            {
                nombre = value;
            }
        }
        float[] notas;
        Asignatura[] asignaturas;
        int contador;
```

We can see the files that are not staged yet (*unstaged*). If we check them, then they will become staged, so that if we do a *commit*, they will be backed up. So we only need to do a *push* operation in order to upload the changes to the remote repository. When we do the *commit*, we must enter a comment that describes the changes that we have made to te project.



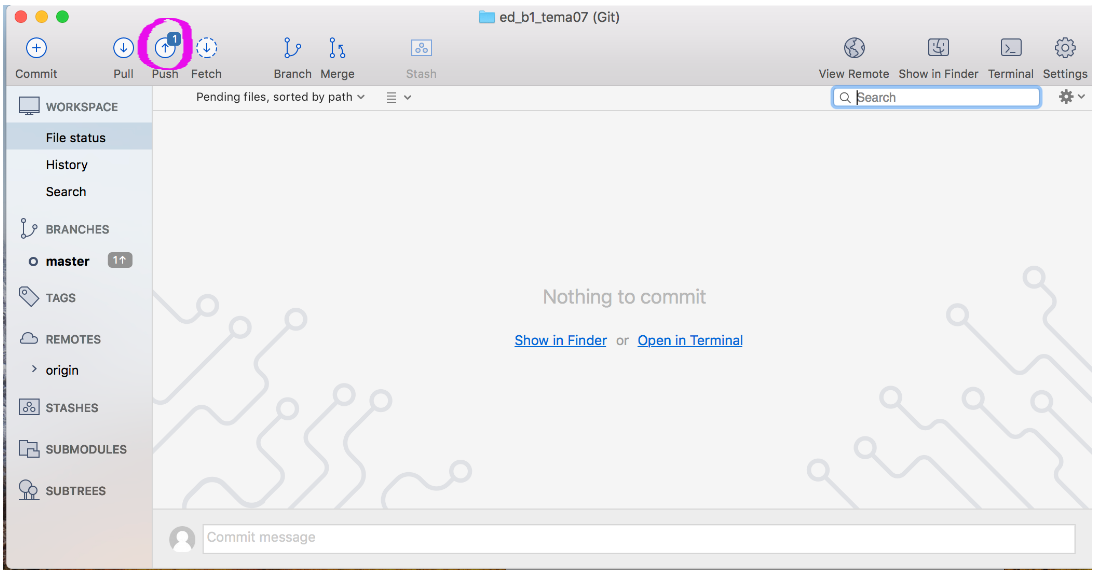
```
+ using System;
+ namespace CCopia
+ {
+     public class Alumno
+     {
+         const int NASIGNATURAS=3;
+         string nombre;
+         public string Nombre
+         {
+             get
+             {
+                 return nombre;
+             }
+             set
+             {
```

Maria Consuelo Rubio Sánchez <marichelorubio@gmail.com>

Primer commit

Push changes immediately to origin/master

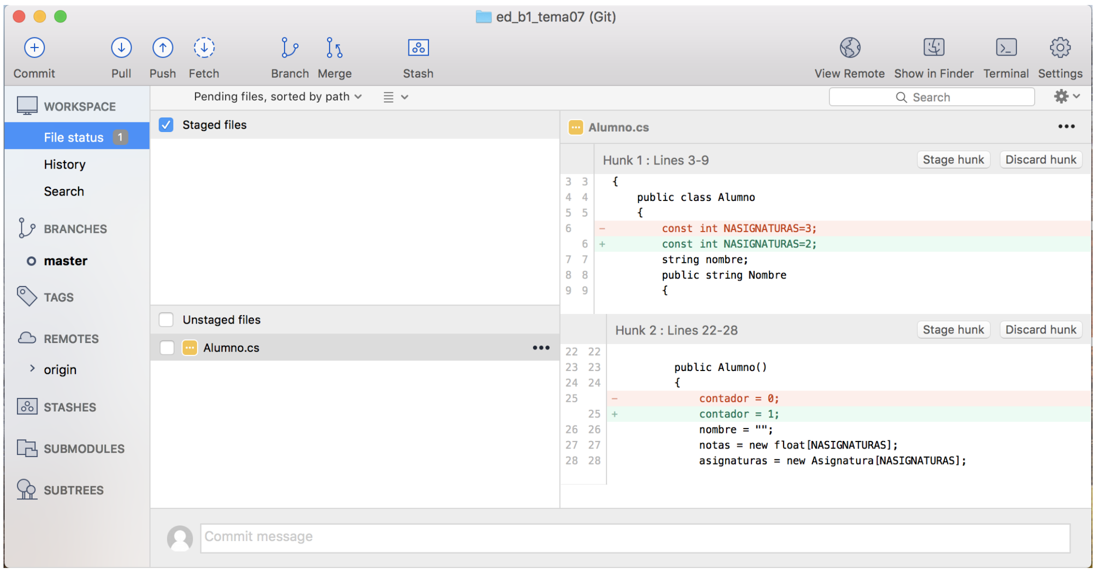
Note that when we commit the changes, they will not be uploaded to the remote repository until we choose the *push* option.



Nothing to commit

Show in Finder or Open in Terminal

If we modify our files, then we can see the changes/differences between versions in the right panel.



Alumno.cs

Hunk 1 : Lines 3-9

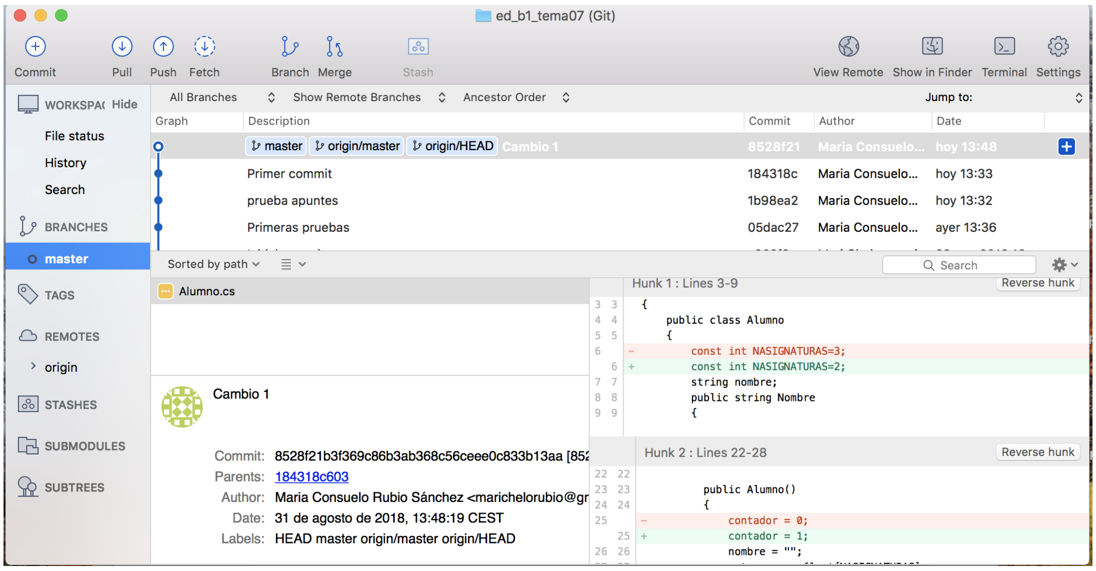
```
   {
       public class Alumno
       {
-          const int NASIGNATURAS=3;
+          const int NASIGNATURAS=2;
           string nombre;
           public string Nombre
           {
```

Hunk 2 : Lines 22-28

```

           public Alumno()
           {
-              contador = 0;
+              contador = 1;
               nombre = "";
               notas = new float[NASIGNATURAS];
               asignaturas = new Asignatura[NASIGNATURAS];
```

Once the changes have been committed and uploaded (pushed), we can see every change in the *History* tab.



Graph	Description	Commit	Author	Date
	master origin/master origin/HEAD Cambio 1	8528f21	Maria Consuelo...	hoy 13:48
	Primer commit	184318c	Maria Consuelo...	hoy 13:33
	prueba apuntes	1b98ea2	Maria Consuelo...	hoy 13:32
	Primeras pruebas	05dac27	Maria Consuelo...	ayer 13:36

Alumno.cs

Cambio 1

Commit: 8528f21b3f369c86b3ab368c56ceee0c833b13aa [852
Parents: 184318c603
Author: Maria Consuelo Rubio Sánchez <marichelorubio@gr
Date: 31 de agosto de 2018, 13:48:19 CEST
Labels: HEAD master origin/master origin/HEAD

Hunk 1 : Lines 3-9

```
   {
       public class Alumno
       {
-          const int NASIGNATURAS=3;
+          const int NASIGNATURAS=2;
           string nombre;
           public string Nombre
           {
```

Hunk 2 : Lines 22-28

```

           public Alumno()
           {
-              contador = 0;
+              contador = 1;
               nombre = "";
```

The panel shows the date, hour, user and comment for every commit.

Fetch option lets us check the changes in remote files, so that we can update our local version through the *pull* option. When we do a *pull* operation there may be conflicts between our local copies and the remote copies. SourceTree will notice us about these conflicts so that we can solve them before pulling the files.

Branch lets us create a branch in the project development. This way, we can both work on the original, master branch of the project and in another update at the same time. For instance, if we are preparing a new software version that requires a lot of changes, we can create a new branch for this new version, and start developing it as we keep on the development of current version. Only when the new version is complete, we can *merge* both branches, then both versions will be combined into a single version.

Proposed exercises:

For the following exercises you need to have a *GitHub* account. So sign up if you don't have any account (here is the official web site).

1.7.3.1. Create a public repository called *CppPrograms* in your GitHub account. Clone it into a local folder using GitKraken or SourceTree. Then, copy the following program in a source file called *hello.cpp* inside that folder.

```
#include <iostream>

using namespace std;

int main()
{
    cout << "Hello world";
    return 0;
}
```

Once you have finished copying the file, commit and push the changes to the remote repository.

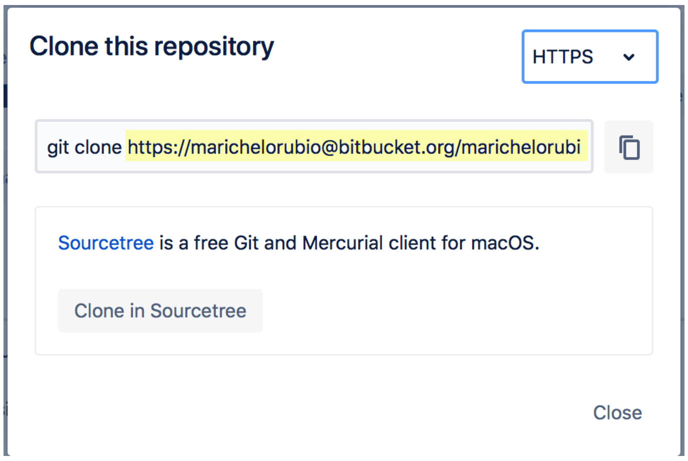
1.7.3.2. Make some changes to previous source file. For instance, change the text that is being printed in the `cout` line. Then, make a second *commit* and its corresponding *push* to update the changes.

1.7.3.3. Share your project with a classmate, and ask him/her to update some of the files. Then, update these files in your local repository through a *pull* operation.

1.7.3.3. Integrating Git with an IDE (Visual Studio Code)

Most of current IDEs lets us use Git internally, so we can perform the main Git operations (commit, push, pull and so on) from the IDE itself. Let's see how it works with Visual Studio Code.

First of all, we must clone our remote repository. We open the command palette (`Control+Shift+P` , or `Cmd+Shift+P` in Mac systems) and we type "clone". Then we will see the `git clone` command, we select it and then we will be asked to enter the remote URL of the repository. We must paste there the URL of our remote GitHub or BitBucket repository (we can see it by clicking on the *Clone* button).

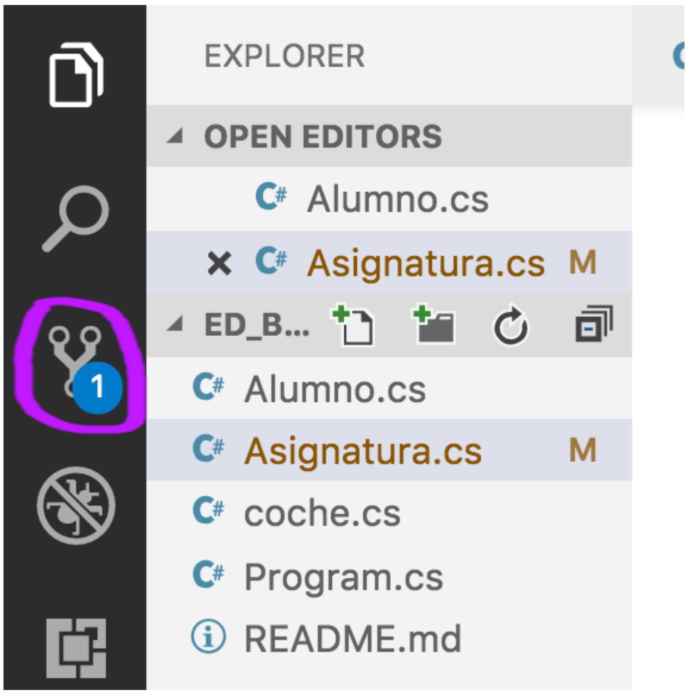

Clone this repository

git clone https://marichelorubio@bitbucket.org/marichelorubi

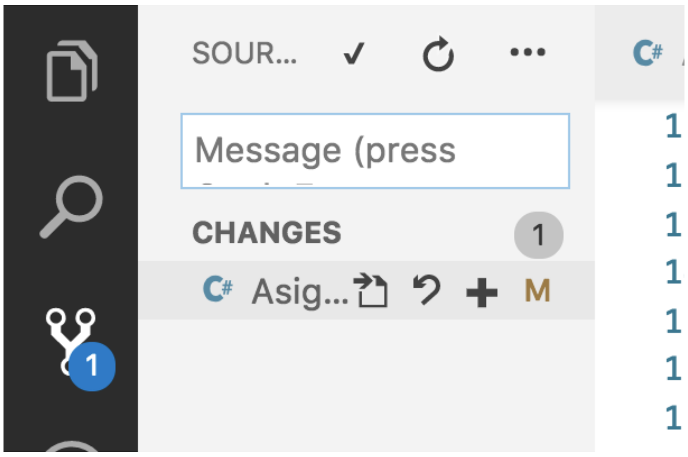
Sourcetree is a free Git and Mercurial client for macOS.

NOTE: if `git clone` command is not recognized by Visual Studio, you may need to install Git before. To do this, you can follow these instructions, depending on your operating system.

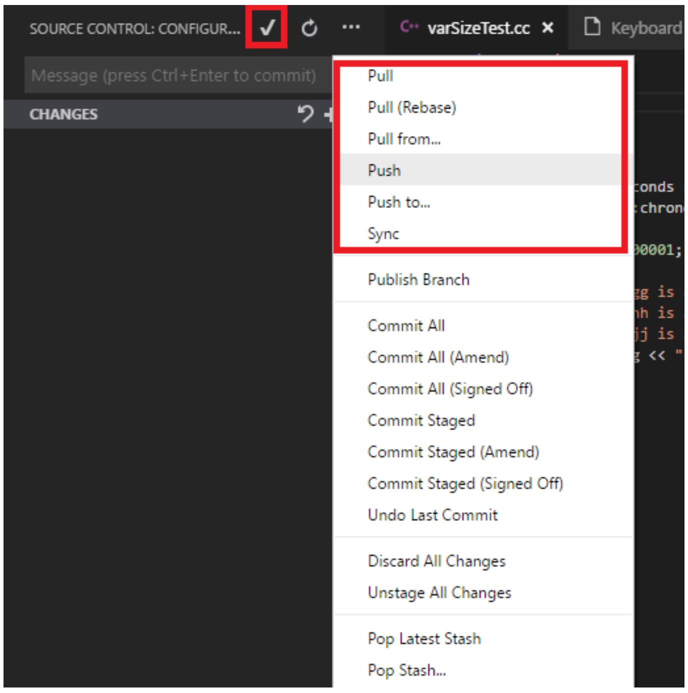
Then, we will see the repository files under Visual Studio Code, and if we made any change to any file, it will be automatically marked with an 'M' next to the file name. This means that the file has been modified from the last version uploaded to the repository. Then we can stage the file by clicking on the *Source Control* icon (see next image). This way, we can see the changes made and the Git options.



Regarding the options, we have *stage* (+), undo changes, *commit* (✓) and more general options in the upper menu.



If we want to push or pull the changes to/from the repository, we must click in the *more options* button (`...`) in the upper left toolbar, and choose the appropriate option.



Proposed exercises:

1.7.3.4. Create a new public repository called *CppProgramsVSCode* in your GitHub account, and then clone it from VS Code.

1.7.3.5. Repeat exercises 1.7.3.1 and 1.7.3.2 using VSCode.

1.7.4. To learn more

You can read more about Git and Git tools from the following links

- https://git-scm.com/book/es/v1
- https://support.gitkraken.com/integrations/github/
- https://confluence.atlassian.com/get-started-with-sourcetree/get-started-with-sourcetree-847359026.html