5 DE ENERO DE 2019 / #GITHUB

Aprenda los conceptos básicos de Git en menos de 10 minutos

por Gowtham Venkatesan

Sí, el título es un clickbait. No hay forma de que pueda *comprender* los conceptos básicos de la tecnología git en solo 10 minutos. Pero puedes acercarte bastante en unos 25 minutos. Y ese es el propósito de este artículo.

If you want to get started on learning about Git technology, you've come to the right place. This is a comprehensive beginner's guide to Git. There are many clients for Git. The technology is all the same no matter the client. But in this guide we'll be using GitHub to understand Git.

Let's get started!

What is Version Control?

> Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later. So ideally, we can place any file in the computer on version control.

Here's Why:

A Version Control System (VCS) allows you to revert files back to a previous state, revert the entire project back to a previous state, review changes made over time, see who last modified something that might be causing a problem, who introduced an issue and when, and more. Using a VCS also means that if you screw things up or lose files, you can generally recover easily. And sometimes you just want to know **"who wrote this crap"**, and having access to that information is worthwhile ?.

So What is Git?

Git is a version-control system for tracking changes in computer files and coordinating work on those files among multiple people. Git is a ***Distributed Version Control System***. So Git does not necessarily rely on a central server to store all the versions of a project's files. Instead, every user "clones" a copy of a repository (a collection of files) and has the ***full*** history of the project on their own hard drive. This clone has *all* of the metadata of the original while the original itself is stored on a self-hosted server or a third party hosting service like GitHub.

Git helps you ***keep track of the changes*** you make to your code. It is basically the history tab for your code editor(With no incognito mode ?). If at any point while coding you hit a fatal error and don't know what's causing it you can always revert back to the stable state. So it is very helpful for debugging. Or you can simply see what changes you made to your code over time.

A simple example of version history of a file.

In the example above, all three cards represent different versions of the same file. We can select which version of the file we want to use at any point of time. So I can jump to and fro to any version of the file in the git time continuum.

Git also helps you ***synchronise code*** between multiple people. So imagine you and your friend are collaborating on a project. You both are working on the same project files. Now Git takes those changes you and your friend made independently and merges them to a single "**Master**" repository. So by using Git you can ensure you both are working on the most recent version of the repository. So you don't have to worry about mailing your files to each other and working with a ridiculous number of copies of the original file. And collaborating long distance becomes as easy as HTML ?.

Git Workflow:

Before we start working with Git commands, it is necessary that you understand what it represents.

What is a Repository ?

Un **repositorio** , también conocido como **repositorio** , no es más que una colección de código fuente.

Repositorio **Remoto** .

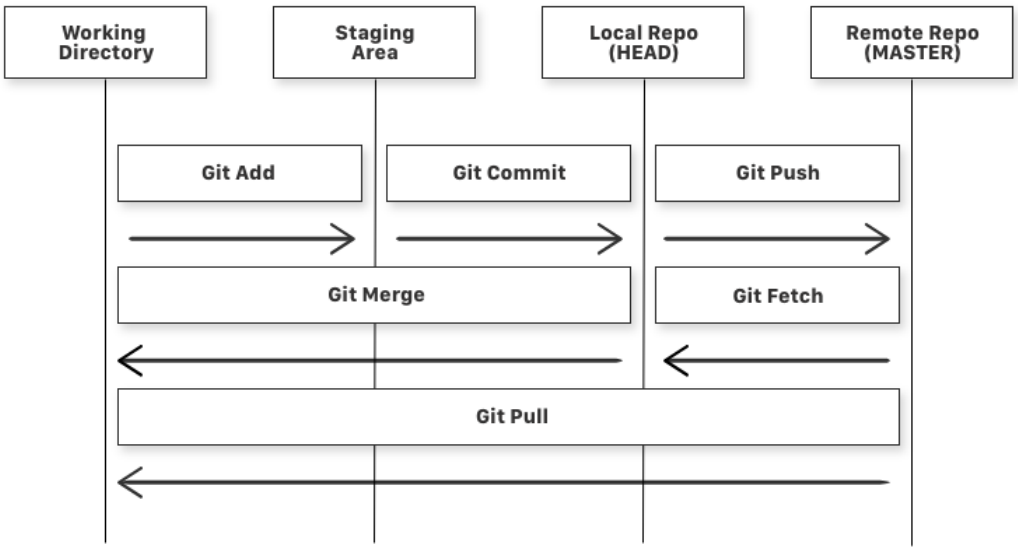


Diagrama de un flujo de trabajo Git simple

Si considera un archivo en su Directorio de trabajo, puede estar en tres estados posibles.

1. **Puede ser escenificado.** Lo que significa que los archivos con los cambios actualizados están marcados para ser confirmados en el repositorio local pero aún no confirmados.
2. **Se puede modificar** . Lo que significa que los archivos con los cambios actualizados aún no están almacenados en el repositorio local.
3. **Se puede cometer** . Lo que significa que los cambios que realizó en su archivo se almacenan de forma segura en el repositorio local.

- `git commit` es un comando que se usa para agregar todos los archivos que están almacenados en el repositorio local.
- `git push` es un comando que se usa para agregar todos los archivos comprometidos en el repositorio local al repositorio remoto. Entonces, en el repositorio remoto, todos los archivos y cambios serán visibles para cualquier persona con acceso al repositorio remoto.
- `git fetch` es un comando que se usa para obtener archivos del repositorio remoto al repositorio local, pero no al directorio de trabajo.
- `git merge` es un comando que se usa para obtener los archivos del repositorio local en el directorio de trabajo.
- `git pull` Este comando se usa para obtener archivos del repositorio remoto directamente en el directorio de trabajo. Es equivalente a ay `git fetch` a `git merge`.

Ahora que sabemos qué es Git y sus terminologías básicas, veamos cómo podemos colocar un archivo en git . Vamos a hacerlo de la manera correcta y de la manera difícil. Sin ninguna aplicación GUI.

Supongo que ya tiene un archivo que desea colocar bajo el control de versiones. Si no, cree una carpeta de muestra llamada 'MuskCult' y coloque algunos archivos de código de muestra en ella.

Paso 0: crea una cuenta de GitHub. Duh.

Si aún no tienes uno, puedes hacer uno aquí .

Paso 1: asegúrese de tener Git instalado en su máquina.

Si está en una **Mac** , inicie la terminal e ingrese el siguiente

```
$ git --version
```

Esto le pedirá que abra un instalador si aún no tiene git. Así que configúralo usando el instalador. Si ya tiene git, solo le mostrará qué versión de git tiene instalada.

Si está ejecutando **Linux** (deb), ingrese lo siguiente en la terminal:

```
$ sudo apt install git-all
```

Si está en **Windows** :

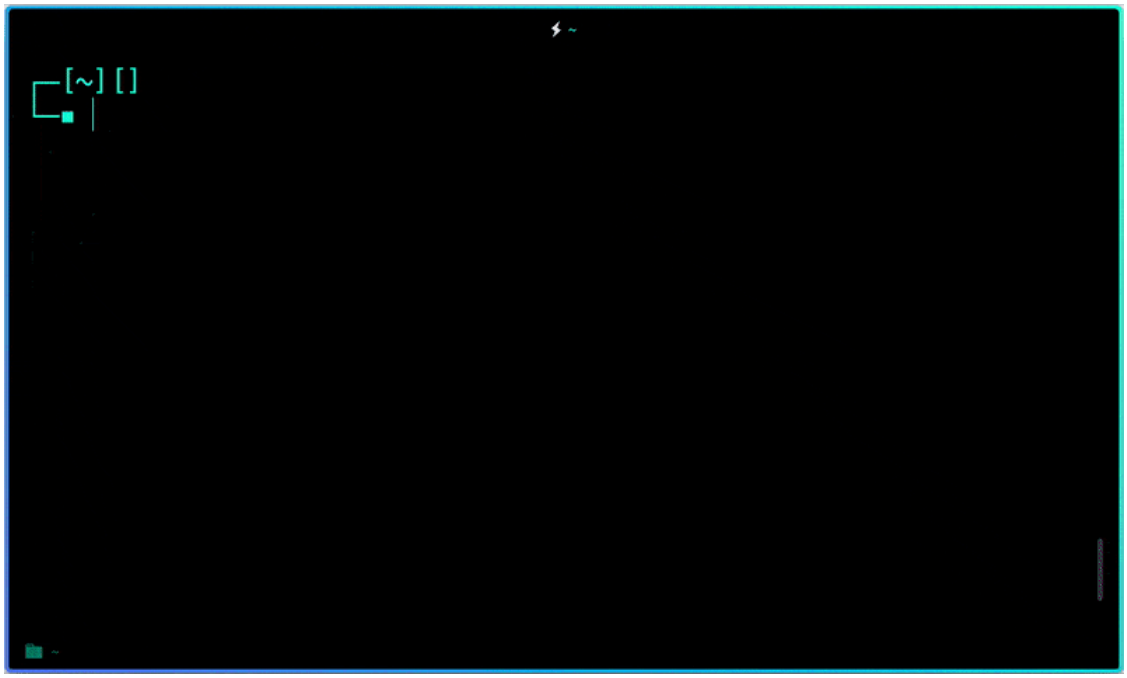
```
$ get a mac
```

Es broma... Relájate... La cantidad de personas que activé... Vaya... Ve a este **enlace** o este **enlace** para obtener más información sobre cómo conseguirlo.

Paso 2: Dile a Git quién eres.

Presentarte. Deslícese. En serio, mencione su nombre de usuario y dirección de correo electrónico de Git, ya que cada confirmación de Git utilizará esta información para identificarlo como el autor.

```
$ git config --global user.name "YOUR_USERNAME"
```



Paso 3: Genere/compruebe su máquina en busca de claves SSH existentes. (Opcional)

¿Porque preguntas? Con el **protocolo SSH** , puede **conectarse y autenticarse** en **servidores y servicios remotos** . Con las claves SSH, puede conectarse a GitHub sin proporcionar su nombre de usuario o contraseña en cada visita.

Siga este **enlace** para obtener más información sobre SSH. Vaya **aquí** para verificar si tiene una clave SSH existente. Vaya **aquí** para generar una clave SSH. Vaya **aquí** para agregar la clave SSH a su cuenta de GitHub. Y finalmente vaya **aquí** para probar su conexión.

```
Instead of : https://github.com/username/reponame

You use    : git@github.com:username/reponame.git

           Note : You can use both ways alternatively
```

Usaré el protocolo SSH en este tutorial.

Paso 4: Vamos a Git

Cree un nuevo repositorio en GitHub. Siga este enlace .
Ahora, ubique la carpeta que desea colocar debajo de git en su terminal.

```
$ cd Desktop/MuskCult
```

Inicializar Git:

Y para colocarlo debajo de git, ingrese:

```
$ touch README.md    # To create a README file for the repositor
$ git init           # Initiates an empty git repository
```

Ahora edite el archivo README.md para proporcionar información sobre el repositorio.

Agregue archivos al área de preparación para la confirmación:

Ahora, para agregar los archivos al repositorio de git para la confirmación:

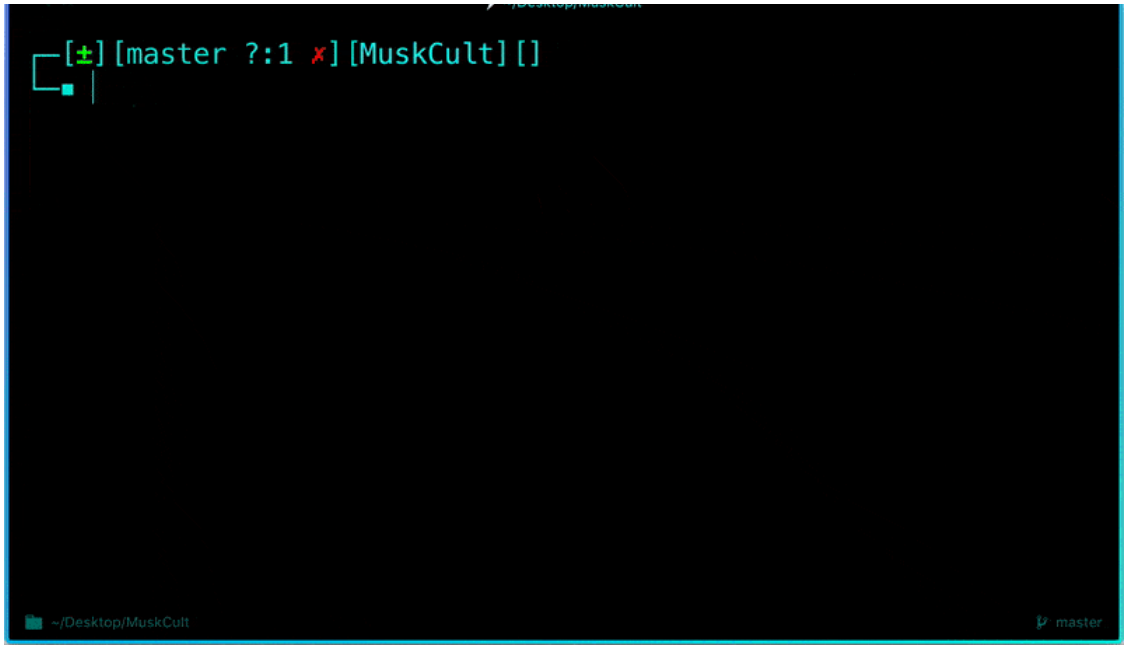
```
$ git add .
# Adds all the files in the local repository and stages them for

OR if you want to add a specific file

$ git add README.md
# To add a specific file
```

Antes de comprometernos, veamos qué archivos están preparados:

```
$ git status # Lists all new or modified files to be committed
```



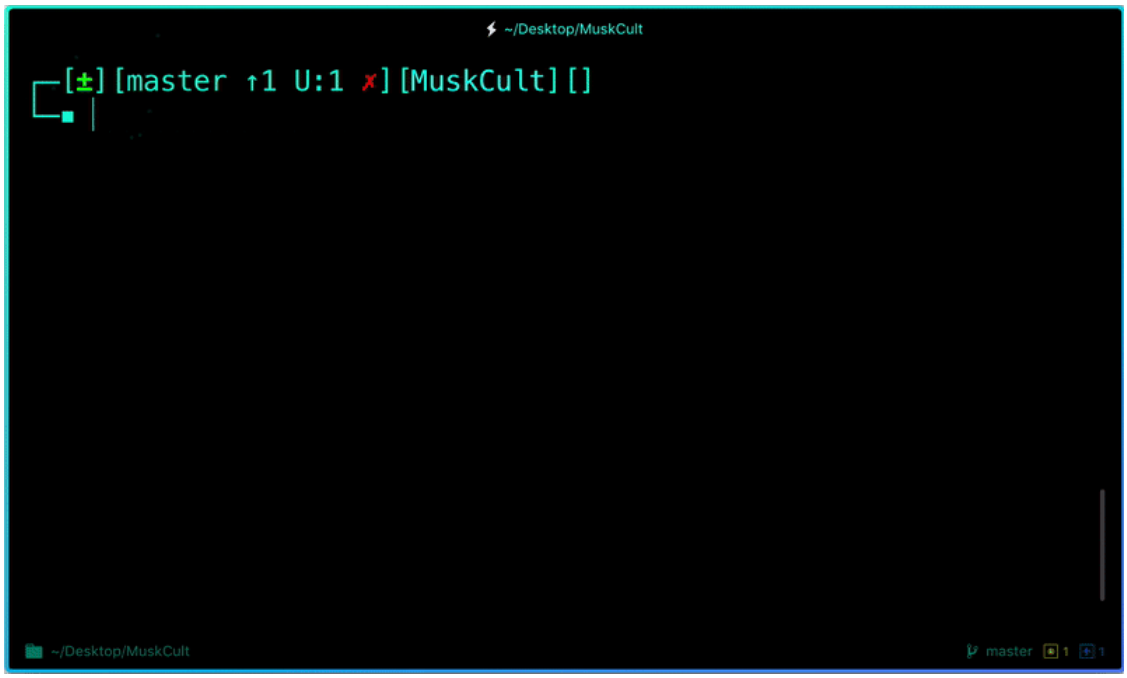
Confirme los cambios que realizó en su Git Repo:

Ahora para confirmar los archivos que agregó a su repositorio de git:

```
$ git commit -m "First commit"
# The message in the " " is given so that the other users can re
```

desorganizar los archivos que acaba de agregar usando:

```
$ git reset HEAD~1
# Remove the most recent commit
# Commit again!
```



Agregue un origen remoto y empuje:

Ahora, cada vez que realice cambios en sus archivos y los guarde, no se actualizarán automáticamente en GitHub. Todos los cambios que hicimos en el archivo se actualizan en el repositorio local. Ahora para actualizar los cambios en el maestro:

```
$ git remote add origin remote_repository_URL
# sets the new remote
```

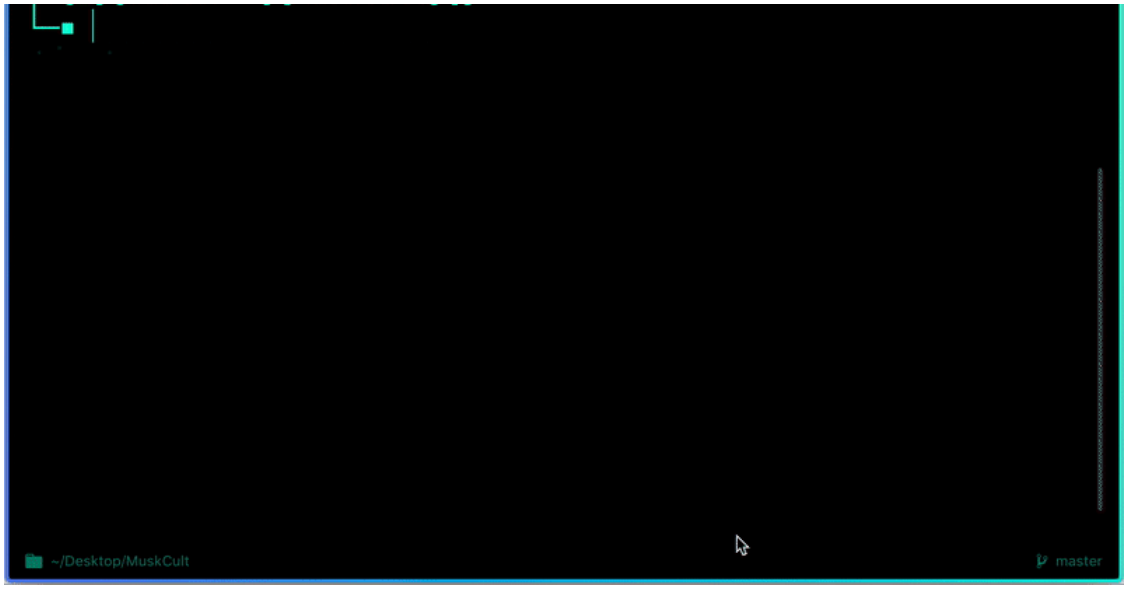
El comando **remoto git** le permite crear, ver y eliminar conexiones a otros repositorios.

```
$ git remote -v
# List the remote connections you have to other repositories.
```

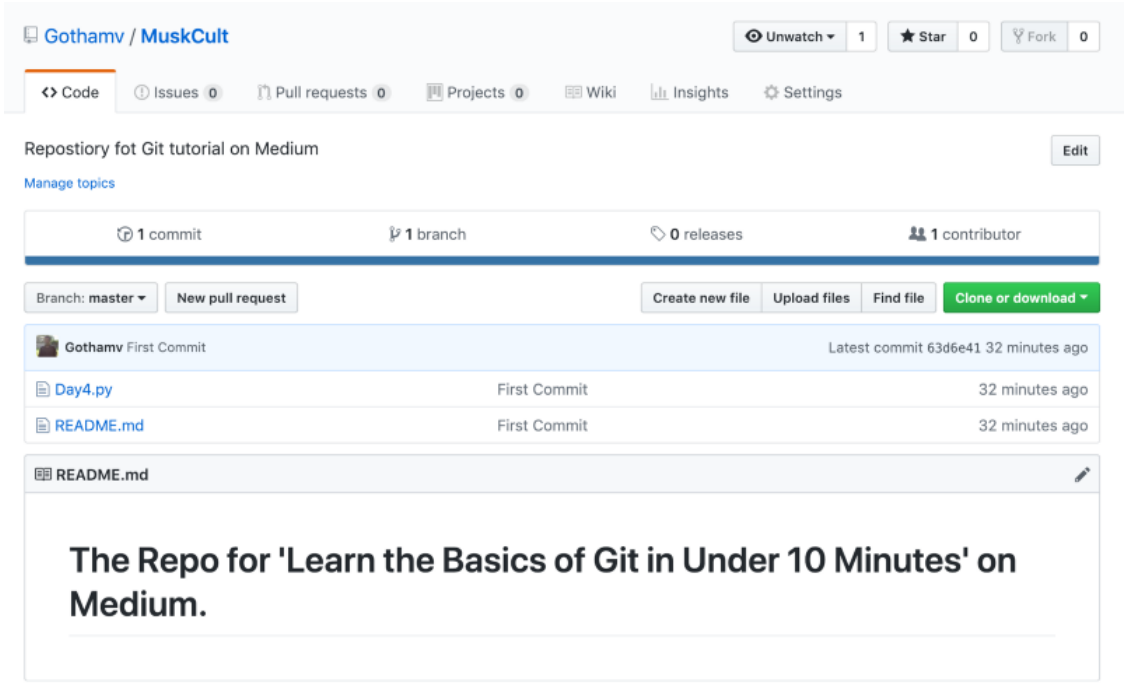
El comando **git remote -v** enumera las URL de las conexiones remotas que tiene con otros repositorios.

```
$ git push -u origin master # pushes changes to origin
```

Ahora, el comando **git push** empuja los cambios en su repositorio local hasta el repositorio remoto que especificó como origen.



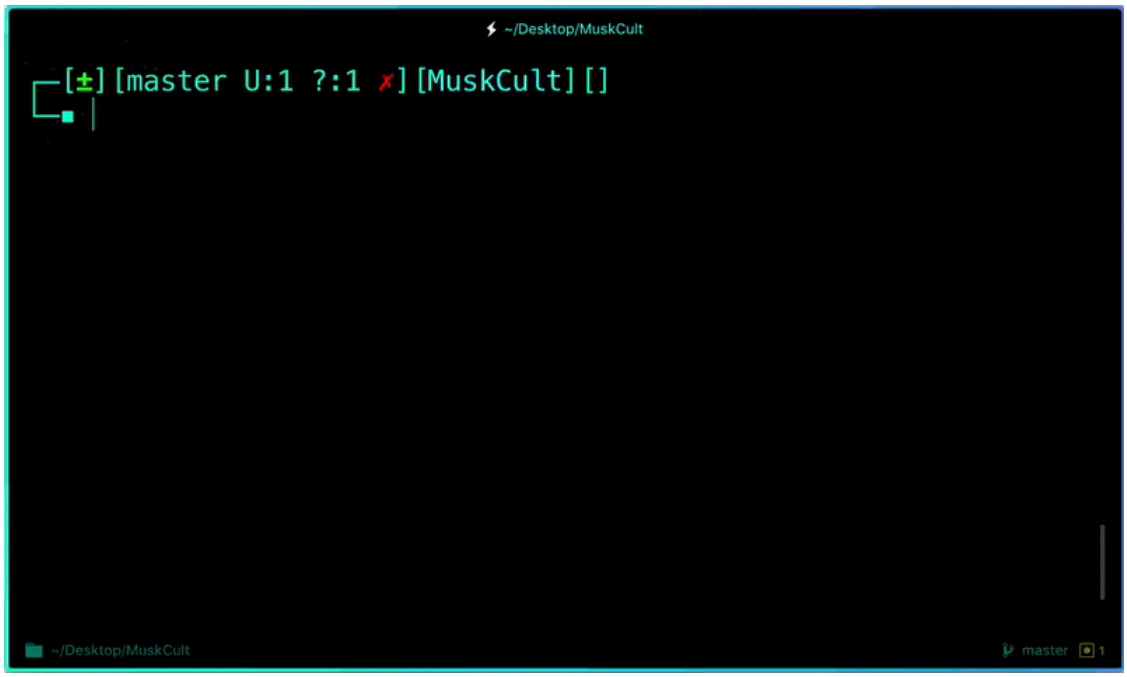
Y ahora, si vamos y revisamos nuestra página de repositorio en GitHub, debería verse así:



Y eso es. Acaba de agregar los archivos al repositorio que acaba de crear en GitHub.

el archivo no coincidirá con la última versión que se envió a git. Para ver los cambios que acabas de hacer:

```
$ git diff # To show the files changes not yet staged
```



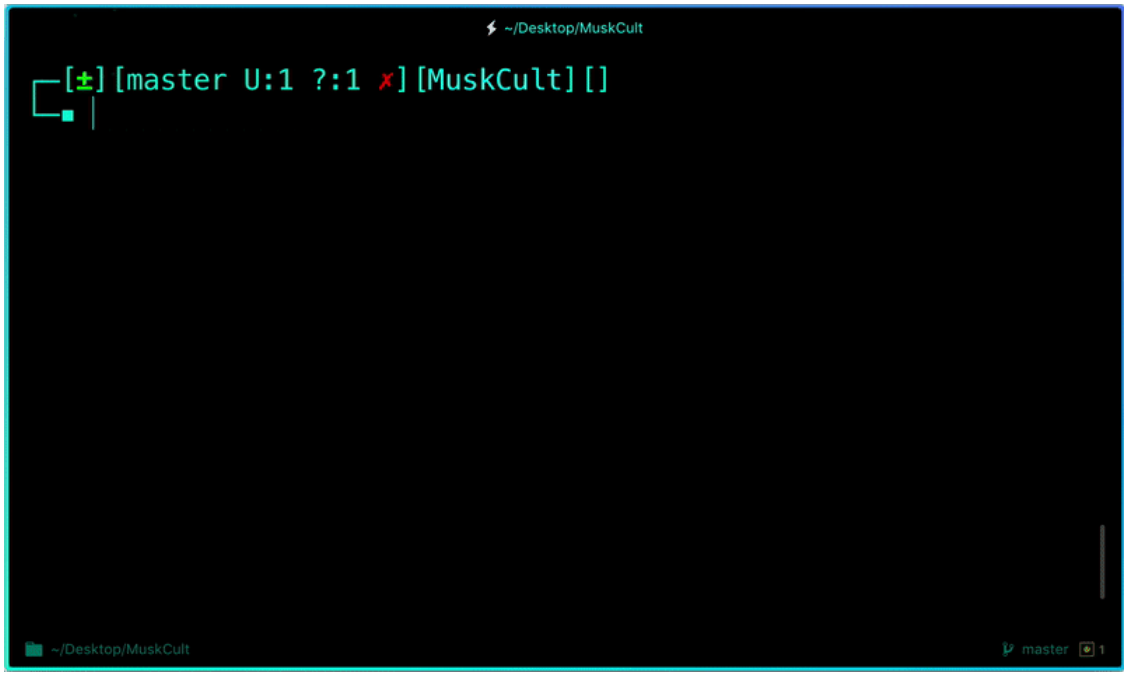
Vuelva a la última versión confirmada en Git Repo:

Ahora puede optar por volver a la última versión confirmada ingresando:

```
$ git checkout .

OR for a specific file

$ git checkout -- <filename>
```



Ver historial de confirmaciones:

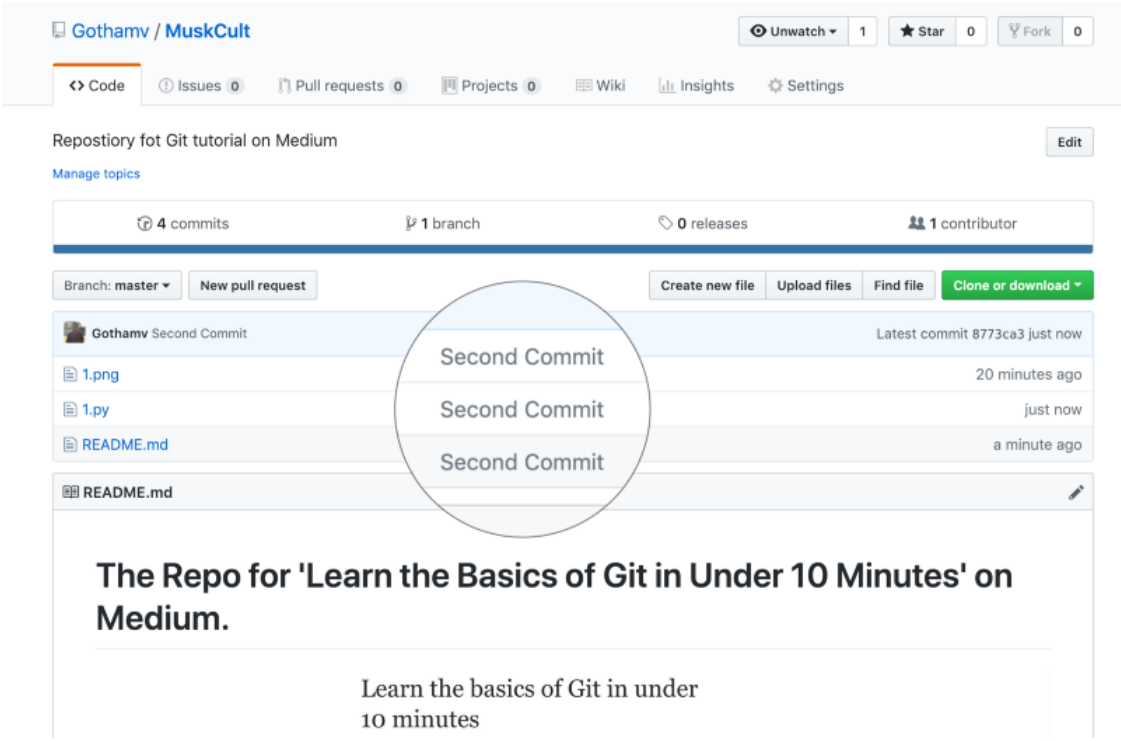
Puede usar el comando **git log** para ver el historial de confirmación que realizó en sus archivos:

```
$ git log
```

Cada vez que realiza cambios que desea que se reflejen en GitHub, los siguientes son los flujos de comandos más comunes:

```
$ git add .
$ git status # Lists all new or modified files to be committed
$ git commit -m "Second commit"
$ git push -u origin master
```

Ahora, si vamos y vemos nuestro repositorio, podemos identificar si



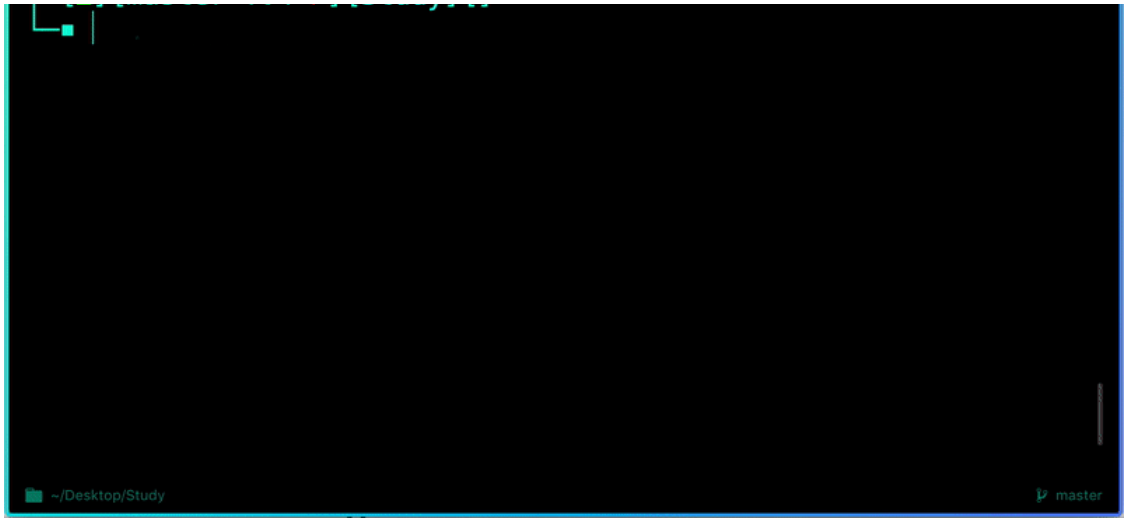
Paso 5: Eso está muy bien... Pero, ¿cómo descargo y trabajo en otros repositorios en GitHub?

Clonación de un repositorio de Git:

Localice el directorio en el que desea clonar el repositorio. Copie el enlace del repositorio que desee e ingrese lo siguiente:

```
$ git clone remote_repository_URL
```

Siéntase libre de continuar y clonar el repositorio que creé anteriormente usando: https://github.com/Gothamv/MuskCult



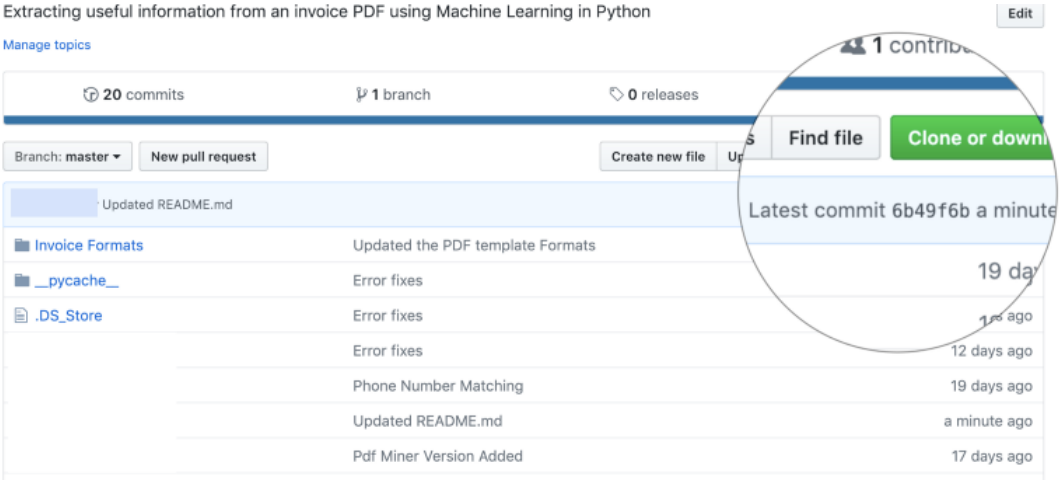
Empujar cambios al Git Repo:

Ahora puede trabajar en los archivos que desee y comprometerse con los cambios localmente. Si desea enviar cambios a ese repositorio, debe agregarse como colaborador para el repositorio o debe crear algo conocido como solicitud de extracción. Vaya y vea cómo hacer uno aquí y envíeme una solicitud de extracción con su archivo de código.

Colaborando:

So imagine you and your friend are collaborating on a project. You both are working on the same project files. Each time you make some changes and push it into the master repo, your friend has to pull the changes that you pushed into the git repo. Meaning to make sure you're working on the latest version of the git repo each time you start working, a git pull command is the way to go.

Now below is an example of a project my friend and I are collaborating on:



There has just been a commit on the repo

So to make sure those changes are reflected on my local copy of the repo:

```
$ git pull origin master
```

Here's two more useful git commands:

```
$ git fetch
    AND
$ git merge
```

In the simplest terms, `git fetch` followed by a `git merge` equals a `git pull`. But then why do these exist?

When you use `git pull`, Git tries to automatically do your work for you. **It is context sensitive**, so Git will merge any pulled commits into the branch you are currently working in. `git pull` **automatically merges the commits without letting you review them first.**

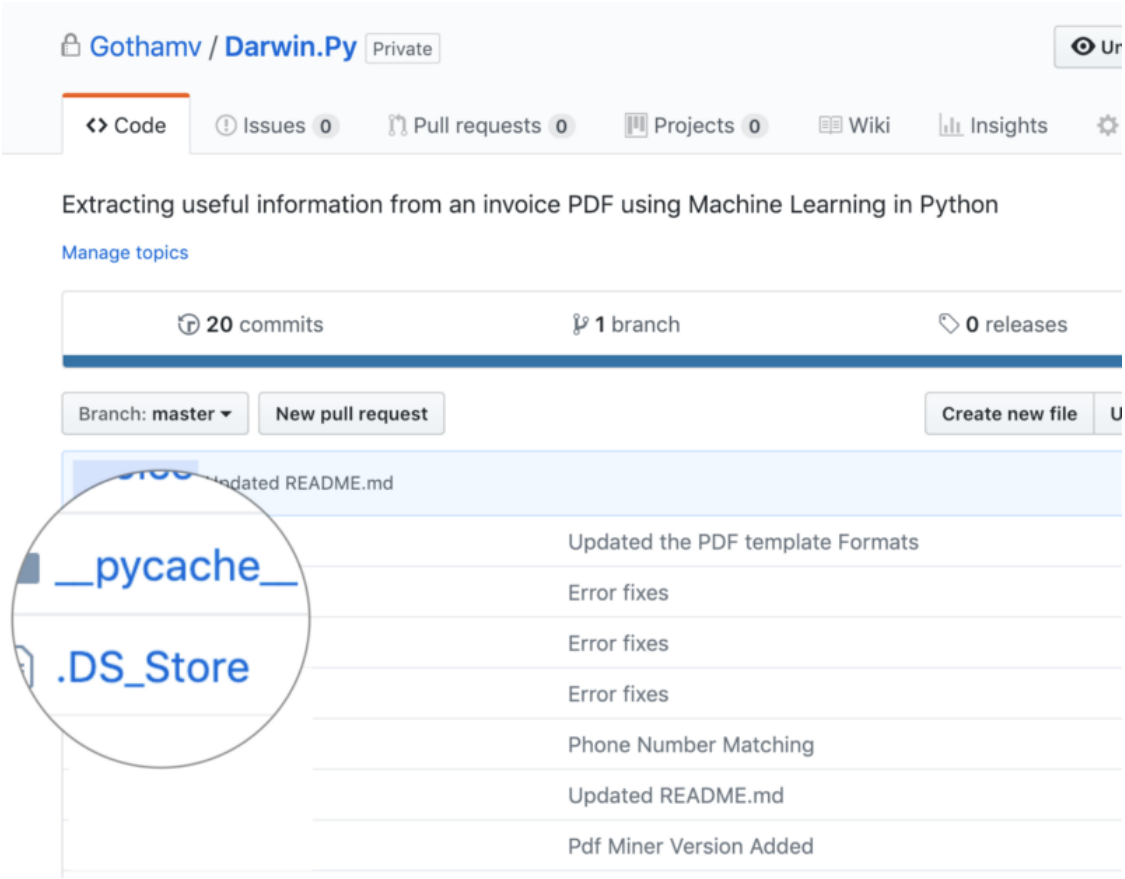
When you `git fetch`, Git gathers any commits from the target branch that do not exist in your current branch and **stores them in your local repository.** However, **it does not merge them with your current branch.** This is particularly useful if you need to keep your repository up to date, but are working on something that might break if you update your files. To integrate the commits into your master branch, you use `git merge`.

One More Thing:

```
.gitignore
```

So what is it?

`.gitignore` tells git which files (or patterns) it should ignore. It's usually used to avoid committing transient files from your working directory that aren't useful to other collaborators, such as compilation products, temporary files IDEs create, etc.



So in the above example, files like __pycache__, .DS_Store are used by the system to store information for faster access. This is not useful for other collaborators. So we can tell git to ignore them by adding a `.gitignore` file.

Use the touch command to create the `.gitignore` file:

And you can add the following patterns to tell git to ignore such files.

```
/*.cmake
/*.DS_Store
/.user
/build
etc. depending upon the files you want git to untrack
```

And that's pretty much it for the basics. Stay tuned for Part 2 which will focus on Branch, Merge, Stash, Rebase etc.

If you enjoyed the article, don't forget to smash that Clap button and make sure you follow me for Part 2.

Peace Out ✌

References :

Adding an existing project to GitHub using the command line - User Documentation

*Putting your existing work on GitHub can let you share and collaborate in lots of great ways. If you are migrating your…*help.github.com

How to undo (almost) anything with Git

*One of the most useful features of any version control system is the ability to "undo" your mistakes. In Git, "undo"…*blog.github.com

Git on the commandline - Don't be afraid to commit 0.3 documentation

There are other ways of installing Git; you can even get a graphical Git

Documentation for GitLab Community Edition, GitLab Enterprise Edition, Omnibus GitLab, and GitLab Runner.docs.gitlab.com

What is the difference between 'git pull' and 'git fetch'?
*Moderator Note: Given that this question has already had sixty-seven answers posted to it (some of them deleted)…*stackoverflow.com

---

If this article was helpful, tweet it.

Learn to code for free. freeCodeCamp's open source curriculum has helped more than 40,000 people get jobs as developers.

Get started

You can make a tax-deductible donation here.

Trending Guides

HTML Font Style
CSS Font
Python Enumerate
HTML Space
<pre> tag in HTML
HTML Forms
Git Delete Branch
How to Code
Git Revert Commit
What is PHP?
Bold Font in HTML
What is HTML?
HTML Button onclick
HTML Image Tag
HTML Underline Text
What is HTTPS?
CSS Display Property
SQL Join Types
SQL Left Join Syntax
Coding Websites
Case Statement in SQL
Remove a Directory in Linux
JavaScript setTimeout()
Text Box in Google Docs
Python Function Examples
Valor atípico en estadísticas
External CSS Stylesheets
Brillo de la pantalla de Windows 10
HTML and CSS Code Examples
Desbloquear a alguien en Instagram

Nuestra organización sin fines de lucro