# Crash Course On Git (10:00)

While beginners are often lost in a sea of jargon and confusing workflows, even advanced developers get overwhelmed by Git.  What is version control anyway? What is a commit message? What does pushing/pulling mean? What's up with...branches?

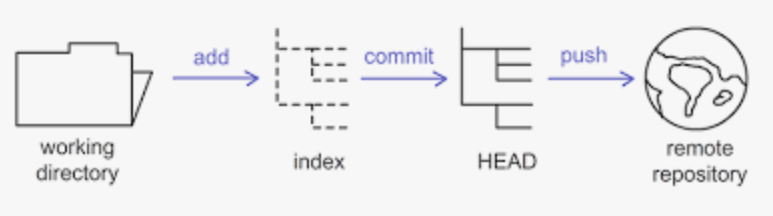
Never fear, for this guide is here to help dispel your confusion and clean up your workflow. Here's a beginner's crash course guide to Git.

Let's start with an explanation of **version control.**

Do you remember how back in your high school days, where you had to write essays? Generally, your teacher would encourage you to start with an outline, then you would move on to a rough draft and revise your work based on feedback until it was ready to submit. Git is the developer's version of keeping track of that exact iterative process, from rough pseudocode all the way to production-ready code.

It is so much easier to write a code if you can go back to something that worked earlier, in case you make some changes in your codebase and accidentally break a lot of things. It saves time to revert to a previous version where your code was working, as opposed to going back and fixing everything you broke. That's where *version control* comes in, giving you the ability to revert to a previous version of your code instead of overriding your current version like a typical Save File every time you save.

A basic, single person Git workflow without any branches looks like this:



1. Initialize a repository (i.e. create a space to store all your files)
2. Create a file
3. Add the file to your index (staging area)
4. COMMIT the file to your local repository
5. Optional: Push the updated repository to an external hosting service, like GitHub or Gitlab, for collaboration or submission.

Let's do these steps together, breaking down what each step means along the way.

First, you'll **open your terminal window** on your machine. As you learned earlier, for Mac users this would be the Mac terminal updated to Zsh and for Windows users, this would be Git Bash.

Now that  you have the terminal open, **navigate to the ROOT folder** of the project you want to use version control with, and**initialize an empty Git repository** by typing this command:

git init

You'll get a message to let you know it worked

Initialized empty Git repository in <<your/filepath/here>>

Now check the status of the repo by typing

git status

You should get a message that says something like

On branch master

No commits yet

Let's create a new file with the touch command. We'll call it index.html.

touch index.html

Check to see if your repository status has changed by typing

git status

You will see that your new file displays in red. This means the file is *staged*, but you have yet to add it and commit it to your repository.

On branch master

No commits yet

Untracked files:

    (use "git add <file>..." to include in what will be committed)

  index.html

This is where things get a little tricky because just because you created a file, it doesn't mean you've placed it in your Git repository yet. To do that, you have to do two things: Add the file to your repository and then commit it.  The first part you have to do only once and then committing the file comes in handy later on as you make more and more changes to the file.

To add the file, type:

git add index.html

Let's check the Git status to see what changes.

On branch master

No commits yet

Changes to be committed:

  (use "git rm --cached <file>..." to unstage)

               new file:   index.html

Perfect! As you can see, the file name has changed to green. This means it has been added to the repository. Now, you must commit it with a message stating what the particular commit changes in our code.

\*\*A note about commits

It may seem strange that after you added a file, you have to carry out an extra step to save it permanently. This is where the magic of version control really shines. Each time you make changes to your code and want to save it, you add the file and execute the commit command.  So, instead of just saving the file and overwriting your previous work, this creates a trackable series of changes, made through separate commits, which you can refer to and even revert to should you need to.

Every commit to Git needs to have a message included. The message should state, in the present tense, what exactly the changes you made do. This step is what keeps track of all your changes, and the commit messages make it easier for you to know what each version does. For this particular commit, we created an empty HTML file, so the commit message will be:

git commit -m "adds empty index.html file"

You should get a message similar to this:

[master (root-commit) 60a0120] adds empty index.html file

 1 file changed, 0 insertions(+), 0 deletions(-)

 create mode 100644 index.html

And that's it! You have successfully created a repo and made your first commit!

To make further commits, all you have to do is stage your file, and commit it, with the commands:

git add <filename>

git commit - "what this commit does"

Now that you are familiar with the Git workflow, you can also try it out with [GitHub desktop.](https://desktop.github.com/) This tool has a GUI, which some people prefer instead of working in the command line. Feel free to use either as you get started, but it is important that you do learn how to use the command line. When you work on enterprise projects, you will have to work through the command line, so it's best to get started on using it now.

Diagrama

Descripción generada automáticamente

GitHub

Username: **betoje**

Email address: [**betoje@gmail.com**](mailto:betoje@gmail.com)

Password: **Denke-0959**

[GitHub](https://github.com/login) (https://github.com/login)

Git Bash (Short Git Bash Prompt)

$ PS1="\A \[\e[97;44m\] \u \[\e[30;43m\] \W \[\e[0m\]`\_\_git\_ps1` $ "



Git useful links

* [git-scm](https://git-scm.com) information

<https://git-scm.com/videos>

<https://git-scm.com/book/en/v2>

* [GitHub](https://github.com) information

<https://skills.github.com/>

<https://docs.github.com/es>

<https://docs.github.com/en/get-started/getting-started-with-git>

<https://docs.github.com/en/get-started/using-git/about-git>

* [stackoverlow](https://www.stackoverflow.com/) [torek](https://stackoverflow.com/users/1256452/torek) <https://stackoverflow.com/a/66309040>

The index has three names: index, staging area, and cache. All refer to the same thing: the place Git sticks these "copies" of each file. The index/staging-area acts as your proposed next commit. When you run git commit, Git is going to package up these copies of the file as the ones to be archived in the snapshot. The copies you have in your working tree are yours; the index / staging-area copies are Git's, ready to go. So, if you change your copies and want the changed copy to be what goes in the next snapshot, you must tell Git: Update the Git copy, in the Git index / staging-area. You do this with git add.3 The git add command means make the proposed-next-commit copy match the working-tree copy. It's the add command that does the updating: this is when Git compresses and de-duplicates the file and makes it ready for archiving, not at git commit time.

* Git & GitHub Crash Course For Beginners ([Traversy Media](https://www.youtube.com/@TraversyMedia))

<https://www.youtube.com/watch?v=SWYqp7iY_Tc&t=626s>

* Simplilearn Git tutorials

<https://www.simplilearn.com/tutorials/git-tutorial#git_tutorial_table_of_contents>

* Miscelaneous

<https://examples.javacodegeeks.com/software-development/git/git-tutorial-beginners/>

https://www.w3schools.blog/category/git

Comandos Git Bash desde Windows Terminal

$ pwd

/home/betoje [WSL Ubuntu]

/Users/betoj [Git Bash Windows] Cambiar carpeta de inicio en configuración

$ git –-version

git version 2.40.0.windows.1 [Git Bash Windows]

Configuración de Git

$ git config –help

$ git config --global user.name "betoje" :GitHub credentials

$ git config --global user.email betoje@gmail.com

$ git config –-global color.ui true

$ git config --global core.editor "code --wait"

Or git config -–global core.editor "'C:\Users\betoj\AppData\Local\Programs\Microsoft VS Code\bin\code ' --wait"

$ git config –list

user.name=betoje

user.email=betoje@gmail.com

color.ui=true

core.editor=code –wait

$ git config –-global --edit edit ~/.gitconfig with core.editor

Creación de repositorio local y primeros commandos Git

$ mkdir Projects

$ mkdir Projects/GitTest

$ cd Projects/GitTest

Initialize this folder as a Git repository in the local system create a new Git instance for a project (local repository)

$ git init

Initialized empty Git repository in C:/Users/betoj/OneDrive/Documentos/Personal/Teaching-Learning/MITxPRO-FullStack/Projects/01-FrontEnd/01-01/GitTest/.git/

$ code file.txt

**Nombre: Alberto**

**Edad: 63**

$ git status

On branch master

No commits yet

Untracked files:

(use "git add <file>..." to include in what will be committed)

file.txt

nothing added to commit but untracked files present (use "git add" to track)

$ git add file.txt add file.txt to the staging area | $ git add . (all files)

$ git status

To see what you have in the staging area, ready to commit. Status of the working tree (differences of the working tree and the staging area or index)

On branch master

No commits yet

Changes to be committed:

~~(use "git rm --cached <file>..." to unstage)~~

(use "git restore --staged <file>..." to unstage)

new file: file.txt

$ git restore –staged file.txt or $ git reset file.txt

to unstage file (file remains in the working tree). $ ~~git rm –-cached file.txt~~ doesn’t work (removes also from working tree)

$ git add file.txt to stage file again

[$ git rm -f file.txt to unstage file and also remove from the working tree]

[$ git reset –-hard [HEAD| 12cad04] to reset your index and working directory to HEAD (last commit) or any previous commit]

12cad04 obtained from $ git log --oneline

$ git commit -m "committing new file.txt"

To take everything that is in the index (staging area) and put into the local repository

[master (root-commit) 41af02a] committing new file.txt

1 file changed, 2 insertions(+)

create mode 100644 file.txt

$ git status

On branch master

nothing to commit, working tree clean

$ code file.txt

**Nombre: Alberto**

**Edad: 63**

**Ciudad: Quito**

$ git diff

Differences between both files

diff --git a/file.txt b/file.txt

index 15529fb..0b33ba2 100644

--- a/file.txt

+++ b/file.txt

@@ -1,2 +1,3 @@

Nombre: Alberto

-Edad: 63

\ No newline at end of file

+Edad: 63

+Ciudad: Quito

\ No newline at end of file

$ git status

On branch master

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git restore <file>..." to discard changes in working directory)

modified: file.txt

no changes added to commit (use "git add" and/or "git commit -a")

$ git add . add all files to the index (staging area), or $ git add –-all | -A

$ git commit -m "add ciudad to file.txt"

[master 12cad04] add ciudad to file.txt

1 file changed, 2 insertions(+), 1 deletion(-)

$ git log

commit 12cad04142424b73901b51e2dafa2bb5cea6d240 (HEAD -> master)

Author: betoje <betoje@gmail.com>

Date: Sun May 14 20:38:44 2023 -0500

add ciudad to file.txt

commit 41af02a009769e0b41518477e592db75662f9f8a

Author: betoje <betoje@gmail.com>

Date: Sun May 14 20:23:51 2023 -0500

committing new file.txt

$ git log –-oneline

12cad04 (HEAD -> master) add ciudad to file.txt

41af02a committing new file.txt

$ git log –-stat –-oneline

$ git log -–graph [--oneline]

Añadir .gitignore

$ code log.txt

**log info**

$ mkdir log-dir

$ cd log-dir

$ code log2.txt

**log2 info**

$ cd ..

$ git status

On branch master

Untracked files:

(use "git add <file>..." to include in what will be committed)

log-dir/

log.txt

nothing added to commit but untracked files present (use "git add" to track)

$ code .gitignore

**log.txt**

**log-dir/** funciona también con /log-dir

$ git status

On branch master

Untracked files:

(use "git add <file>..." to include in what will be committed)

.gitignore

nothing added to commit but untracked files present (use "git add" to track)

Luego de añadir .gitignore, los archivos y directorios allí indicados no aparecen para ser añadidos al staging area o index

$ git add .

$ git status

On branch master

Changes to be committed:

(use "git restore --staged <file>..." to unstage)

new file: .gitignore

$ git commit -m "se añade .gitignore"

Creación de nueva Branch login, Uso de Git merge

$ git branch login

$ git branch

login

\* master

$ git checkout login

Switched to branch 'login'

$ code login.txt

**login info**

$ code file.txt

**Nombre: Alberto**

**Edad: 63**

**Ciudad: Quito**

**Login: true**

$ git status

On branch login

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git restore <file>..." to discard changes in working directory)

modified: file.txt

Untracked files:

(use "git add <file>..." to include in what will be committed)

login.txt

no changes added to commit (use "git add" and/or "git commit -a")

$ git add –all

$ git status

On branch login

Changes to be committed:

(use "git restore --staged <file>..." to unstage)

modified: file.txt

new file: login.txt

$ git commit -m "login code: add login.txt update file.txt"

[login cdedc2d] login code: add login.txt update file.txt

2 files changed, 3 insertions(+), 1 deletion(-)

create mode 100644 login.txt

$ git status

On branch login

nothing to commit, working tree clean

$ git log –-stat –-oneline

cdedc2d (HEAD -> login) login code: add login.txt update file.txt

file.txt | 3 ++-

login.txt | 1 +

2 files changed, 3 insertions(+), 1 deletion(-)

716361b (master) add .gitignore with log.txt and log-dir/

.gitignore | 2 ++

1 file changed, 2 insertions(+)

12cad04 add ciudad to file.txt

file.txt | 3 ++-

1 file changed, 2 insertions(+), 1 deletion(-)

41af02a committing new file.txt

file.txt | 2 ++

1 file changed, 2 insertions(+)

$ ls -a

./ ../ .git/ .gitignore file.txt log.txt log-dir/ login.txt

Aparece: el nuevo archivo login.txt y los cambios en file.txt

$ git checkout master regresa a branch master (no aparece login code)

Switched to branch 'master'

$ ls -a

./ ../ .git/ .gitignore file.txt log.txt log-dir/

No aparece: el nuevo archivo login.txt ni los cambios en file.txt

$ git merge login

Fast-forward

file.txt | 3 ++-

login.txt | 1 +

2 files changed, 3 insertions(+), 1 deletion(-)

create mode 100644 login.txt

$ git branch

login

\* master

$ ls -a

./ ../ .git/ .gitignore file.txt log.txt log-dir/ login.txt

Aparece: el nuevo archivo login.txt en master

$ cat file.txt

Nombre: Alberto

Edad: 63

Ciudad: Quito

Login: true

Aparece: cambios en archivo file.txt (en ese caso no hay conflicto)

[$ git checkout -d login]

Now that your work is merged in, you have no further need for the login branch. You can close the issue in your issue-tracking system, and delete the branch:

[$ git checkout -D login]

Si se han hecho cambios adicionales en la branch login y se quiere forzar su eliminación

Git merge con otro cambio adicional en branch login

[$ git branch login]

Si se ha borrado la branch con los comandos $ git checkout -d|-D login

$ git checkout login

Nos movemos nuevamente a la branch login y creamos un conflicto. Si la hemos borrado con $ git checkout -d login la volvemos a crear nuevamente con $ git branch login

$ code file.txt

**Nombre: Alberto**

**Edad: 63**

**Ciudad: San Diego** se cambia en la misma línea Quito por San Diego

**Login: true**

Se cambia Ciudad: San Diego (no es conflicto)

$ git add .

$ git commit -m 'cambio Ciudad: San Diego en file.txt'

[login a8ae924] cambio Ciudad: San Diego en file.txt

1 file changed, 1 insertion(+), 1 deletion(-)

$ git checkout master

Switched to branch 'master'

$ git merge login

Updating cdedc2d..a8ae924

Fast-forward

file.txt | 2 +-

1 file changed, 1 insertion(+), 1 deletion(-)

$ cat file.txt

Nombre: Alberto

Edad: 63

Ciudad: San Diego

Login: true

$ git log --oneline

a8ae924 (HEAD -> master, login) cambio Ciudad: San Diego en file.txt

cdedc2d login code: add login.txt update file.txt

716361b add .gitignore with log.txt and log-dir/

12cad04 add ciudad to file.txt

41af02a committing new file.txt

$ git reset –-hard cdedc2d

HEAD is now at cdedc2d login code: add login.txt update file.txt

Se regresa al estado anterior cdedc2d antes del cambio Ciudad: San Diego

Uso de Git merge en caso de conflicto

$ git branch conflicto

$ git checkout conflicto

Switched to branch 'conflicto'

$ code file.txt

$ cat file.txt

**Nombre: Alberto**

**Edad: 63**

**Ciudad: San Diego**

**Login: true**

$ git add .

$ git commit -m 'branch conflicto: cambio Ciudad: San Diego'

[conflicto d15e1e6] branch conflicto: cambio Ciudad: San Diego

1 file changed, 1 insertion(+), 1 deletion(-)

$ git log –oneline

d15e1e6 (HEAD -> conflicto) branch conflicto: cambio Ciudad: San Diego

cdedc2d login code: add login.txt update file.txt

716361b add .gitignore with log.txt and log-dir/

12cad04 add ciudad to file.txt

41af02a committing new file.txt

$ git checkout master

Switched to branch 'master'

$ code file.txt

$ cat file.txt

**Nombre: Alberto**

**Edad: 63**

**Ciudad: Luxemburgo**

**Login: true**

$ git add .

$ git commit -m 'branch master: cambio Ciudad: Luxemburgo'

[master 8c82c22] branch master: cambio Ciudad: Luxemburgo

1 file changed, 1 insertion(+), 1 deletion(-)

$ git log –oneline

8c82c22 (HEAD -> master) branch master: cambio Ciudad: Luxemburgo

cdedc2d login code: add login.txt update file.txt

716361b add .gitignore with log.txt and log-dir/

12cad04 add ciudad to file.txt

41af02a committing new file.txt

$ git merge conflicto

Auto-merging file.txt

CONFLICT (content): Merge conflict in file.txt

Automatic merge failed; fix conflicts and then commit the result.

$ git status

On branch master

You have unmerged paths.

(fix conflicts and run "git commit")

(use "git merge --abort" to abort the merge)

Unmerged paths:

(use "git add <file>..." to mark resolution)

both modified: file.txt

no changes added to commit (use "git add" and/or "git commit -a")

$ code file.txt resolver conflictos en VS Code

$ git status

On branch master

All conflicts fixed but you are still merging.

(use "git commit" to conclude merge)

Changes to be committed:

modified: file.txt

$ git commit -m 'se acepta cambio de branch conflicto - Ciudad: San Dieg

o'

[master 636fb66] se acepta cambio de branch conflicto - Ciudad: San Diego

o simplemente con $ git commit Se obtiene un mensaje por default

En GitHub crear repositorio GitTest

Repository HTTPS <https://github.com/betoje/GitTest.git> copy

$ pwd

/c/Users/betoj/OneDrive/Documentos/Personal/Teaching-Learning/MITxPRO-FullStack/Projects/01-FrontEnd/01-01/GitTest

$ git remote

$ git remote add origin https://github.com/betoje/GitTest.git

$ git remote

origin

$ git push -u origin master

Enumerating objects: 20, done.

Counting objects: 100% (20/20), done.

Delta compression using up to 8 threadsSubir a repositorio remoto en GitHub

Compressing objects: 100% (14/14), done.

Writing objects: 100% (20/20), 1.98 KiB | 1015.00 KiB/s, done.

Total 20 (delta 0), reused 0 (delta 0), pack-reused 0

To https://github.com/betoje/GitTest.git

\* [new branch] master -> master

branch 'master' set up to track 'origin/master'.

$ code README.md

**# GitTest**

**## Notas de Git**

<ol>

<li>Comandos Git Bash desde Windows Terminal</li>

<li>Configuración de Git</li>

<li>Creación de repositorio local y primeros commandos Git</li>

<li>Añadir .gitignore</li>

<li>Creación de nueva Branch login, Uso de Git merge</li>

<li>Git merge con otro cambio adicional en branch login</li>

<li>Uso de Git merge en caso de conflicto</li>

<li>Subir a repositorio remoto en GitHub</li>

</ol>

<p>Incluye archivo <strong>GitNotes.docx</strong></p>

$ cp "C:\Users\betoj\OneDrive\Documentos\Personal\Teaching-Learning\MITx

PRO-FullStack\Module01\Week01-Setting\_Up\_Your\_System\GitNotes.docx" .

Copiar archive GitNotes.doc en este directorio

$ git add .

$ git commit ­­-m "add README.md & GitNotes.docx"

Enviar los cambios nuevamente a repositorio remoto en GitHub

$ git push