Git

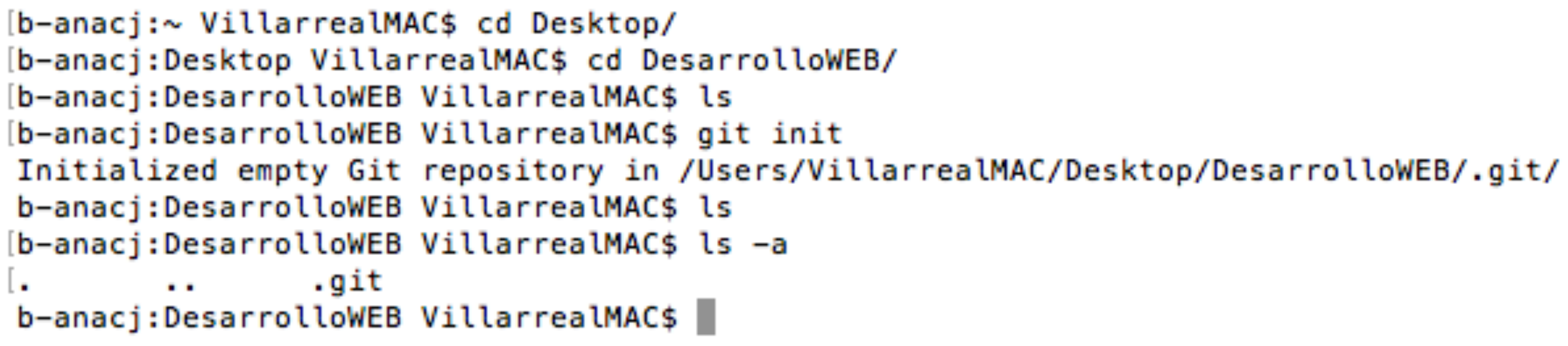
Jhon Edisson Villarreal Padilla

Desarrollo WEB

Git allows groups of people to work on the same documents (often code) at the same time, and without stepping on each other's toes. It's a distributed version control system.

To initialize a Git repository here, type the following command:

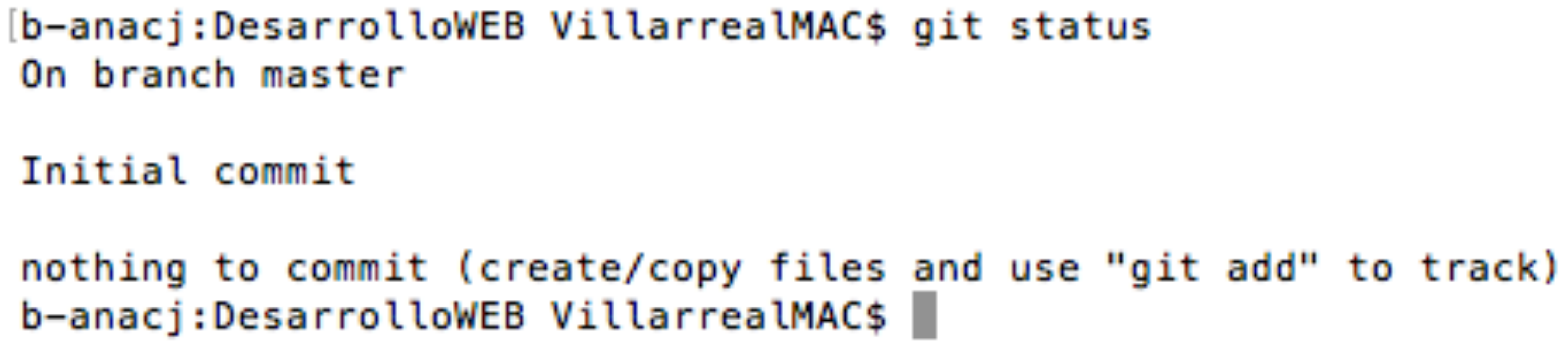
git init



```
[b-anacj:~ VillarrealMAC$ cd Desktop/
[b-anacj:Desktop VillarrealMAC$ cd DesarrolloWEB/
[b-anacj:DesarrolloWEB VillarrealMAC$ ls
[b-anacj:DesarrolloWEB VillarrealMAC$ git init
 Initialized empty Git repository in /Users/VillarrealMAC/Desktop/DesarrolloWEB/.git/
 b-anacj:DesarrolloWEB VillarrealMAC$ ls
[b-anacj:DesarrolloWEB VillarrealMAC$ ls -a
[.        ..       .git
 b-anacj:DesarrolloWEB VillarrealMAC$ 
```

Checking the Status

Next up, let's type the **git status** command to see what the current state of our project is:

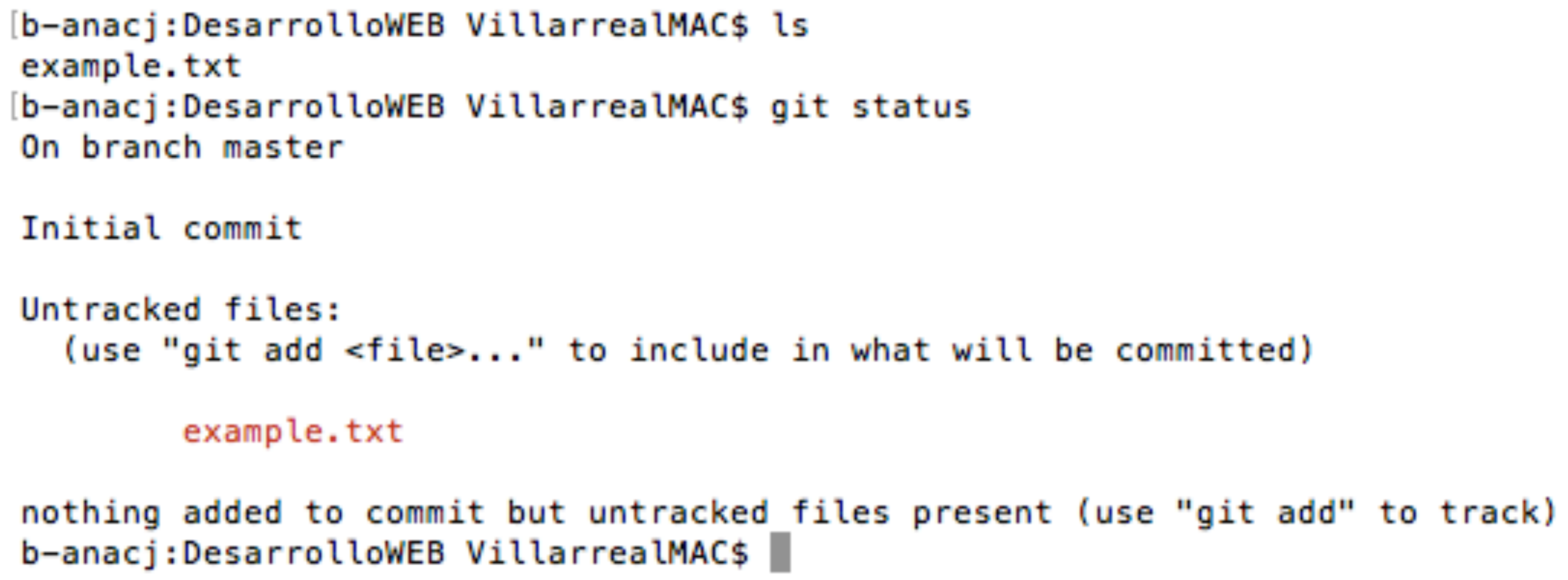


```
[b-anacj:DesarrolloWEB VillarrealMAC$ git status
On branch master

Initial commit

nothing to commit (create/copy files and use "git add" to track)
b-anacj:DesarrolloWEB VillarrealMAC$
```

Adding & Committing

You should created a file called example.tx in the repository folder.

You should run the git status command again to see how the repository status has changed.

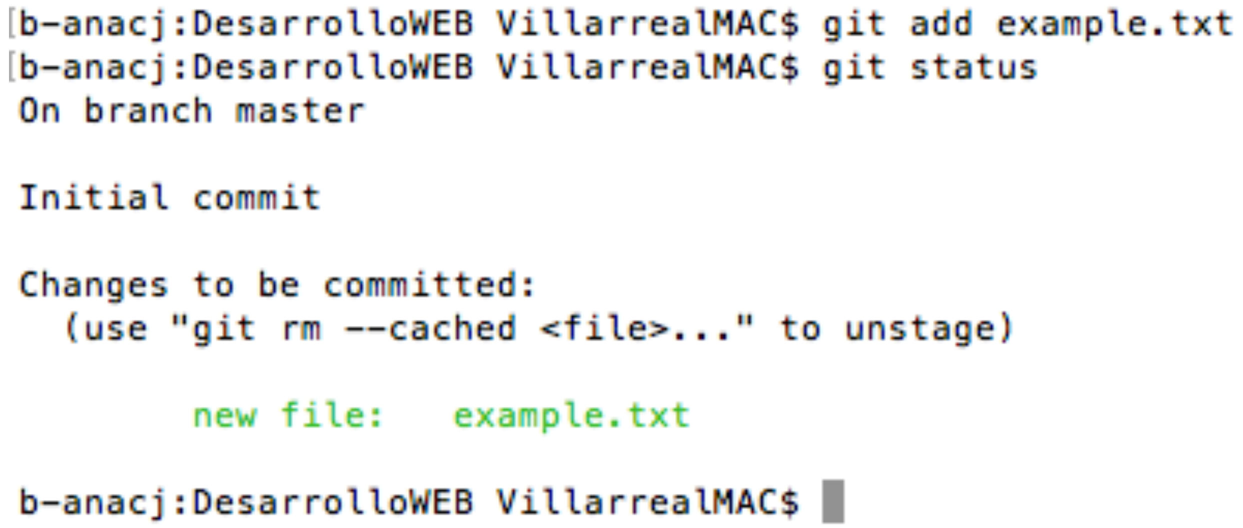


```
b-anacj:DesarrolloWEB VillarrealMAC$ ls
example.txt
b-anacj:DesarrolloWEB VillarrealMAC$ git status
On branch master

Initial commit

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

        example.txt

nothing added to commit but untracked files present (use "git add" to track)
b-anacj:DesarrolloWEB VillarrealMAC$
```

Adding Changes

Notice how Git says **example.txt** is "untracked"? That means Git sees that **example.txt** is a new file.

To tell Git to start tracking changes made to **example.txt** , we first need to add it to the staging area by using **git add.**



```
b-anacj:DesarrolloWEB VillarrealMAC$ git add example.txt
b-anacj:DesarrolloWEB VillarrealMAC$ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   example.txt

b-anacj:DesarrolloWEB VillarrealMAC$
```

Adding Changes

- **staged:**Files are ready to be committed.
- **unstaged:**Files with changes that have not been prepared to be committed.
- **untracked:**Files aren't tracked by Git yet. This usually indicates a newly created file.
- **deleted:**File has been deleted and is waiting to be removed from Git.

Adding Changes

- **add all:**You can also type git add -A . where the dot stands for the current directory, so everything in and beneath it is added. The -A ensures even file deletions are included.
- **git reset:**You can use git reset <filename> to remove a file or files from the staging area.

Committing

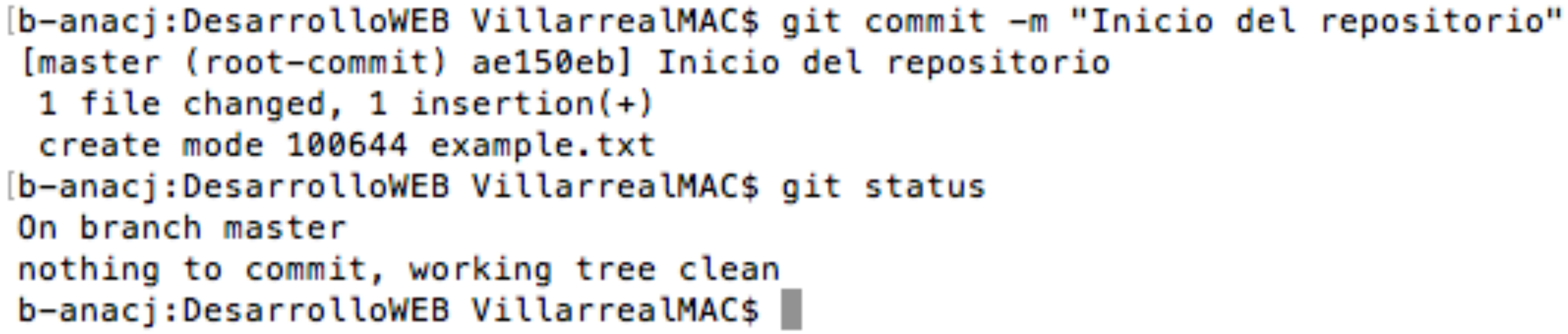
Notice how Git says changes to be committed? The files listed here are in the Staging Area, and they are not in our repository yet.

We could add or remove files from the stage before we store them in the repository.

To store our staged changes we run the commit command with a message describing what we've changed. Let's do that now by typing:

```
git commit –m “Esto es un mensaje”
```

Committing



```
[b-anacj:DesarrolloWEB VillarrealMAC$ git commit -m "Inicio del repositorio"
 [master (root-commit) ae150eb] Inicio del repositorio
  1 file changed, 1 insertion(+)
  create mode 100644 example.txt
[b-anacj:DesarrolloWEB VillarrealMAC$ git status
 On branch master
 nothing to commit, working tree clean
 b-anacj:DesarrolloWEB VillarrealMAC$ 
```

Committing

Staging Area:A place where we can group files together before we "commit" them to Git.

Commit A "commit" is a snapshot of our repository. This way if we ever need to look back at the changes we've made (or if someone else does), we will see a nice timeline of all changes.

Adding All Changes

You also can use wildcards if you want to add many files of the same type.

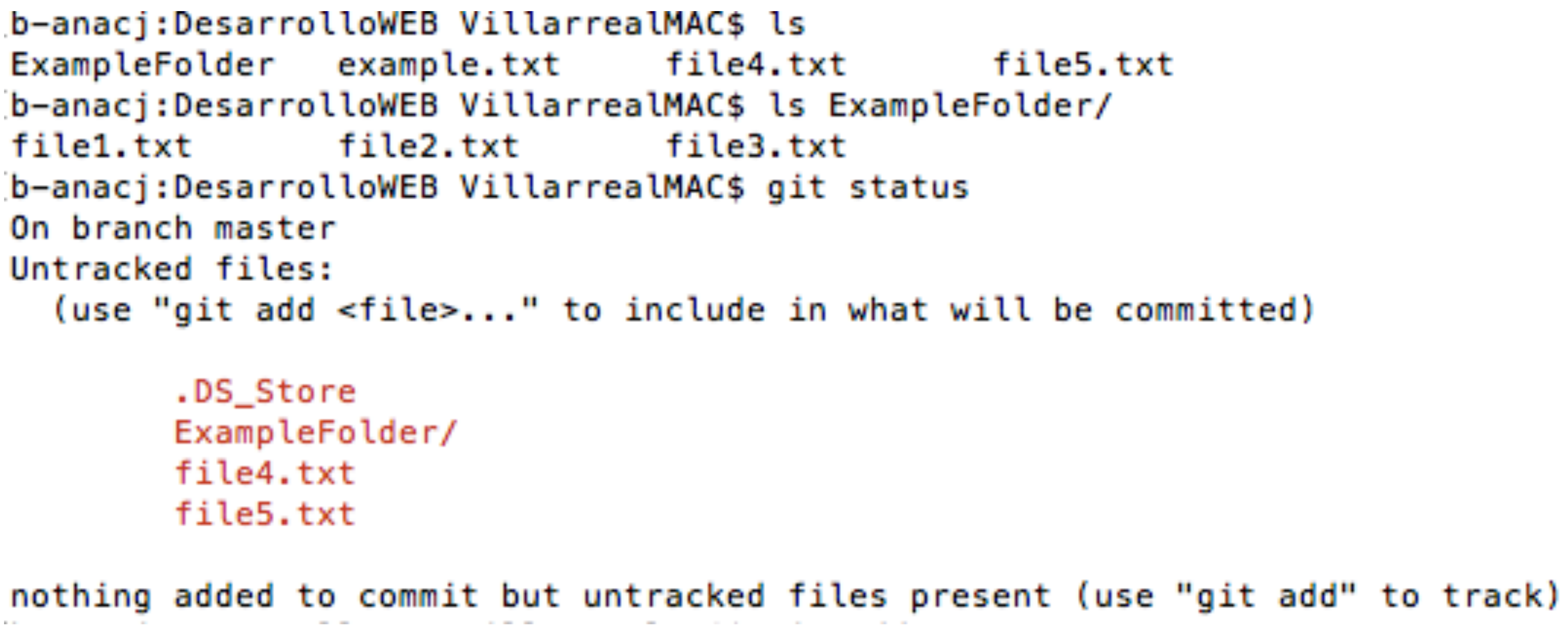
You should add a bunch of .txt files into your directory.

You should crear a directory named “ExampleFolder” and put some .txt into it.

we can add all the new files using a wildcard with git add. Don't forget the quotes!

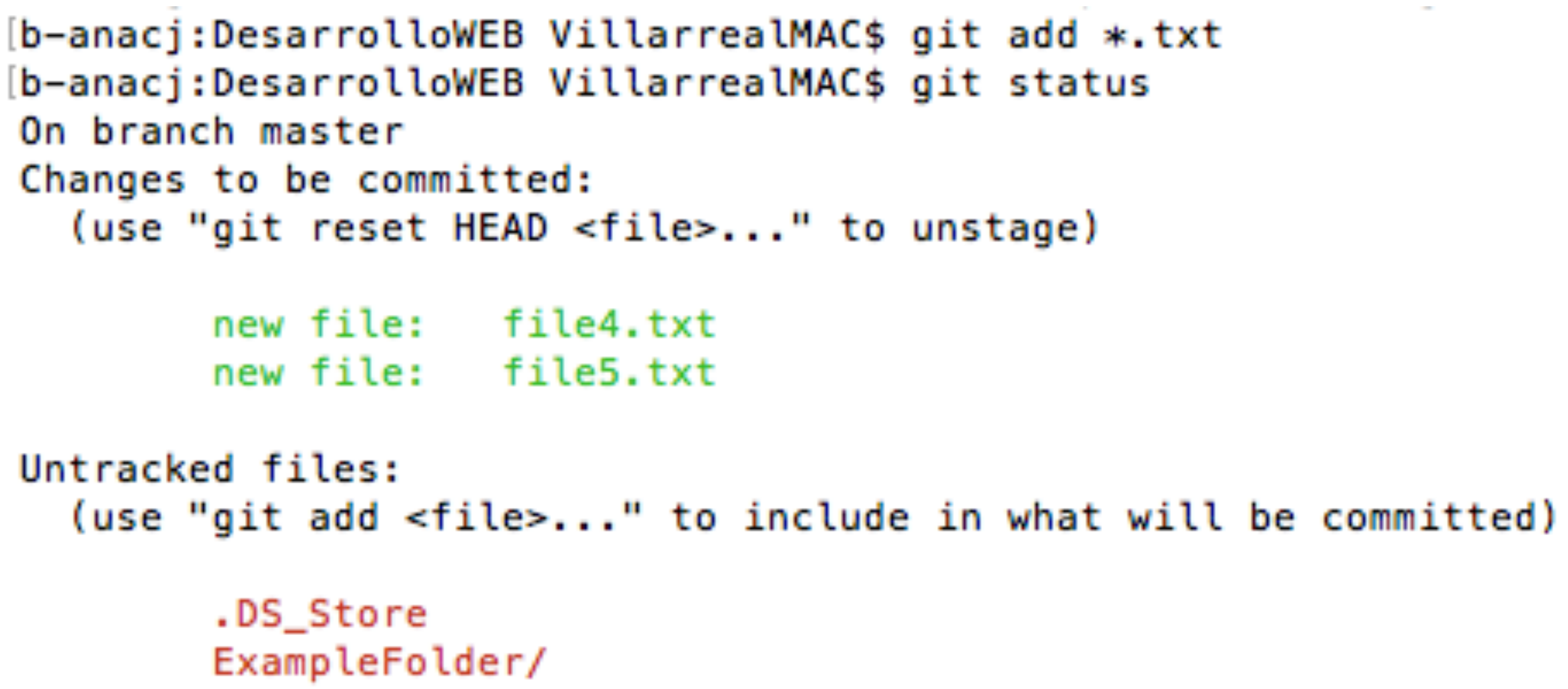
```
git add '*.txt'
```

Adding All Changes



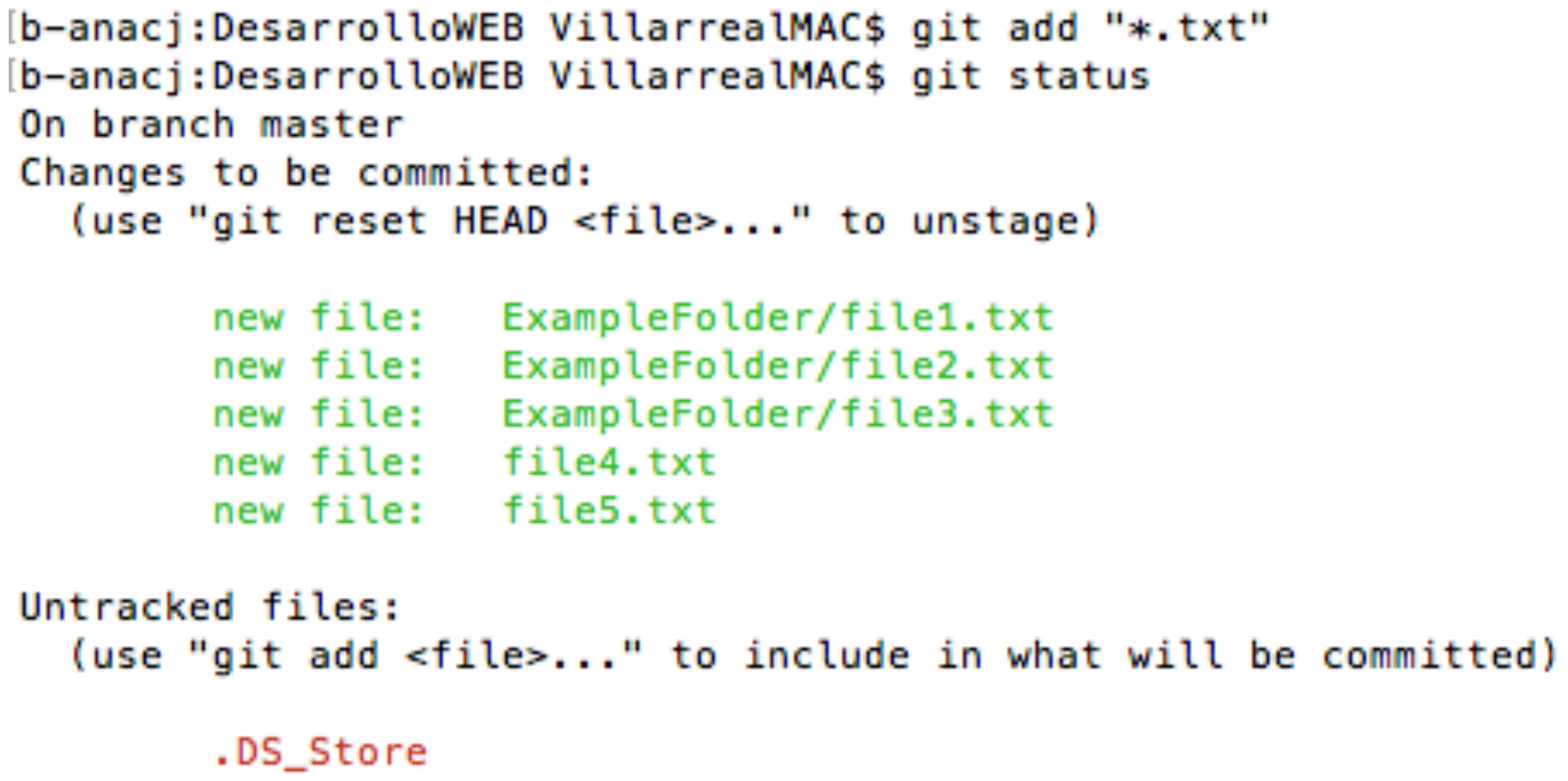
```
b-anacj:DesarrolloWEB VillarrealMAC$ ls
ExampleFolder    example.txt      file4.txt        file5.txt
b-anacj:DesarrolloWEB VillarrealMAC$ ls ExampleFolder/
file1.txt        file2.txt        file3.txt
b-anacj:DesarrolloWEB VillarrealMAC$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        .DS_Store
        ExampleFolder/
        file4.txt
        file5.txt

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

Adding All Changes



```
b-anacj:DesarrolloWEB VillarrealMAC$ git add *.txt
b-anacj:DesarrolloWEB VillarrealMAC$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   file4.txt
        new file:   file5.txt

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

        .DS_Store
        ExampleFolder/
```

Adding All Changes



```
b-anacj:DesarrolloWEB VillarrealMAC$ git add "*.txt"
b-anacj:DesarrolloWEB VillarrealMAC$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   ExampleFolder/file1.txt
        new file:   ExampleFolder/file2.txt
        new file:   ExampleFolder/file3.txt
        new file:   file4.txt
        new file:   file5.txt

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

        .DS_Store
```

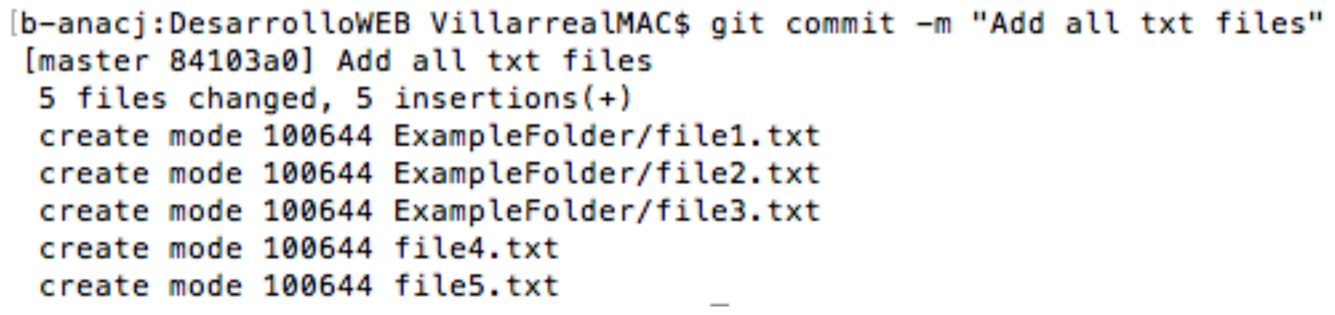
Committing All Changes

you've added all the text files to the staging area.

Feel free to run git status to see what you're about to commit.

If it looks good, go ahead and run:

git commit -m 'Add all txt files'



```
[b-anacj:DesarrolloWEB VillarrealMAC$ git commit -m "Add all txt files"
[master 84103a0] Add all txt files
 5 files changed, 5 insertions(+)
 create mode 100644 ExampleFolder/file1.txt
 create mode 100644 ExampleFolder/file2.txt
 create mode 100644 ExampleFolder/file3.txt
 create mode 100644 file4.txt
 create mode 100644 file5.txt
```

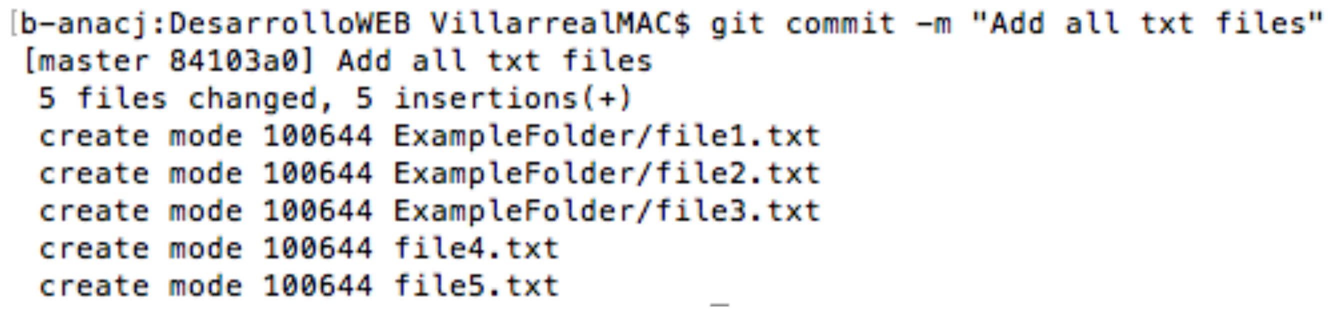
Committing All Changes

you've added all the text files to the staging area.

Feel free to run git status to see what you're about to commit.

If it looks good, go ahead and run:

git commit -m 'Add all txt files'



```
[b-anacj:DesarrolloWEB VillarrealMAC$ git commit -m "Add all txt files"
[master 84103a0] Add all txt files
 5 files changed, 5 insertions(+)
 create mode 100644 ExampleFolder/file1.txt
 create mode 100644 ExampleFolder/file2.txt
 create mode 100644 ExampleFolder/file3.txt
 create mode 100644 file4.txt
 create mode 100644 file5.txt
```

History

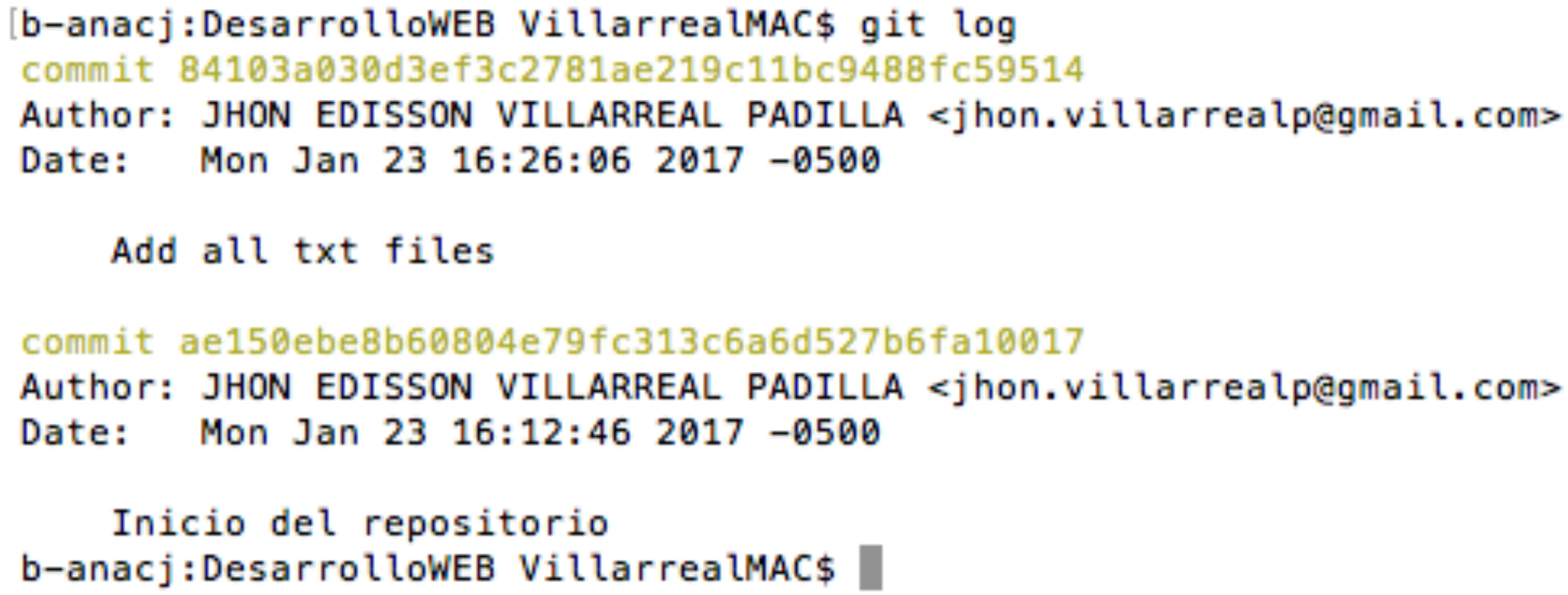
So we've made a few commits.

Now let's browse them to see what we changed.Fortunately for us, there's git log.

Think of Git's log as a journal that remembers all the changes we've committed so far, in the order we committed them. Try running it now:

git log

History



```
b-anacj:DesarrolloWEB VillarrealMAC$ git log
commit 84103a030d3ef3c2781ae219c11bc9488fc59514
Author: JHON EDISSON VILLARREAL PADILLA <jhon.villarrealp@gmail.com>
Date:   Mon Jan 23 16:26:06 2017 -0500

    Add all txt files

commit ae150ebe8b60804e79fc313c6a6d527b6fa10017
Author: JHON EDISSON VILLARREAL PADILLA <jhon.villarrealp@gmail.com>
Date:   Mon Jan 23 16:12:46 2017 -0500

    Inicio del repositorio
b-anacj:DesarrolloWEB VillarrealMAC$
```

History

More useful logs:
Use git log --summary to see more information for each commit.

You can see where new files were added for the first time or where files were deleted. It's a good overview of what's going on in the project.

Remote Repositories

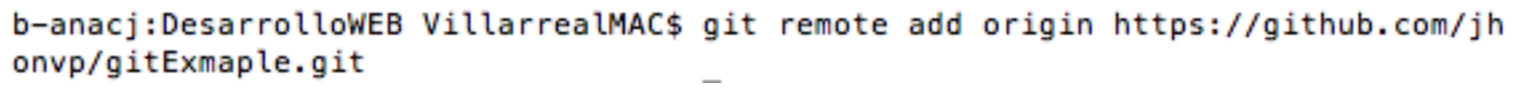
You should create a new empty GitHub repository for you.

To push our local repo to the GitHub server we'll need to add a remote repository.
This command takes a remote name and a repository URL, which in your case is

https://github.com/userName/gitExmaple.git

Go ahead and run git remote add with the options below:

git remote add origin https://github.com/jhonvp/gitExmaple.git



```
b-anacj:DesarrolloWEB VillarrealMAC$ git remote add origin https://github.com/jh
onvp/gitExmaple.git
```

Remote Repositories

Git remote:
Git doesn't care what you name your remotes, but it's typical to name your main one origin.It's also a good idea for your main repository to be on a remote server like GitHub in case your machine is lost at sea during a transatlantic boat cruise or crushed by three monkey statues during an earthquake.

Remote Repositories

Git remote:
Git doesn't care what you name your remotes, but it's typical to name your main one origin.It's also a good idea for your main repository to be on a remote server like GitHub in case your machine is lost at sea during a transatlantic boat cruise or crushed by three monkey statues during an earthquake.

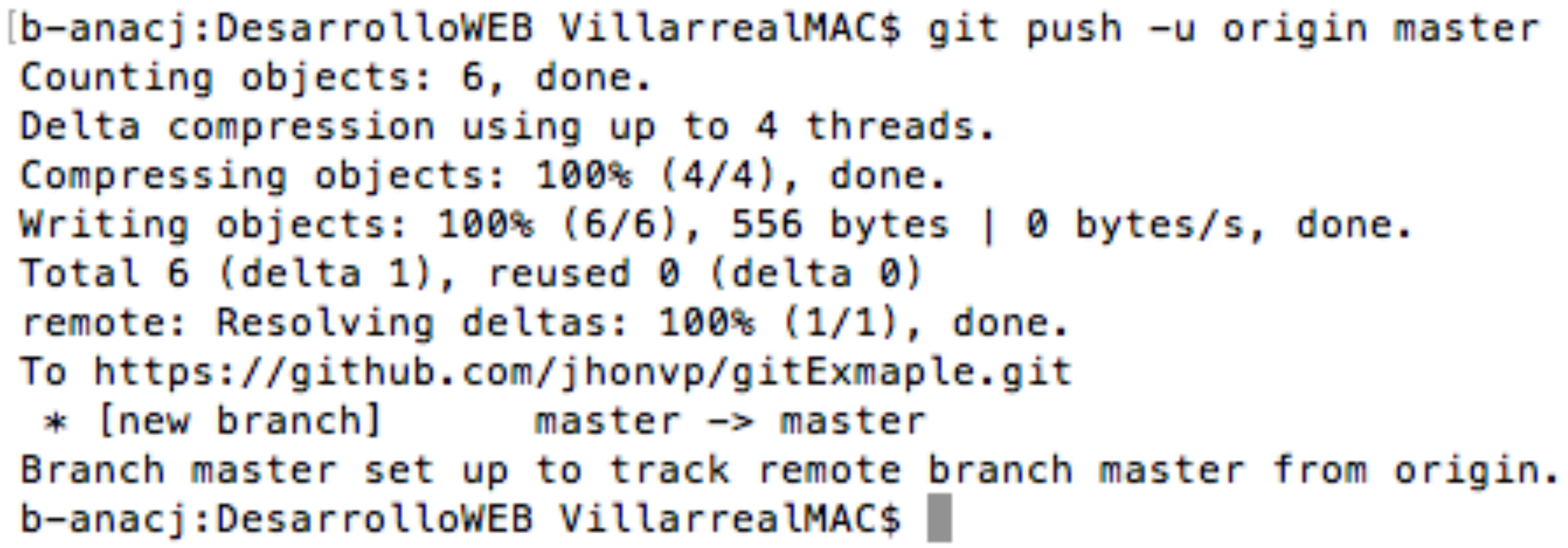
Pushing Remotely

The push command tells Git where to put our commits when we're ready, and boy we're ready. So let's push our local changes to our origin repo (on GitHub).

The name of our remote is origin and the default local branch name is master. The -u tells Git to remember the parameters, so that next time we can simply run git push and Git will know what to do. Go ahead and push it!

```
git push -u origin master
```

Pushing Remotely



```
[b-anacj:DesarrolloWEB VillarrealMAC$ git push -u origin master
Counting objects: 6, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (6/6), 556 bytes | 0 bytes/s, done.
Total 6 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), done.
To https://github.com/jhonvp/gitExmaple.git
 * [new branch]      master -> master
Branch master set up to track remote branch master from origin.
b-anacj:DesarrolloWEB VillarrealMAC$
```

Pushing Remotely

Cool Stuff:
When you start to get the hang of git you can do some really cool things with hooks when you push.For example, you can upload directly to a webserver whenever you push to your master remote instead of having to upload your site with an ftp client.

Check out Customizing Git - Git Hooks for more information.

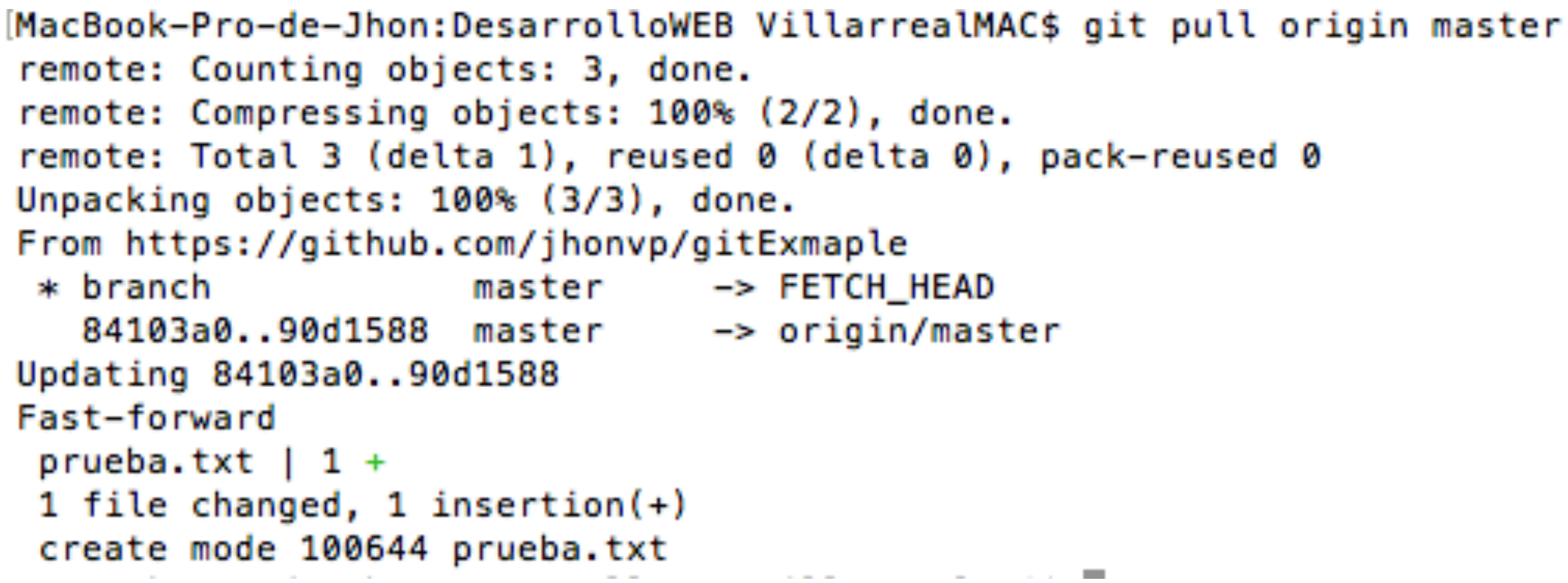
Pulling Remotely

Let's pretend some time has passed. We've invited other people to our GitHub project who have pulled your changes, made their own commits, and pushed them.

We can check for changes on our GitHub repository and pull down any new changes by running:

```
git pull origin master
```

Pulling Remotely



```
[MacBook-Pro-de-Jhon:DesarrolloWEB VillarrealMAC$ git pull origin master
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/jhonvp/gitExmaple
 * branch            master     -> FETCH_HEAD
   84103a0..90d1588  master     -> origin/master
Updating 84103a0..90d1588
Fast-forward
 prueba.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 prueba.txt
```

Pulling Remotely

git stash:

Sometimes when you go to pull you may have changes you don't want to commit just yet. One option you have, other than commiting, is to stash the changes.Use the command 'git stash' to stash your changes, and 'git stash apply' to re-apply your changes after your pull.

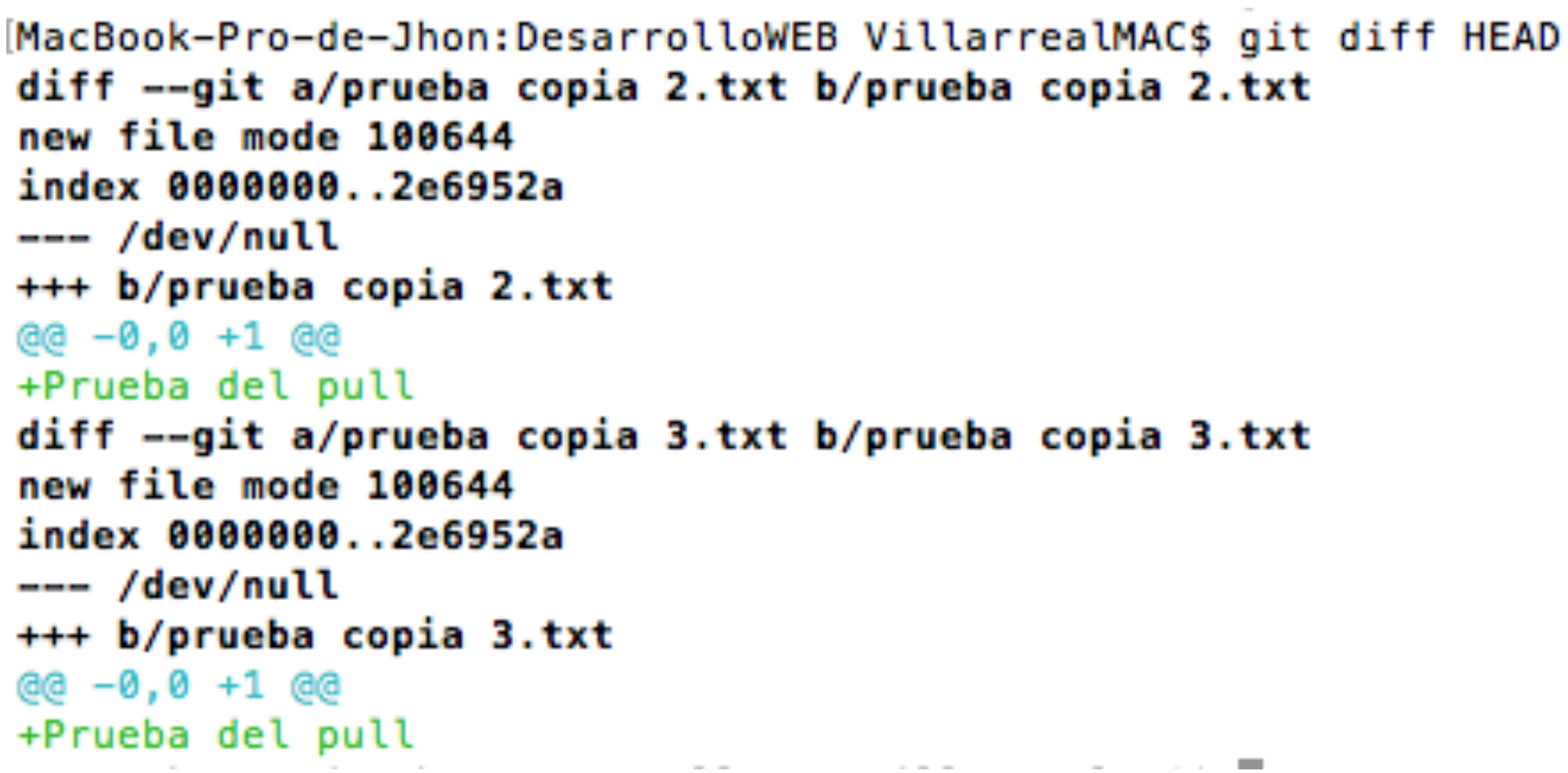
Differences

Let's take a look at what is different from our last commit by using the git diff command.

In this case we want the diff of our most recent commit, which we can refer to using the HEAD pointer.

```
git diff HEAD
```

Differences



```
MacBook-Pro-de-Jhon:DesarrolloWEB VillarrealMAC$ git diff HEAD
diff --git a/prueba copia 2.txt b/prueba copia 2.txt
new file mode 100644
index 0000000..2e6952a
--- /dev/null
+++ b/prueba copia 2.txt
@@ -0,0 +1 @@
+Prueba del pull
diff --git a/prueba copia 3.txt b/prueba copia 3.txt
new file mode 100644
index 0000000..2e6952a
--- /dev/null
+++ b/prueba copia 3.txt
@@ -0,0 +1 @@
+Prueba del pull
```

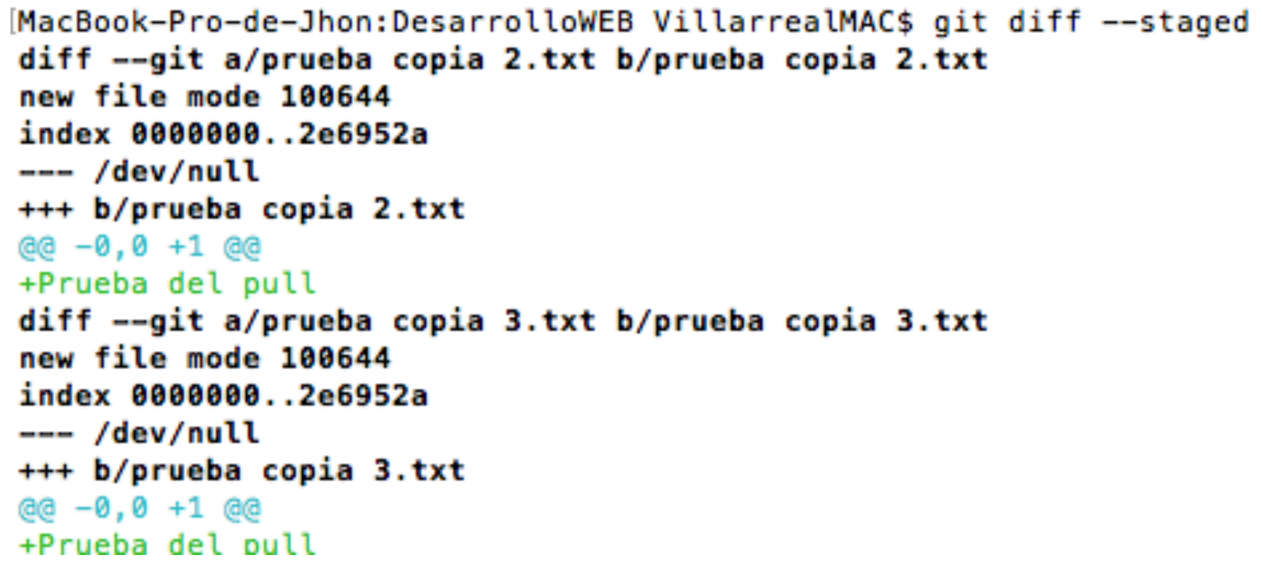
Differences

Commit Etiquette: You want to try to keep related changes together in separate commits. Using 'git diff' gives you a good overview of changes you have made and lets you add files or directories one at a time and commit them separately.

Differences

Good, now go ahead and run git diff with the --staged option to see the changes you just staged.

git diff --staged

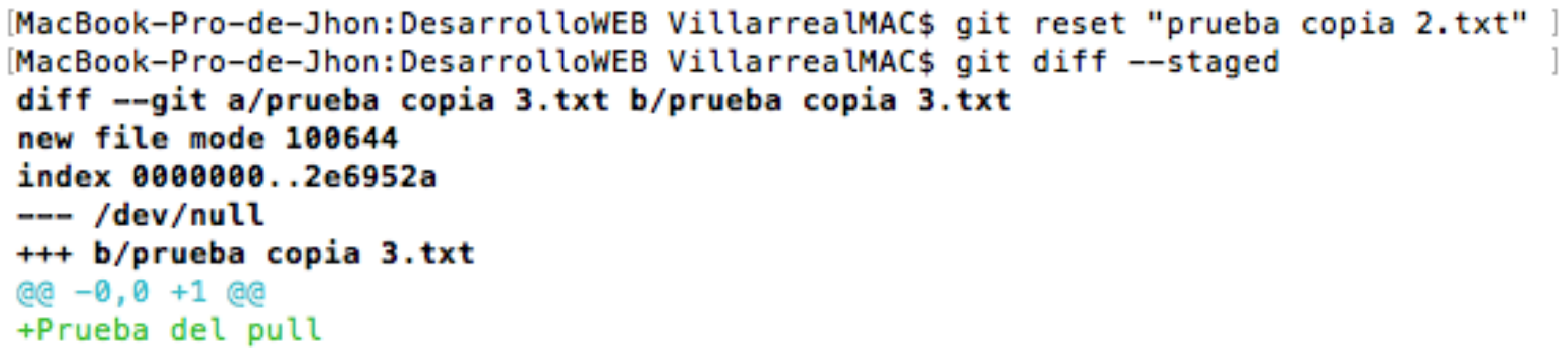


```
[MacBook-Pro-de-Jhon:DesarrolloWEB VillarrealMAC$ git diff --staged
diff --git a/prueba copia 2.txt b/prueba copia 2.txt
new file mode 100644
index 0000000..2e6952a
--- /dev/null
+++ b/prueba copia 2.txt
@@ -0,0 +1 @@
+Prueba del pull
diff --git a/prueba copia 3.txt b/prueba copia 3.txt
new file mode 100644
index 0000000..2e6952a
--- /dev/null
+++ b/prueba copia 3.txt
@@ -0,0 +1 @@
+Prueba del pull
```

Resetting the Stage

You can unstage files by using the git reset command.

git reset ‘prueba copia 2.txt’

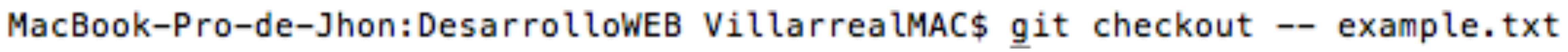


```
[MacBook-Pro-de-Jhon:DesarrolloWEB VillarrealMAC$ git reset "prueba copia 2.txt" ]
[MacBook-Pro-de-Jhon:DesarrolloWEB VillarrealMAC$ git diff --staged              ]
diff --git a/prueba copia 3.txt b/prueba copia 3.txt
new file mode 100644
index 0000000..2e6952a
--- /dev/null
+++ b/prueba copia 3.txt
@@ -0,0 +1 @@
+Prueba del pull
```

Undo

Files can be changed back to how they were at the last commit by using the command: git checkout -- <target>. Go ahead and get rid of all the changes since the last commit

git checkout -- example.txt



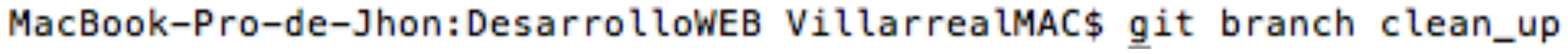
```
MacBook-Pro-de-Jhon:DesarrolloWEB VillarrealMAC$ git checkout -- example.txt
```

Branching Out

When developers are working on a feature or bug they'll often create a copy (aka. branch) of their code they can make separate commits to. Then when they're done they can merge this branch back into their main master branch.

Let's create a branch called clean_up, where we'll do all the work:

git branch clean_up

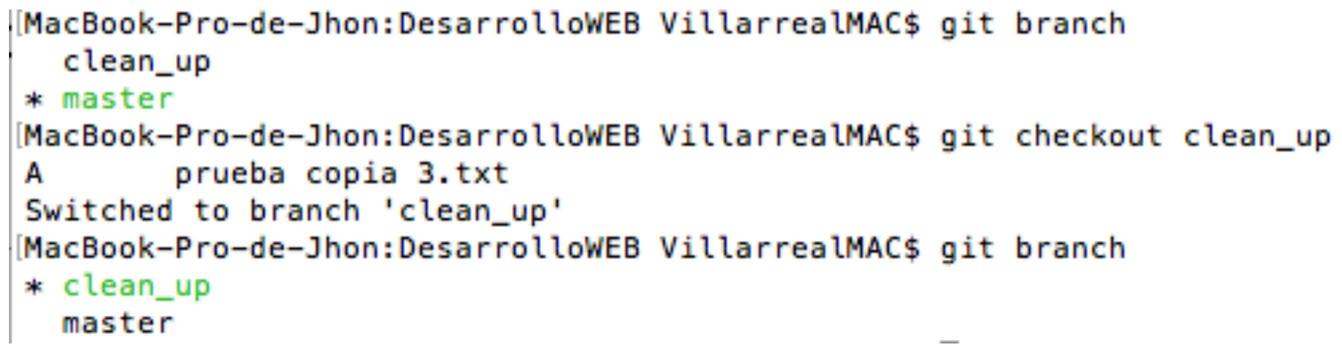


```
MacBook-Pro-de-Jhon:DesarrolloWEB VillarrealMAC$ git branch clean_up
```

Switching Branches

Great! Now if you type git branch you'll see two local branches: a main branch named master and your new branch named clean_up.
You can switch branches using the git checkout <branch> command. Try it now to switch to the clean_up branch:

git checkout clean_ip



```
MacBook-Pro-de-Jhon:DesarrolloWEB VillarrealMAC$ git branch
  clean_up
* master
MacBook-Pro-de-Jhon:DesarrolloWEB VillarrealMAC$ git checkout clean_up
A       prueba copia 3.txt
Switched to branch 'clean_up'
MacBook-Pro-de-Jhon:DesarrolloWEB VillarrealMAC$ git branch
* clean_up
  master
```

Switching Branches

All at Once

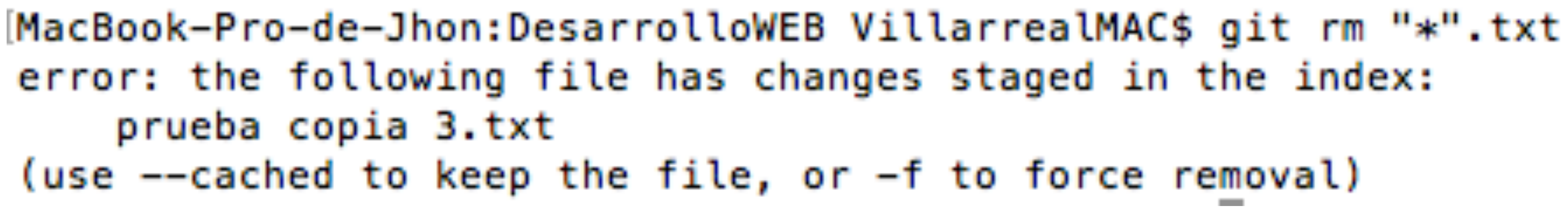
You can use: git checkout -b new_branch to checkout and create a branch at the same time. This is the same thing as doing:
git branch new_branch
git checkout new_branch

Removing All The Things

Ok, so you're in the clean_up branch. You can finally remove all those files by using the git rm command which will not only remove the actual files from disk, but will also stage the removal of the files for us.

Lets' sweep, go ahead and run:

git rm *.txt



```
[MacBook-Pro-de-Jhon:DesarrolloWEB VillarrealMAC$ git rm "*".txt
error: the following file has changes staged in the index:
    prueba copia 3.txt
(use --cached to keep the file, or -f to force removal)
```

Removing All The Things

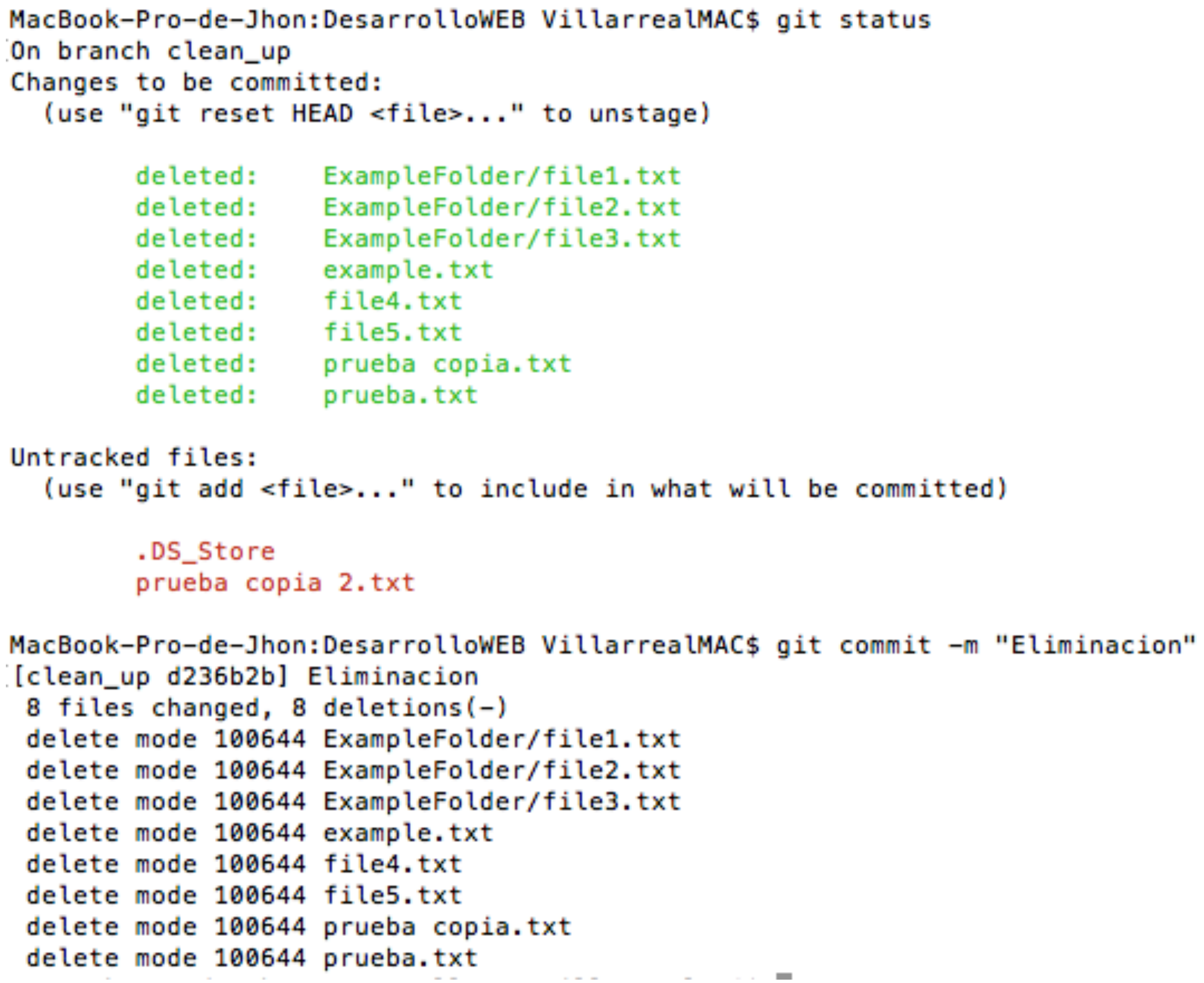
Remove all the things!

Removing one file is great and all, but what if you want to remove an entire folder? You can use the recursive option on git rm:

```
git rm -r folder
```

will recursively remove all folders and files from the given directory.

Commiting Branch Changes

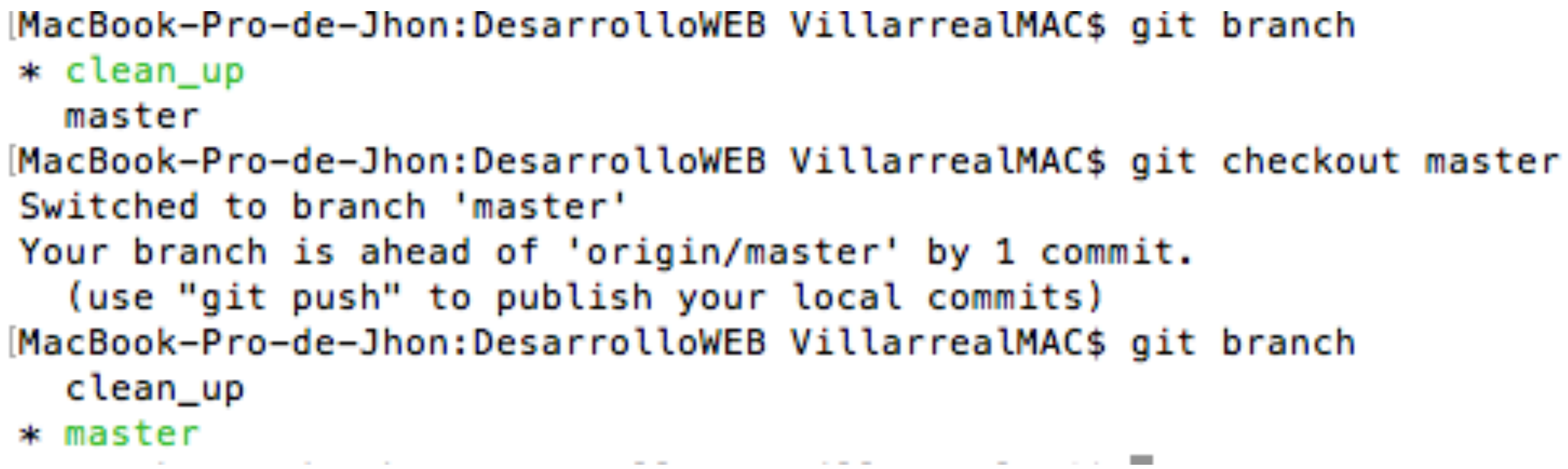


```
MacBook-Pro-de-Jhon:DesarrolloWEB VillarrealMAC$ git status
On branch clean_up
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        deleted:    ExampleFolder/file1.txt
        deleted:    ExampleFolder/file2.txt
        deleted:    ExampleFolder/file3.txt
        deleted:    example.txt
        deleted:    file4.txt
        deleted:    file5.txt
        deleted:    prueba copia.txt
        deleted:    prueba.txt

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

        .DS_Store
        prueba copia 2.txt

MacBook-Pro-de-Jhon:DesarrolloWEB VillarrealMAC$ git commit -m "Eliminacion"
[clean_up d236b2b] Eliminacion
 8 files changed, 8 deletions(-)
 delete mode 100644 ExampleFolder/file1.txt
 delete mode 100644 ExampleFolder/file2.txt
 delete mode 100644 ExampleFolder/file3.txt
 delete mode 100644 example.txt
 delete mode 100644 file4.txt
 delete mode 100644 file5.txt
 delete mode 100644 prueba copia.txt
 delete mode 100644 prueba.txt
```

Switching Back to master

Go ahead and checkout the master branch:

git checkout master



```
[MacBook-Pro-de-Jhon:DesarrolloWEB VillarrealMAC$ git branch
* clean_up
  master
[MacBook-Pro-de-Jhon:DesarrolloWEB VillarrealMAC$ git checkout master
Switched to branch 'master'
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)
[MacBook-Pro-de-Jhon:DesarrolloWEB VillarrealMAC$ git branch
  clean_up
* master
```

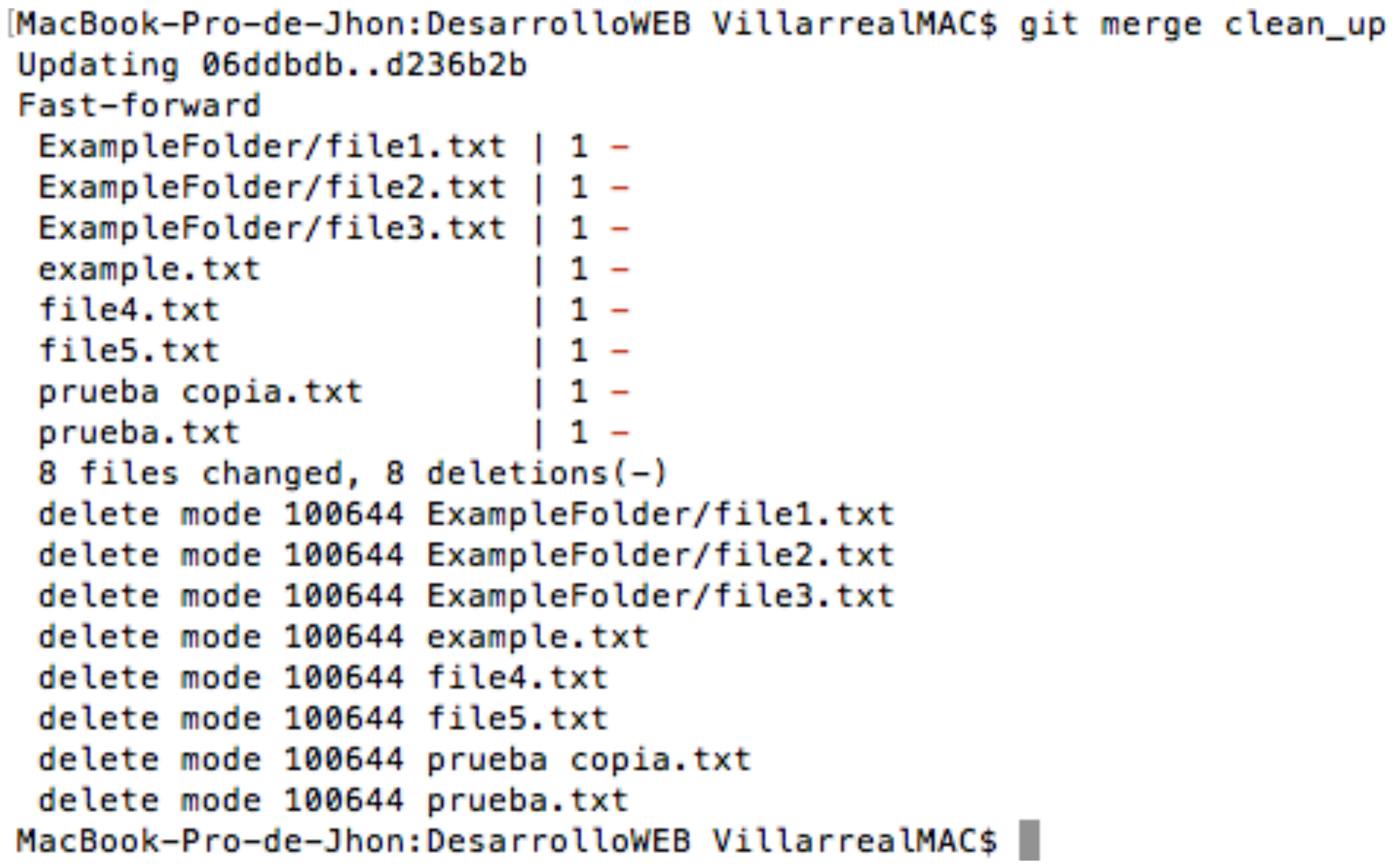
Preparing to Merge

Alrighty, the moment has come when you have to merge your changes from the clean_up branch into the master branch. Take a deep breath, it's not that scary.

We're already on the master branch, so we just need to tell Git to merge the clean_up branch into it:

```
git merge clean_up
```

Preparing to Merge



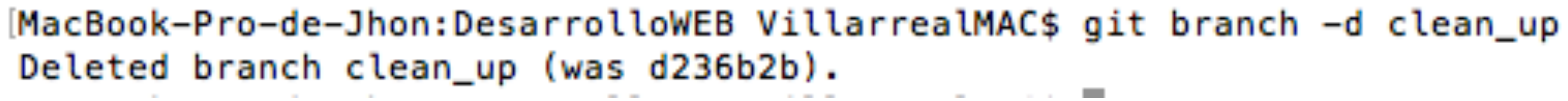
```
MacBook-Pro-de-Jhon:DesarrolloWEB VillarrealMAC$ git merge clean_up
Updating 06ddbdb..d236b2b
Fast-forward
 ExampleFolder/file1.txt | 1 -
 ExampleFolder/file2.txt | 1 -
 ExampleFolder/file3.txt | 1 -
 example.txt             | 1 -
 file4.txt               | 1 -
 file5.txt               | 1 -
 prueba copia.txt        | 1 -
 prueba.txt              | 1 -
 8 files changed, 8 deletions(-)
 delete mode 100644 ExampleFolder/file1.txt
 delete mode 100644 ExampleFolder/file2.txt
 delete mode 100644 ExampleFolder/file3.txt
 delete mode 100644 example.txt
 delete mode 100644 file4.txt
 delete mode 100644 file5.txt
 delete mode 100644 prueba copia.txt
 delete mode 100644 prueba.txt
MacBook-Pro-de-Jhon:DesarrolloWEB VillarrealMAC$ 
```

Keeping Things Clean

All that's left to do is clean up after yourself. Since you're done with the clean_up branch you don't need it anymore.

You can use git branch -d <branch name> to delete a branch. Go ahead and delete the clean_up branch now:

git branch -d clean_up



```
[MacBook-Pro-de-Jhon:DesarrolloWEB VillarrealMAC$ git branch -d clean_up
 Deleted branch clean_up (was d236b2b).
```

Resource

https://try.github.io/levels/1/challenges/1