Programming fundamentals with Python

Pepe García

2020-04-20

Programming fundamentals with Python

Plan for today

Learn about version control systems

Introduction to Git

Learn a bit about CLI tools

Command line

The command line allows users navigating the computer and managing it in the same way a **Graphical User Interface** can.

```
$ ls -l

total 0
drwx------+  5 pepe  staff   160 Nov 11 02:22 Desktop
drwx------+  3 pepe  staff    96 Nov 10 21:07 Documents
drwx------+ 12 pepe  staff   384 Nov 11 16:32 Downloads
drwx------@ 66 pepe  staff  2112 Nov 11 15:54 Library
drwx------+  4 pepe  staff   128 Nov 11 09:18 Movies
drwx------+  3 pepe  staff    96 Nov 11 02:22 Music
drwx------+  4 pepe  staff   128 Nov 10 23:50 Pictures
```

Command line

We can **list files** in a folder using the **ls** command (**dir** command on Windows)

```
$ ls
```

```
Desktop  Documents Downloads Library  Movies   Music    Pictures Public   opt
```

Command line

We change directories using **cd**.

```
$ ls
```

```
Desktop  Documents Downloads Library  Movies   Music    Pictures Public   opt
```

```
$ cd Desktop
```

Command line

We can go to *upper* directories using **cd ..**

```
$ ls

Desktop  Documents Downloads Library  Movies   Music    Pictures Public   opt

$ cd Desktop

$ cd ..

$ ls

Desktop  Documents Downloads Library  Movies   Music    Pictures Public   opt
```

Command line

We can see where we are with the **pwd** command (**echo %cd%** in Windows... ¯_( )_/¯)

```
$ pwd
/Users/pepe

$ cd Desktop

$ pwd
/Users/pepe/Desktop

$ cd ..

$ pwd
/Users/pepe
```

Version control

Version control is the process of handling programs, versions, changes, and differences

Version control

Who made changes

To which files

When did they do it

Why did they do it

Git is a tool to help us handling code changes, versions, reverts, etc.
Git was created by Linus Torvalds to handle all contributions in the Linux Kernel in a sane way

Using Git

In order to use git, we need a git client. There are several of them, from command line clients to Graphical User Interface clients.

Installing Git

If you don't have it installed, you can get it from **https://git-scm.com/downloads**

Git concepts

Git terminology can be very broad, but we'll focus on the parts that matter

Git concepts

Working directory

The **working directory** is the folder in which our code will be. The contents of this folder will be controlled by **git**.

Staging area

Whenever we're happy about the state of a file, we move it to the **staging area**. In the **staging area** we save files that are ready to be saved.

Local repository

The **local repository** is the place in which we store all the changes made to all the files of our projects, over time.

Creating our first repository

Create a folder called **my-first-repo** in your desktop

Navigate to it using the terminal (**cd**)

Open **spyder**, create a python file and save it in **my-first-repo** folder

In the terminal, initialize the repository with **git init**

Git operations

We already have a file in a repository

(**my-first-repo**).

We can always see the status of our repo:

```
$ git status

On branch master

No commits yet

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

    file.py
```

Git operations

We can use **git add file.py** to add the file to the staging area, in which we store the files ready to be commited.

```
$ git add file.py

$ git status

On branch master

No commits yet

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

    new file:   file.py
```

Git operations

When there is a meaningful change we want to save, we use **git commit** to save it to our local repository.

We use **git commit -m "message"** and try to use a meaningful description of the changes we just made.

```
$ git commit -m "add file.py to git"
[master (root-commit) 123cd8b] add file.py to git
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 file.py
```

Git concepts

Git operations

One of the most powerful features of **git** is handling changes. Let's do some changes to our **file.py**.

```
def func(a, b):
    return a + b
```

Git operations

And let's see the changes now! **git diff**

Git will show the lines we added with a + **sign** before, and those we removed with a - **sign**

```
$ git diff

diff --git a/file.py b/file.py
index e69de29..c09bd0e 100644
--- a/file.py
+++ b/file.py
@@ -0,0 +1,3 @@
+
+def func(a, b):
+    return a + b
```

Commit the last changes

Git operations

Other of the cool features of **git** is watching the history of our repository. We can see all the changes to it!

```
$ git log

commit 123cd8b45ae31065cdd7cf0ecd8ce83b444886db (HEAD -> master, origin/mas
Author: Pepe García <pepe@pepegar.com>
Date:   Mon Nov 11 23:55:49 2019 +0100

    add file.py to git
```

Break

Github

Github is a code hosting service. We can host our coding projects there.

https://github.com

Github

Create an account at Github.

https://github.com

Creating a remote repository in Github

Creating a remote repository in Github

Creating a remote repository in Github

We'll use our repo URL now, to set up a remote repository in our local repository.

Setting up a remote repository

```
$ git remote add origin https://github.com/popogor/my-first-repo.git
```

Setting up a remote repository

We just created this, let's **push** some code!

Pushing

```
$ git push origin master

Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 608 bytes | 608.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/popogor/my-first-repo.git
 * [new branch]      master -> master
```

git push sends changes from the local repository to the remote one. It's the way we have to *upload* code to github.

Github

pushing our code to Github

Pulling

```
$ git pull origin master
```

git pull is the opposite of **git push**. It brings changes from the remote repository to the local one.

Cloning a project

```
$ git clone https://github.com/octocat/Spoon-Knife.git

Cloning into 'Spoon-Knife'...
remote: Enumerating objects: 16, done.
remote: Total 16 (delta 0), reused 0 (delta 0), pack-reused 16
Unpacking objects: 100% (16/16), done.
```

we use **git clone** to copy a repository to our local computer.

Solving conflicts

Conflicts occur naturally when coding. Mostly when we do collaborate with others.

Let's introduce a conflict and fix it ourselves!

Solving conflicts

First, in our local copy of **my-first-repo**, let's change the function we had to:

```
def func(a, b):
    return a - b
```

A simple change, just modify it so it substracts instead of adding.

Solving conflicts

And then **git add** and **git commit** it.

```
$ git add file.py

$ git commit -m "change function and make it substract"

[master ae46fc3] change function and make it substract
 1 file changed, 1 insertion(+), 1 deletion(-)
```

Solving conflicts

Now let's simulate the changes someone else would make in github.

Solving conflicts

Then, in our local repository, let's **git pull**

```
$ git pull origin master


remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/popogor/my-first-repo
 * branch            master     -> FETCH_HEAD
   4c659b6..e441a78  master     -> origin/master
Auto-merging file.py
CONFLICT (content): Merge conflict in file.py
Automatic merge failed; fix conflicts and then commit the result.
```

There's lots of output there, the important bit is the **CONFLICT** line

Solving conflicts

First of all, we need to see the conflicts

```
$ git diff

diff --cc file.py
index 64ad20f,84e4d51..0000000
--- a/file.py
+++ b/file.py
@@@ -1,3 -1,3 +1,7 @@

  def func(a, b):
++<<<<<<< HEAD
 +    return a - b
++=======
+     return a * b
++>>>>>>> e441a78ff5f91b986f0da3afddbb7a7a01ee1859
```

Solving conflicts

To do so, we need to edit the file and select the part we prefer, **deleting the rest**. Let's say we prefer the multiplication.

We finish the process by doing **git add** and **git commit** after solving the conflict, telling git we're happy with the result.

```
$ git add file.py

$ git commit -m "merged conflict in file.py"
[master 1abfd41] merged conflict in file.py
```

Solving conflicts

```
$ git log

commit 1abfd4151a6d44e3268c59b56065730676e545db (HEAD -> master)
Merge: ae46fc3 e441a78
Author: Pepe García <pepe@pepegar.com>
Date:   Tue Nov 12 01:55:00 2019 +0100

    merged conflict in file.py

commit e441a78ff5f91b986f0da3afddbb7a7a01ee1859 (origin/master)
Author: popogor <46658846+popogor@users.noreply.github.com>
Date:   Tue Nov 12 01:38:13 2019 +0100

    change function and make it multiply

commit ae46fc37dbbf344f3ee4e5d189bb0714a543dd0f
```

Bibliography

Images and inspiration drawn from

https://rachelcarmena.github.io/2018/12/12/how-to-teach-git.html

https://dev.to/unseenwizzard/learn-git-concepts-not-commands-4gjc