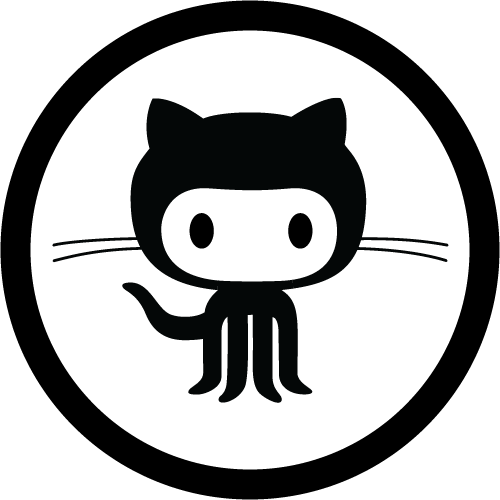
**GithubConfiguration Guideline**

****

Document Version No. 1.0

Created by: Ariel Duarte

© Copyright Ariel Duarte2013.

Table of Contents

[1 • Introduction 3](#_Toc328101372)

[2 • Download and Install GitHub for Windows 4](#_Toc328101373)

[3 • Create user account 8](#_Toc328101376)

[4 • Create [SSH keys](https://help.github.com/articles/generating-ssh-keys) 9](#_Toc328101379)

[5 • Account settings 11](#_Toc328101379)

[6 • Clone master repository 14](#_Toc328101379)

[7 • Git Commands](#_Toc328101379) 15

***Change Control***

| **Date** | **Author** | **Version** | **Change Reference** |
| --- | --- | --- | --- |
|  |  |  |  |
| 24/09/2012 | Ariel Duarte | 1.0 |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

***Sign-off***

| **Position** | **Name** | **RevisionComment** | **Signature** | **Date** |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

***Reviews***

| **Name** | **Position** | **Date** |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# Introduction

GitHub is a web-based hosting service for software development projects that uses the Git revision control system.

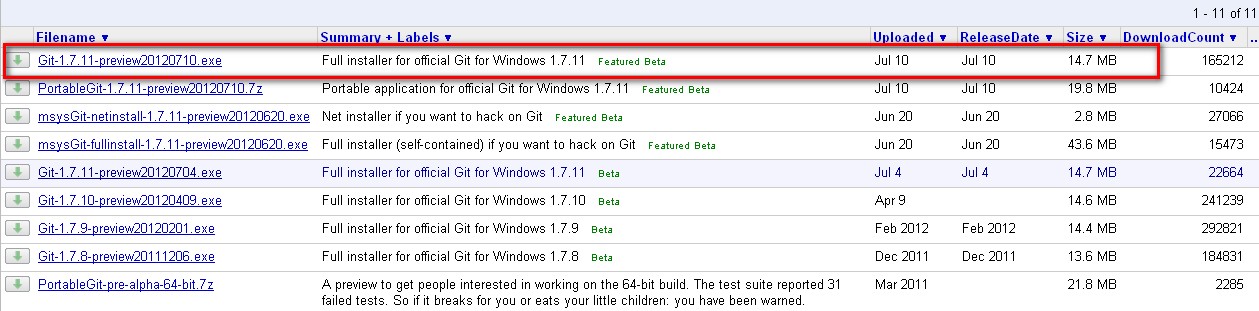
Git is a version control system, likes SVN or CVS, with the difference thatthis is a distributed version control system.

*This document intended as a quick guide for the installation and configuration of GitHub on Windows.*

# Download and Install GitHub for Windows

Download the **msysgitGit for Windows** installer from :

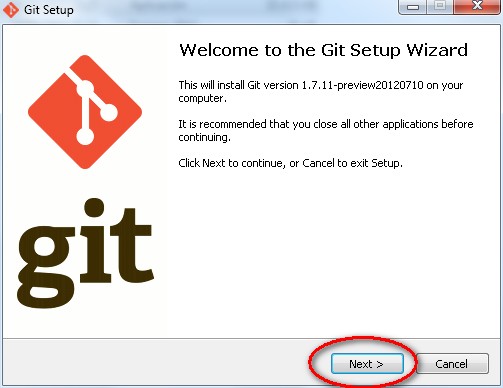
[**http://code.google.com/p/msysgit/downloads/list**](http://code.google.com/p/msysgit/downloads/list)

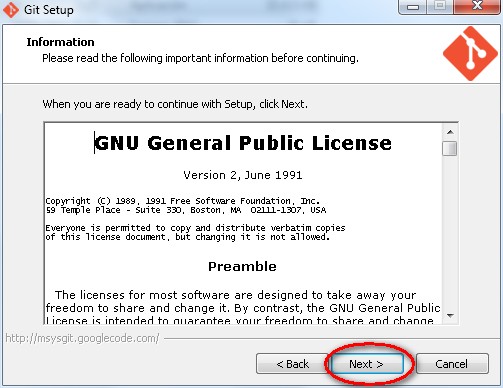


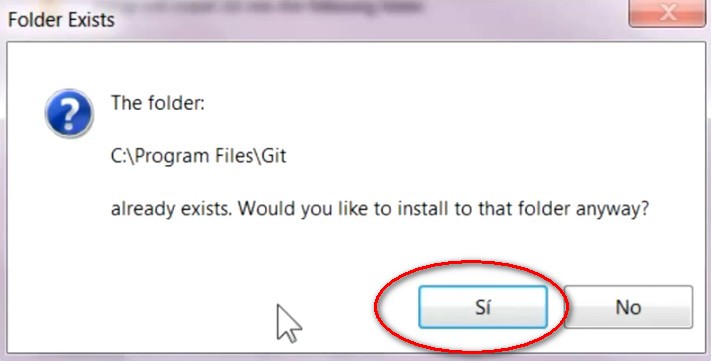
Note: You’ll need administrative privileges.

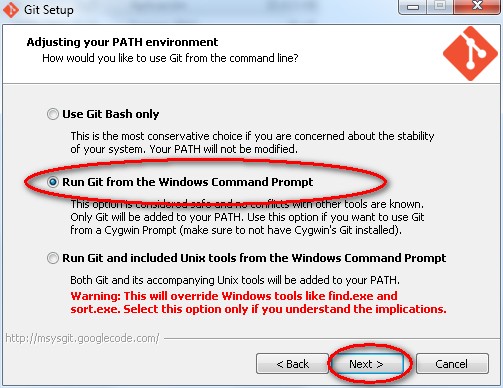
**That is a basic installer.**

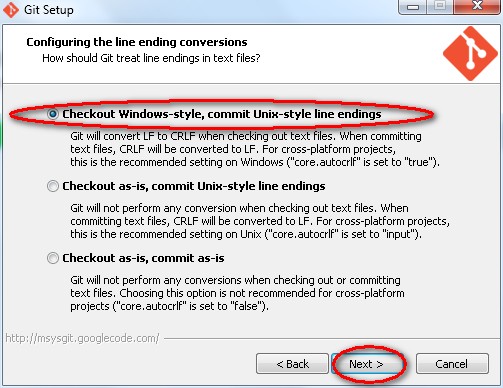
Run theGit-1.7.11-preview20120620.exe.

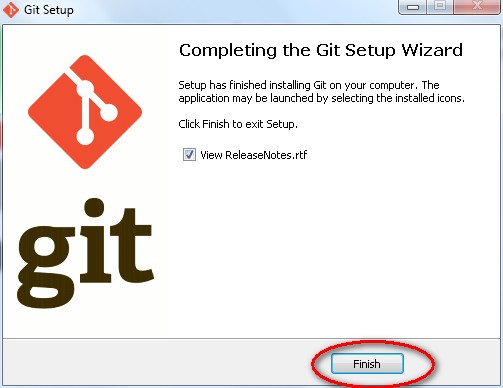












# Create user account

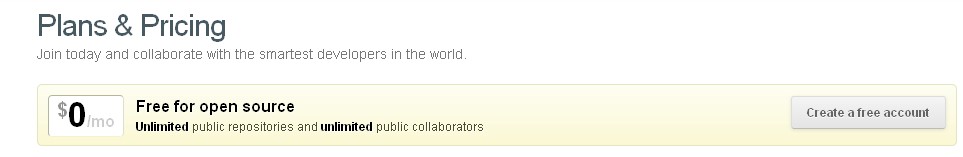
To create a user account GitHub, we must go to the following URL:

[**http://github.com/plans**](http://github.com/plans)

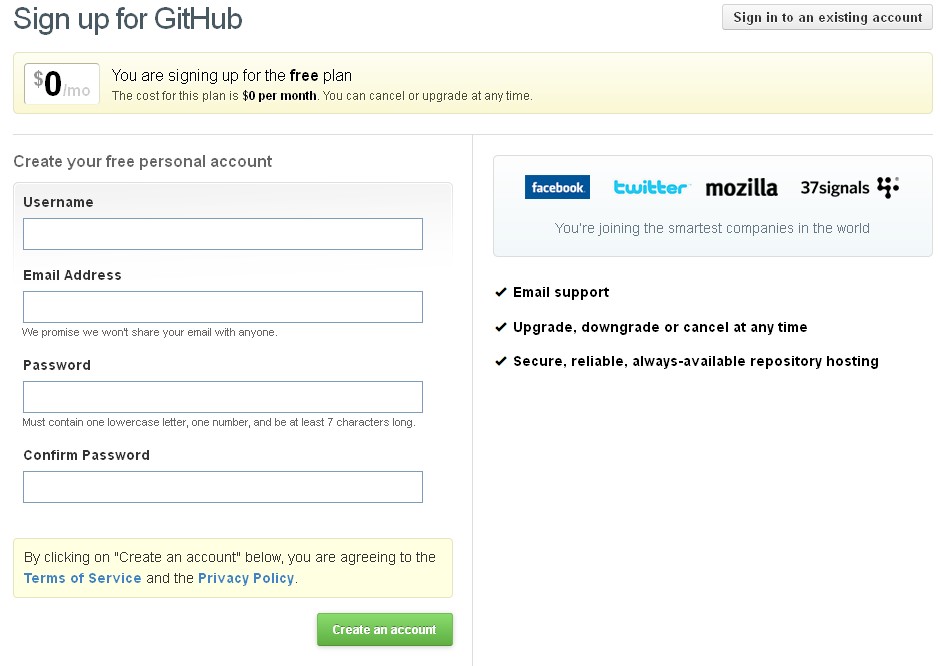
Click on the button (Plans, Pricing and Signup)



And in Plans & Pricing we have to click on the button **“Create a free account”**



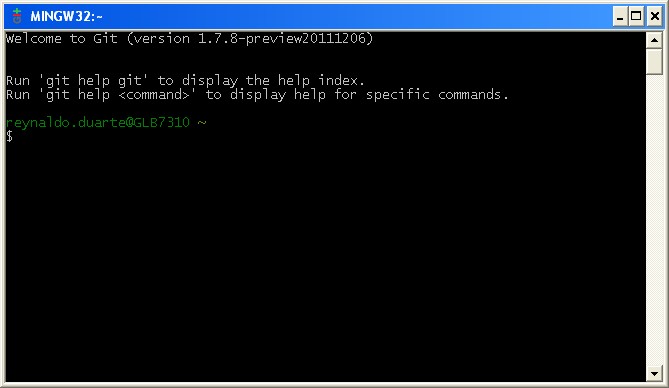
We have to fill the required fields and press the button **“Create an account”**



# Create [SSH keys](https://help.github.com/articles/generating-ssh-keys)

To open up a terminal Git we have to click on the file “Fit Bash.exe” in the following path directory of Windows:

**C:\Program Files\Git\Git Bash.exe**



**Note : you can use this Git commands both Unix and Windows**

\*Check if there is an SSH key on your computer

**>cd ~/.ssh**

If we already have one key created we must create a backup

**>ls**

Create a directory for the backup key

**>mkdirkey\_backup**

Create a backup of the key

**>cpid\_rsa\* key\_backup**

\*In case of having no key we can create it the following way:

**>ssh-keygen -t rsa -C** [**xxx@gmail.com**](mailto:xxx@gmail.com) (**email user account**)

1º -->It will ask you the directory where you want to save the key (**press enter**)

2º -->It will require you to enter the user password (**password user account**)

3º -->Key is generated: The key fingerprint is: **4a:cf:88:54:47:15:29:e7:ec:7e:7b:61:84:8d:19:c1xxx@gmail.com**

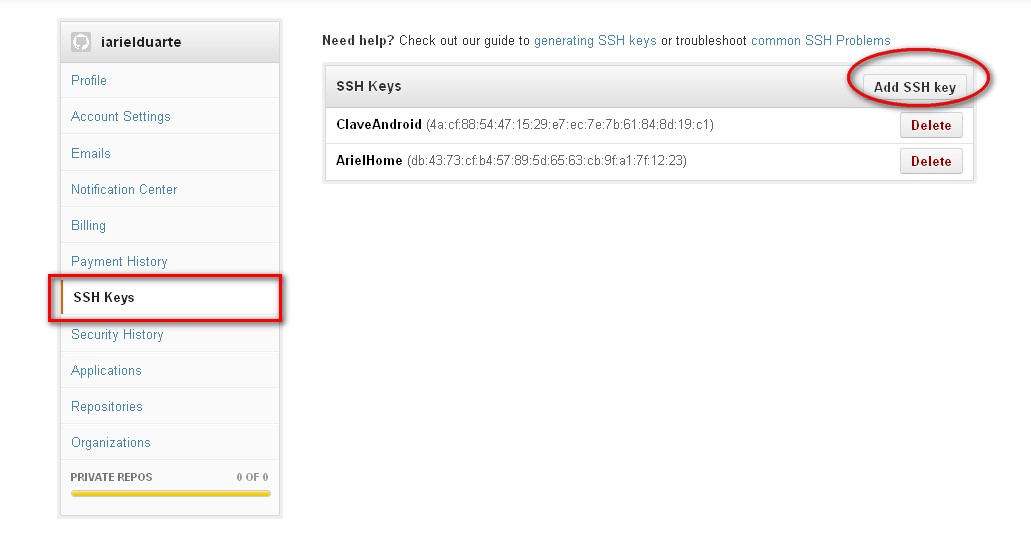
# Account settings

1. First of all, we have to login it the following URL

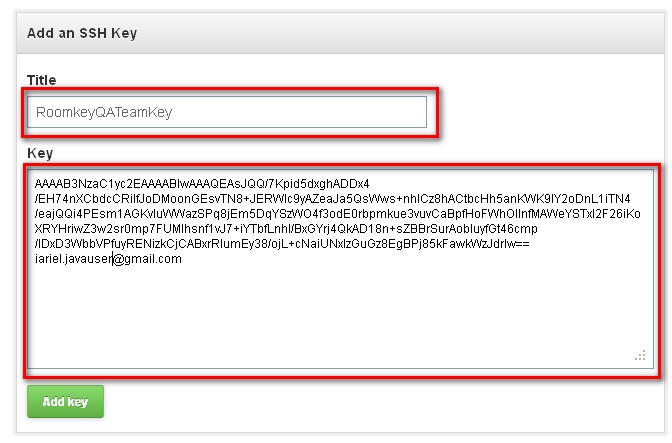
[**https://github.com/login**](https://github.com/login)

Select the "Account Settings" icon



Select the option"SSH Keys"and pressthe "AddSSHkey"

On the next screen you will enter a **“title”** for the key and the **“key ssh-rsa”** that you wantto generate.



keyssh-rsa

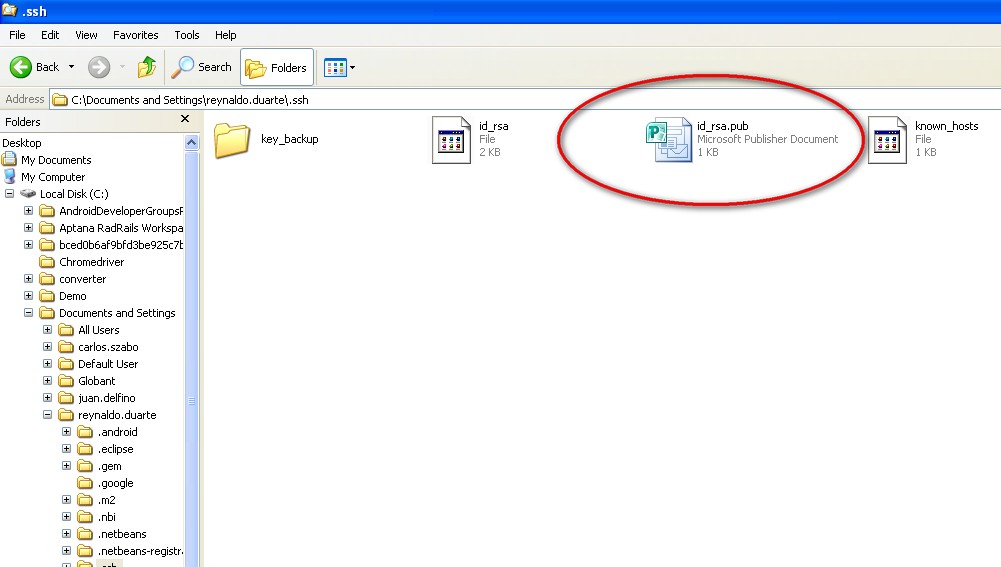
We have to fill the required fields and press the button **“Add Key”**

**keyssh-rsa**

The key"ssh-rsa" is locatedin the following directoryin the file "id\_rsa.pub".

**e.g.: C:\Documents and Settings\user name\.ssh**

Openthis file"id\_rsa.pub" to copy the key.



2. The followingcommand verifiesthat the keyis correctly configured.

Open the console"GitBash" orMS-Dos

**Note : you can use this Git commands both Unix and Windows**

**>ssh git@github.com**

-->It will require you to enter the user password (**password user account**)

It will display the following message in case that it was successful.

**Hi iarielduarte! You've successfully authenticated, but GitHub does not provideshell access.**

**Connection to github.com closed.**

In that way we have created a SSH key!

**3. Finally we have to authenticate our user account**

\* Setting up the user name for the development group:

**>gitconfig --global user.name "YourUserName"**(**user account**)

\* Setting up the email of the user account:

**>gitconfig --global user.email**[**xxx@gmail.com**](mailto:xxx@gmail.com) (**email user account**)

**Note : you can use this Git commands both Unix and Windows**

# Clone master repository

Openthe console"GitBash" orMS-Dos

Go to the directorywhere youwant to saverepository

e.g.:**> cd C:\**

We have to clone the git repository

**Note : you can use this Git commands both Unix and Windows**

**>git clone git@github.com:iarielduarte/Jericho.git**

-->It will require you to enter the user password (**password user account**)

It will display a message similar to the following one in case that it was successful.

**remote: Counting objects: 13, done.**

**remote: Compressing objects: 100% (13/13), done.**

**remote: Total 13 (delta 3), reused 0 (delta 0)**

**Receiving objects: 100% (13/13), done.**

**Resolving deltas: 100% (3/3), done.**

# GitCommands

[](http://rafurunet.files.wordpress.com/2012/02/octocat_constructocat2.jpg)

The main commands:

**>git add**

Add file contents to the index.

**>git status**

Show the working tree status.

**>git branch**

List, create, or delete branches.

**>git checkout**

Checkout a branch or paths to the working tree.

**>git commit**

Record changes to the repository.

**>git push**

Update remote refs along with associated objects.

**>git pull**

Fetch from and merge with another repository or a local branch.

**>git merge**

Join two or more development histories together.

**>gitrm**

Remove files from the working tree and from the index.

**>**[**git stash**](file:///C:\Program%20Files\Git\doc\git\html\git-stash.html)

Stash the changes in a dirty working directory away.

**>gitclone**

Clone a repository into a new directory.

**Note : you can use this Git commands both Unix and Windows.**

We can change the most used commands with the following aliases

So:

[alias]

  st = status

  ct = commit

  br = branch

  co = checkout

Openthe console"GitBash" orMS-Dos and enter this commnads.

**>gitconfig --global alias.ct commit**

**>gitconfig --global alias.br branch**

**>gitconfig --global alias.co checkout**

**>gitconfig --global alias.ststatus**

More Information:

[**https://help.github.com/articles/set-up-git**](https://help.github.com/articles/set-up-git)

Ultimaactualización 02/04/2013

**¿Cómo obtener un repositorio Git?**

Se puede hacer mediante dos métodos principales:

$gitinit: Toma un repositorio ya existente y lo importa dentro de git.

$git clone: Clona un repositorio ya existente en el servidor y lo clona de forma local

**$gitinit:**

Abrimos la consola y no colocamos dentro de la carpeta donde está el proyecto que queremos inicial en Git.

Ejemplo:

Creamos una carpeta python en la unidad C

$ mkdir Python

$ cd Python

$ gitinit

Listo se creo el repositorio en el Servidor, por lo tanto ahora es necesario agregar nuestro archives.

Si queremos agregar todos los archivos con una determinada extensión :

$ git add \*.py

$ git add README.md

$ gitcommit –m “primera versión del proyecto”

**$git clone:**

Importante para clonar un repositorio en el que nos inviten a participar este es el comando que necesitas, por lo tanto te debería haber pasado la ruta del repositorio en github.

Ejemplo:

$ git clone [git@github.com:iarielduarte/Python.git](mailto:git@github.com:iarielduarte/Python.git)ProjectPython

al final de la lineaagrege un nombre con el cual me gustaria que se nombre mi archivo clonado.

**¿Cómo ignorar archivos en Git?**

Abrimos la consola en directorio del proyecto git

$ cdC:\projectpython

$ edit .gitingnore

Se abrirá la consola de azul de edición de archivos.

Hace alt+F y alt+S con lo que guardamos el archivo.

Con esto hacemos un

$ dir

Y vamos a ver que tenemos un archivo nuevo llamado “.gitignore”

Vamos a editar este archivo y agregamos los siguientes comandos:

$ cat .gitignore

\*.[ao]

\*~

\*.[temp]

C:\projectpython\logs

Podemos indicar rutas completad de subdirectorios *“C:\projectpython\logs*

”

Se puede usar un patrón para indicar por ejemplo que ciertos tipos de archivos con extensión “.a “ no se deberían ignorar.

!Lib.a

Para indicar directorios completos podemos pedirle que lo ignore asi

direcorio/

Con el # le indicamos que es un comentario por lo tanto no hace caso de uso

# Ignore todos los archivos con extesion .py

\*.py[co]

Nuestro archivo final “.gitignore” quedo asi:

**# Gitdeberia ignorar todo estos patrones**

**$ cat .gitignore**

**\*.[ao]**

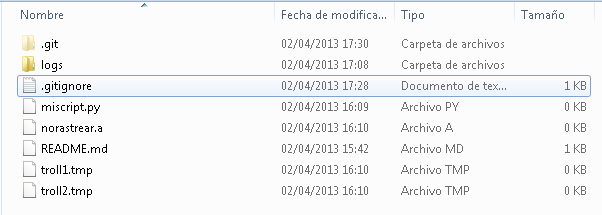
**\*~**

**tmp**

**\*.tmp**

**logs/**

Para ignore todos los archivos de nuestro repositorio Git.



<http://www.jedit.org/users-guide/globs.html>

En esta página tenemos la forma de administrar los patrones de extensiones para crear archivos .gitignore mejor detallados.

Ejemplo:

# a –commet esto es ignorado

\*.a # no rastrear los archivos .a

!lib.a # si debería rastrear lib.a

/TODO # solo ignora el archivo raíz TODO no subdirectorios

build/ # ignora todo los archivos en el directorio build/

doc/\*.txt # ignora doc/notes.txt, pero no doc/server/arch.txt

**¿Cómo saber en Git lo que he cambiado?**

**$gitdiff:**

Realmente saber que archivo se ha modificado y que esta listo para el commit lo hacemos con $git status, pero saber específicamente cuales son la líneas de código agregadas o modificadas con relación al original lo hacemos con $gitdiff

Veamos:

Tenemos un archivo inicial de python con este contenido:

defmain():

print "Convierte medidas inglesas a sistema metrico"

millas = input("Cuántas millas?: ")

pies = input("Y cuántos pies?: ")

pulgadas = input("Y cuántas pulgadas?: ")

metros = 1609.344 \* millas + 0.3048 \* pies + 0.0254 \* pulgadas

print "La longitud es de ", metros, " metros"

main()

Cambiamos algunas líneas del archivo:

defmain():

print "Ademas con python se puede hacer muchas cosas no solo script sin sentidos"

print "Realmente si es tan bueno porque no tenemos buena documentacion de python"

print "Este es un programa del sistema metrico"

millas = input("Cuántas millas?: ")

pies = input("Y cuántos pies?: ")

pulgadas = input("Y cuántas pulgadas?: ")

metros = 1609.344 \* millas + 0.3048 \* pies + 0.0254 \* pulgadas

print "La longitud es de ", metros, " metros"

main()

Hacemos un

$ gitaddmiscript.py

$ gitdiff –staged

Este es el resultado:

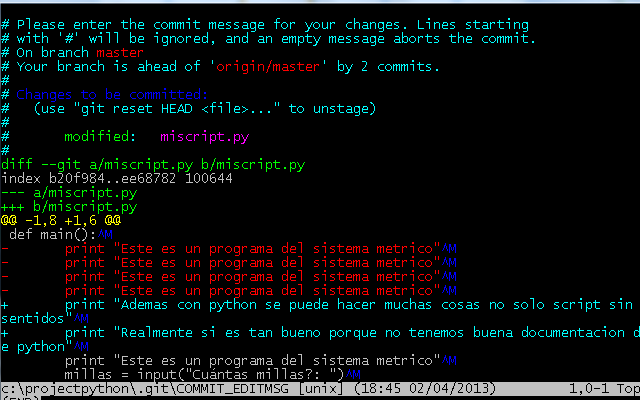


Bueno podemos ver que las líneas en rojo son aquellas que se borraron (-), las líneas verdes son aquellas que agregamos(+) y las que están en blanco son originalmente del programa inicial.

Algo parecido a esto nos muestra el comando:

$ gitcommit –v

Pero en este caso no pide que agreguemos un comentario en la parte superior de la consola asi:



**¿Cómo saltear en Git el area de Staged?**

**$gitcommit –m –a:**

Bueno Git no provee de un mecanismo rápido para hacer commit y son necesidad de pasar por el areastaged ($gitadd)

El comando es el siguiente:

$ gitcommit –m “Actualizacion de master” –a

**¿Cómo elimino archivos en Git?**

**$gitrm:**

En el caso que desemos borrar un archivo que se no rastreado por Git, es necesario hacer lo siguiente:

* Borramos el archivo físicamente de la carpeta local del proyecto
* Hacer un $ git status el cul no indica que el archivo esta deleted
* Y ejecutamos :

$ gitrm<Nombre del archivo>

$ git commit “deleted file”

$ git push

También podemos eliminar directamente el archivo con $ gitrm<Nombre del archivo>

Cuando el archivo ya se encuentra rastreado es necesario usar $ gitrm –f

Ejemplo:

$ gitrm –f <Nombre del archivo>

Para poder sacar un archivo de staged que ya ha sido trackeado, se usa el comando:

$ gitrm –cached<Nombre del archivo>

Con el comando rm podemos eliminar directorios completos asi:

$ gitrmlogs/

Tambien podemos eliminar archivos con ciertos patrones:

$ gitrm \\***~ :** elimina todos lo archives que contengan esos caracteres

$ gitrm log/ \\*.log : elimina todos los archives .log de ese directorio.

**¿Cómo renombra Git un archivo?**

**$git mv:**

Git puede renombrar archivos utitilizando $git mv [nombre viejo][nombre nuevo]

Ejemplo:

$ git mv miscript.py sample.py

$ git commit –m “rename file”

$ git push

<https://github.com/AlexZeitler/gitcheatsheet/blob/master/gitcheatsheet.pdf>

**¿Cómo hacer un rollback Git?**

**$git reset:**

Cuando se trabaja en GIT, suele pasar que hacemos un commit que no debia ocurrir o bien que éste commit no se haya formado correctamente, dando como resultado que nuestro codigo fuente se contamine con codigo de git.

*Suele pasar*, aunque siempre lo ideal es tener un branch principal y un branch propio para hacer movimientos al codigo y después hacer un merge.

Pero bueno, si te ha ocurrido algo así es necesario **hacer un rollback**, es decir, regresar el codigo a un estado anterior en alguncommit que hayamos hecho anteriormente y que sea adecuado para *volver a empezar.*

Lo primero es localizar el *commit* ideal para regresar a él, ésto lo haces desde la terminal dirigiéndote a tu repositorio y tecleando:

$ git log

Ahí saldrán los *commits* más recientes con su ID, y es precisamente el ID lo que requerimos para realizar el rollback.

Si trabajas con **github**puedes también localizar el ID del commit dirigiéndote a la pestaña “Commits” de tu proyecto y seleccionando el commit ideal, ahí aparecerá el ID de tu commit. (Rayos, soñarán con la palabra commit).

Una vez que tenemos el ID, podemos ejecutar lo siguiente:

$ git reset –hard <idCommit>

Por ejemplo:

$ git reset –hard 15b86083d234edc74be19565abef9217289q9sadaf

Y forzando un push para que éstos cambios nuevamente se reflejen del lado del servidor:

$ git push origin master -f

Si es necesario cambia el origin master por el branch en el que trabajas, por default es ése, la opcion**-f** indica que se va a forzar el push

Una vez que ha terminado hacemos un pull para asegurarnos que localmente están los cambios que se acaban de hacer de forma remota..

$ git pull

|  |  |
| --- | --- |
|  |  |

Asegúrate de realizar éste método en casos de emergencia evitando realizar cambios directamente en el master y creando mejor un nuevo branch para modificar el código y luego hacer un merge. De ésta forma evitas poner en riesgo el trabajo de todos los que colaboran en el proyecto o bien, poner en riesgo tu propio codigo.