03008915 - C/ Ferrocaril, 22, 03570 La Vila Joiosa – Tel 966870140 . Fax 966870141. – http://iesmarcoszaragoza.edu.gva.es

MANUAL DE GIT

Índice

1. Introducción....3
2. Instalación de GIT....3
3. Conceptos básicos....4
A. Repositorio....4
B. Confirmación o commit....4
C. Zonas de GIT....4
D. Estados de un archivos....4
E. Flujo de trabajo....5
F. HEAD....5
G. Rama....5
4. GIT desde la línea de comandos....6
A. Clonar un repositorio existente....6
B. Creación de un proyecto propio....6
C. Eliminando archivos de un proyecto....7
D. Renombrar ficheros....8
E. Ignorar ficheros....9
F. Deshacer cambios en la zona de trabajo y en la de preparación....9
5. RAMAS....11
A. Crear una rama....11
B. Cambiar de rama....12
C. Eliminar una rama....12
D. Fusionar ramas....13

Manual realizado en base a los documentos:

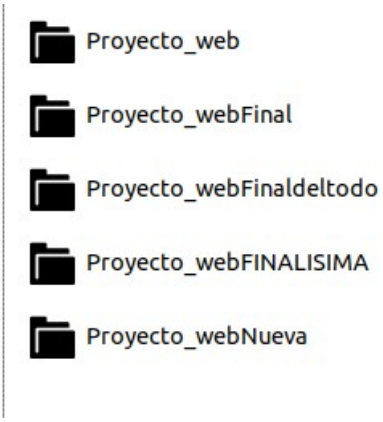
- Git Fundamentos (Jesús Amieiro Becerra)
- Introduction to GIT Dr. Chris Bourke cbourke@cse.unl.edu

Antonio Coloma Brotons
@antoniocoloma

03008915 - C/ Ferrocaril, 22, 03570 La Vila Joiosa – Tel 966870140 . Fax 966870141. – http://iesmarcoszaragoza.edu.gva.es

1. Introducción

Un proyecto ya sea una simple presentación en libreoffice impress o una extensa página web con acceso a bases de datos van a tener muchas modificaciones durante su ciclo de vida. Podemos encontrarnos esta estructura de carpetas:



Es por eso que necesitamos tener un Control de Versiones y saber en qué momento hemos modificado algo y si queremos volver a una versión anterior, poder hacerlo y tener todo guardado correctamente.

Existen muchos sistemas de control de versiones, como CVS, SVN, GIT. CVS es historia. SVN tiene un servidor centralizado, mientras que GIT es descentralizado. Cada copia local de cada desarrollador es una copia completa del proyecto. Y podemos tener el proyecto además en un servidor GIT (nosotros vamos a utilar github.com).

2. Instalación De GIT

Para la instalación de GIT debemos ir a la web:

https://git-scm.com/downloads

WINDOWS: El instalador instala automáticamente GIT.
MAC OS X: También al descargarse se inicia el asistente de instalación.

Tanto Windows como MAC OS X tienen la versión GUI de GIT pero nosotros nos vamos a basar en la parte CLI.

Linux: En las versiones debian-ubuntu podemos instalarlo de la siguiente manera:

```
#sudo apt install git-all
```

Si usamos una distrubución RedHat o CentOS, para instalarlos usamos:

```
#sudo dnf install git-all
```

03008915 - C/ Ferrocaril, 22, 03570 La Vila Joiosa – Tel 966870140 . Fax 966870141. – http://iesmarcoszaragoza.edu.gva.es

3. Conceptos Básicos

A. Repositorio

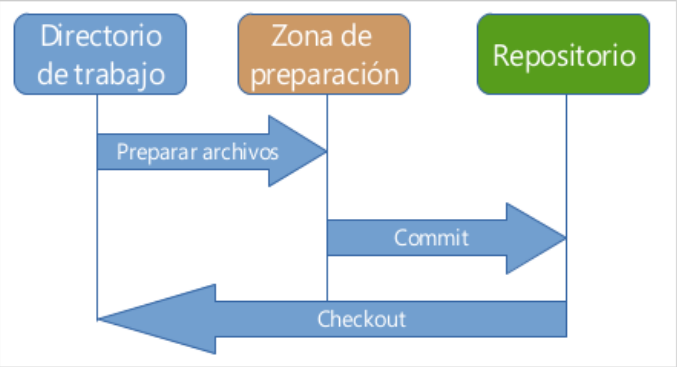
Es una base de datos que almacena todos los cambios que hemos realizado en nuestro proyecto y que han sido confirmados mediante **commit**.

B. Confirmación O Commit

Cuando confirmamos uno o varios archivos, grabamos en el repositorio una instantánea del proyecto o de los archivos modificados, así en cualquier momento podemos acceder a dicha información y saber los cambios que se han realizado en los ficheros del proyecto.

C. Zonas De GIT

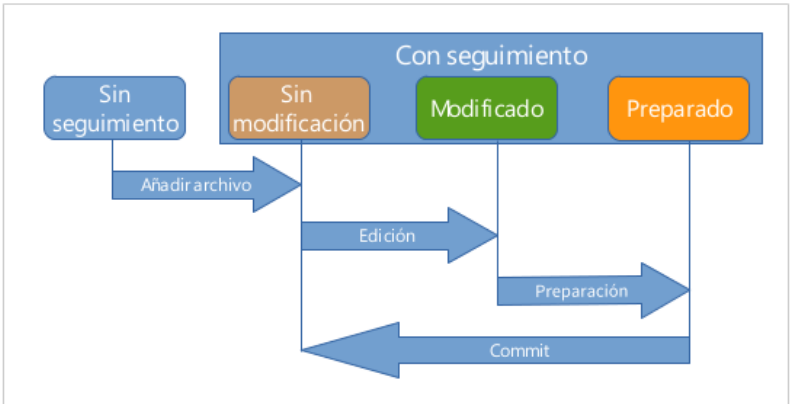
Las zonas en las que puede estar un archivo en el proyecto pueden ser:
a) Directorio de trabajo (working directory). Es el directorio en el que se encuentran los archivos del proyecto en nuestro sistema.



b) Zona de preparación (staging area). Es la zona en la que están los archivos preparados para la confirmación. No visible desde el directorio, solamente con **git**,

c) Repositorio (repository). Zona en la que se guardan todos los cambios que se han producido en los archivos del proyecto. En esta zona se guarda además quién ha realizado los cambios y cuándo.

D. Estados De Un Archivos



Cada archivo del proyecto puede estar bajo seguimiento o sin seguimiento por parte de GIT.
Si está bajo seguimiento puede estar:
- **Sin modificación**. El archivo es el mismo en el directorio de trabajo, preparación y repositorio.
- **Modificado**. El fichero de la zona de trabajo difiere de la zona de preparación y repositorio.
- **Preparado**. El fichero de la zona de trabajo y preparación difiere del repositorio.

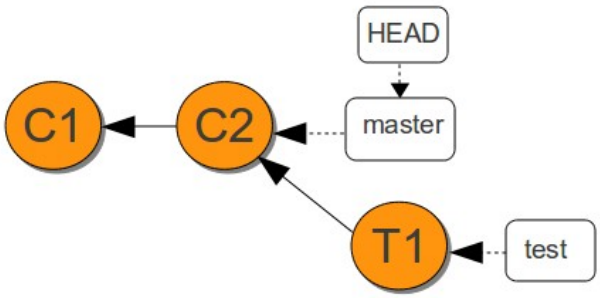
03008915 - C/ Ferrocaril, 22, 03570 La Vila Joiosa – Tel 966870140 . Fax 966870141. – http://iesmarcoszaragoza.edu.gva.es

E. Flujo De Trabajo

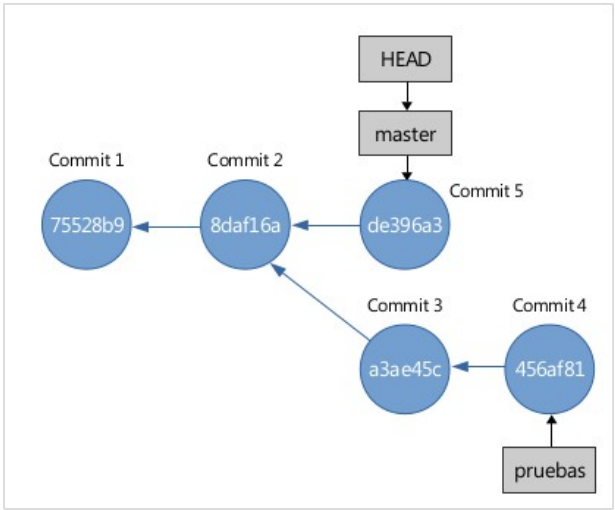
- Crear fichero en el directorio de trabajo (working directory) (**estado:** *sin seguimiento*)
- Añadir el fichero al proyecto. (**estado**: *bajo seguimiento, modificado*) el contenido del working directory difiere de la zona de preparación y repositorio.
- Preparar el archivo. (**estado**: *preparado*) Se añade el archivo tal y como se encuentra en el directorio de trabajo a la zona de preparación. El fichero está preparado para ir al repositorio. El contenido del fichero es el mismo en el directorio de trabajo y la zona de preparación.
- Confirmamos el fichero (**commit**). El fichero pasa a formar parte del repositorio y cambia su estado. (**estado**: *sin modificación*)
- Ahora empezamos de nuevo, modificando el fichero y pasando por todos los estados.

F. HEAD

HEAD simplemente es un puntero que apunta a la rama local en la que estas en un momento determinado. HEAD sigue todos los movimientos que hagas en el **git** de tu proyecto. HEAD se moverá en cada commit, checkout que se haga en el proyecto.



G. Rama



Una rama o branch es una bifurcación del proyecto que se crea para, por ejemplo, incluir una librería, corregir determinados ficheros, o algún cambio en el proyecto que no estamos seguros que pueda ser viable.
Al iniciar un proyecto, GIT crea una rama llamada **master** pero podemos crear más ramas y movernos entre ellas de tal manera que el contenido del directorio de trabajo irá cambiando.

En la ilustración podemos ver que se han realizado dos commits, se ha creado la rama prueba y en ella se han realizado otros dos commits. Por último se ha cambiado de nuevo a la rama master para hacer otro commit. El HEAD siempre apunta en el lugar donde nos encontramos actualmente.

03008915 - C/ Ferrocaril, 22, 03570 La Vila Joiosa – Tel 966870140 . Fax 966870141. – http://iesmarcoszaragoza.edu.gva.es

4. GIT Desde La Línea De Comandos

Existen diferentes clientes gráficos de GIT pero en nuestro caso vamos a utilizar la línea de comandos. Es un poco más lento el aprendizaje pero cuando se domina es un método más rápido que los GUI.

A. Clonar Un Repositorio Existente

Para empezar, podemos descargar un repositorio existente en github, pero antes de nada podemos verificar la versión de git.

```
#git –version

git version 2.17.1
```

Si no tenemos la salida de la versión de GIT y nos indica un error, debemos verificar que tenemos correctamente instalado GIT.

1. Entrar en un directorio en el que se descargará el proyecto. Este será nuestro directorio de trabajo.
2. Clonar el proyecto mediante el siguiente comandos

```
$git clone https://github.com/acolomab/Proy001-GIT

Clonando en 'Proy001-GIT'...
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 83 (delta 0), reused 3 (delta 0), pack-reused 79
Desempaquetando objetos: 100% (83/83), listo.
```

Una vez clonado el proyecto ya lo tenemos todo en nuestro ordenador, incluso podemos ir a versiones anteriores del proyecto y recuperar archivos concretos.

B. Creación De Un Proyecto Propio

Una vez que hayamos creado nuestro directorio y tengamos nuestros ficheros iniciales, vamos a crear nuestro repositorio GIT.

1. Inicializar el directorio

```
$git init
Inicializado repositorio Git vacío en ./GIT/.git/
```

2. Añadir los ficheros iniciales

```
$git add -all
```

03008915 - C/ Ferrocaril, 22, 03570 La Vila Joiosa – Tel 966870140 . Fax 966870141. – http://iesmarcoszaragoza.edu.gva.es

3. Realizar un Commit con un texto de identificación.

```
$git commit -m " Commit Inicial del proyecto"

[master (commit-raíz) 3feff91] Commit inicial
 1 file changed, 6 insertions(+)
 create mode 100644 README.md
```

4. Asociar el Repositorio con uno propio creado en github

```
$git remote add origin https://github.com/acolomab/Proy001-GIT.git
```

5. Subir los cambios al repositorio remoto

```
$git push -u origin master

Username for 'https://github.com': acolomab
Password for 'https://acolomab@github.com':
Contando objetos: 3, listo.
Delta compression using up to 2 threads.
Comprimiendo objetos: 100% (2/2), listo.
Escribiendo objetos: 100% (3/3), 324 bytes | 324.00 KiB/s, listo.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/acolomab/Proy001-GIT.git
 * [new branch]      master -> master
Rama 'master' configurada para hacer seguimiento a la rama remota 'master' de 'origin'.
```

En este caso, como el repositorio es privado, pide el usuario y la contraseña de github. No lo pediría si fuese público. A partir de aquí podemos ver en la web de nuestra cuenta de GITHUb el proyecto en el apartado de repositorios.

C. Eliminando Archivos De Un Proyecto

El borrado de ficheros de un proyecto es una cosa muy habitual, y en GIT se puede realizar de dos maneras.
Para realizarlo vamos a crear dos ficheros en nuestro proyecto:

```
$echo "Archivo a borrar" >borra.me
$echo "Segundo Archivo a borrar" >borra2.me

$git status
En la rama master
Tu rama está actualizada con 'origin/master'.

Archivos sin seguimiento:
  (usa "git add <archivo>..." para incluirlo a lo que se será confirmado)

        borra.me
        borra2.me

no hay nada agregado al commit pero hay archivos sin seguimiento presentes (usa "git add" para hacerles
seguimiento)
```

03008915 - C/ Ferrocaril, 22, 03570 La Vila Joiosa – Tel 966870140 . Fax 966870141. – http://iesmarcoszaragoza.edu.gva.es

Ahora los posicionamos en la zona de preparación y los confirmamos:

```
$ git add .
$ git commit -m "Añado dos ficheros para borrarlos"

[master 23a7043] Añado dos ficheros para borrarlos
 2 files changed, 2 insertions(+)
 create mode 100644 borra.me
 create mode 100644 borra2.me
```

Para borrar el fichero podemos eliminarlo directamente del directorio pero al revisar el estado, git nos avisa que debemos borrarlo de git también.

```
$rm borra.me
$git status
En la rama master
Tu rama está adelantada a 'origin/master' por 1 commit.
  (usa "git push" para publicar tus commits locales)

Cambios no rastreados para el commit:
  (usa "git add/rm <archivo>..." para actualizar a lo que se le va a hacer commit)
  (usa "git checkout -- <archivo>..." para descartar los cambios en el directorio de trabajo)

        borrado:        borra.me

sin cambios agregados al commit (usa "git add" y/o "git commit -a")

$git rm borra.me
$git commit -m “Eliminado fichero borra.me”
```

Pero podemos realizar directamente el borrado del fichero mediante git, que se encargará de eliminarlo del directorio de trabajo.

```
$git rm borra2.me
$git commit -m “borrado el segundo fichero”
```

D. Renombrar Ficheros.

Igual que al eliminar, al renombrar ficheros además de realizarlo en el directorio de trabajo, también debemos realizarlo en git, y posteriormente ya podremos hacer el commit.

```
$mv archivo nuevonombre
$git mv archvo nuevonombre
```

03008915 - C/ Ferrocaril, 22, 03570 La Vila Joiosa – Tel 966870140 . Fax 966870141. – http://iesmarcoszaragoza.edu.gva.es

E. Ignorar Ficheros

En un proyecto podemos tener múltiples ficheros que, aunque forman parte del proyecto, no necesitamos que estén dentro del control de versiones, por ejemplo, ficheros de log, ficheros objeto, o compilados, etc.

GIT nos permite ignorarlos a la hora de realizar commits. Para ello debemos crear un fichero llamado .gitignore en el directorio raiz de nuestro proyecto. Y después por cada línea podemos especificar el nombre del fichero (o patrón) a ignorar.

El fichero .gitignore

- Líneas en blanco no son tratadas
- Comentarios empiezan por #
- "*" es un comodín de cero o más caracteres
- "?·es un comodín de un carácter cualquiera
- Se pueden usar expresiones regulares [0-9a-z] …

F. Deshacer Cambios En La Zona De Trabajo Y En La De Preparación

Al realizar cambios en ficheros del área de trabajo, git lo detecta y nos indica que se han realizado cambios.

```
$git status
En la rama master
Tu rama está actualizada con 'origin/master'.

Cambios no rastreados para el commit:
  (usa "git add <archivo>..." para actualizar lo que será confirmado)
  (usa "git checkout -- <archivo>..." para descartar los cambios en el directorio de trabajo)

        modificado:     README.md

sin cambios agregados al commit (usa "git add" y/o "git commit -a")
```

GIT nos indica que podemos realizar un "checkout "

```
$git checkout -- README.md
$ git status
En la rama master
Tu rama está actualizada con 'origin/master'.

nada para hacer commit, el árbol de trabajo esta limpio" para descartar los cambios en el directorio de trabajo.
```

Ahora si lo que hemos hecho es pasar el fichero con cambios a la zona de preparación:

03008915 - C/ Ferrocaril, 22, 03570 La Vila Joiosa – Tel 966870140 . Fax 966870141. – http://iesmarcoszaragoza.edu.gva.es

```
$git status
En la rama master
Tu rama está actualizada con 'origin/master'.

Cambios a ser confirmados:
  (usa "git reset HEAD <archivo>..." para sacar del área de stage)

        modificado:     README.md
```

Nos indica que debemos realizar un reset HEAD:

```
$git reset HEAD README.md
Cambios fuera del área de stage tras el reset:
M       README.md
```

En estos momentos ya lo hemos sacado de la zona de preparación, ahora podemos eliminar los cambios como anteriormente.

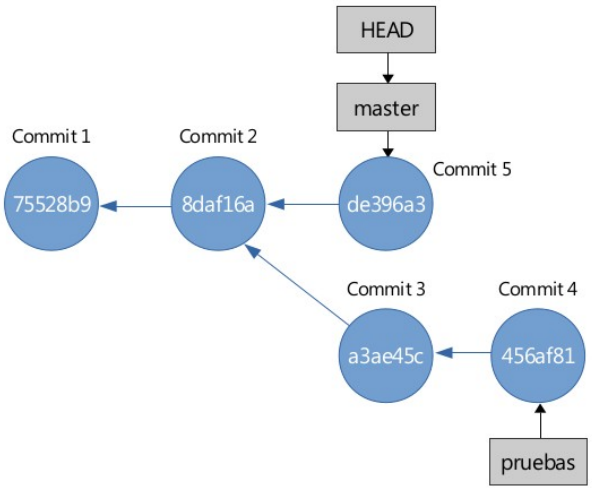
```
$git checkout -- README.md
$ git status
En la rama master
Tu rama está actualizada con 'origin/master'.

nada para hacer commit, el árbol de trabajo esta limpio" para descartar los cambios en el directorio de trabajo.
```

03008915 - C/ Ferrocaril, 22, 03570 La Vila Joiosa – Tel 966870140 . Fax 966870141. – http://iesmarcoszaragoza.edu.gva.es

5. RAMAS

Cuando estamos trabajando en un proyecto ya sea de desarrollo web, programación, o una simple presentación a menudo nos interesa probar una serie de cosas ya sea una librería nueva, una hoja de estilos diferente o incluso cambiar imágenes de diapositivas por videos.



En ese caso GIT nos permite realizar RAMAS del proyecto. Una rama es un camino alternativo del proyecto que seguimos en un determinado momento.

La rama que crea GIT al iniciar un proyecto se denomina **master** y es sobre la que empezamos el trabajo. A partir de ahí podemos crear más ramas.

A. Crear Una Rama

```
$ mkdir prueba_ramas
$ cd prueba_ramas/
prueba_ramas$ git init

Inicializado repositorio Git vacío en prueba_ramas/.git/

prueba_ramas$ > uno.txt
prueba_ramas$ git add uno.txt
prueba_ramas$ git commit -m "Añado uno.txt"

[master (commit-raíz) a70354c] Añado uno.txt
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 uno.txt
prueba_ramas$ >dos.txt
prueba_ramas$ git add dos.txt
prueba_ramas$ git commit -m "añado dos.txt"
[master 9cc4d1b] añado dos.txt

 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 dos.txt
prueba_ramas$ git log –oneline

9cc4d1b (HEAD -> master) añado dos.txt
a70354c Añado uno.txt
```

Con git branch podemos ver las ramas que tenemos y en cual nos encontramos.

```
prueba_ramas$git branch

* master
```

Para crear una nueva rama únicamente le ponemos el nombre de la rama al comando branch.

```
prueba_ramas$git branch pruebas
```

Al mostrar de nuevo las ramas podemos ver las dos ramas y que nos encontramos en la rama principal **master**.

```
prueba_ramas$ git branch
* master
  pruebas
```

B. Cambiar De Rama

Utilizamos el comando **checkout** para cambiarnos de rama.

```
prueba_ramas$ git checkout pruebas
Cambiado a rama 'pruebas'
```

Ahora todas las modificaciones que realicemos estarán en la rama **pruebas** manteniendo la rama **master** tal y como estaba hasta que las fusionemos.

C. Eliminar Una Rama

A menudo creamos una rama y realizamos los cambios pertinentes pero nos damos cuenta que no son fructíferos. En este caso podemos borrar la rama (siempre que no estemos en ella) con el parámetro **-d**. A continuación creamos una rama **a_eliminar** que eliminaremos finalmente.

```
prueba_ramas$ git branch
* master
  pruebas

prueba_ramas$ git checkout pruebas
Cambiado a rama 'pruebas'

prueba_ramas$ git checkout master
Cambiado a rama 'master'

prueba_ramas$ git branch
* master
  pruebas

prueba_ramas$ git branch a_eliminar
prueba_ramas$ git branch
  a_eliminar
* master
  pruebas

prueba_ramas$ git checkout a_eliminar
Cambiado a rama 'a_eliminar'

prueba_ramas$ git branch
* a_eliminar
  master
  pruebas

prueba_ramas$ git branch -d a_eliminar
error:    No    se    puede    borrar    rama    'a_eliminar'    que    cuenta    con    checkout    en
'/home/antonio/Documentos/PROYECTOSGIT/prueba_ramas'

prueba_ramas$ git checkout master
Cambiado a rama 'master'

prueba_ramas$ git branch -d a_eliminar
Eliminada la rama a_eliminar (era 9cc4d1b)..
```

03008915 - C/ Ferrocaril, 22, 03570 La Vila Joiosa – Tel 966870140 . Fax 966870141. – http://iesmarcoszaragoza.edu.gva.es

D. Fusionar Ramas

Cuando hemos bifurcado el proyecto porque hemos querido probar una librería y al final la librería funciona correctamente, podemos hacer que laa rama se fusione con la rama master para que dicha librería forme parte del proyecto final. Esto lo vamos a hacer con el comando **merge**.

```
prueba_ramas$ git checkout master
Cambiado a rama 'master'

prueba_ramas$ git log --oneline --graph --decorate --all
* 5540c01 (pruebas)  modificado uno
* 9cc4d1b (HEAD -> master) añado dos.txt
* a70354c Añado uno.txt

prueba_ramas$ git merge pruebas
Actualizando 9cc4d1b..5540c01
Fast-forward
 uno.txt | 1 +
 1 file changed, 1 insertion(+)

prueba_ramas$ git log --oneline --graph --decorate --all
* 5540c01 (HEAD -> master, pruebas)  modificado uno
* 9cc4d1b añado dos.txt
* a70354c Añado uno.txt
```

Vemos que una vez que hemos fusionado las dos ramas, ambas apuntan al mismo commit.

Si nos damos cuenta que hemos fusionado pero queremos volver al estado anterior de la fusión, solamente tenemos que abortar la fusión con el comando: