Uso básico de Git

Vamos primero a recordar algunas cuestiones básicas de git.

Secciones principales de un repositorio `git`

En un repositorio `git` podemos diferenciar las siguientes secciones:

- *Workspace*
- *Staging area (aparece en la imagen como Index)*
- *Local repository*
- *Remote repository*

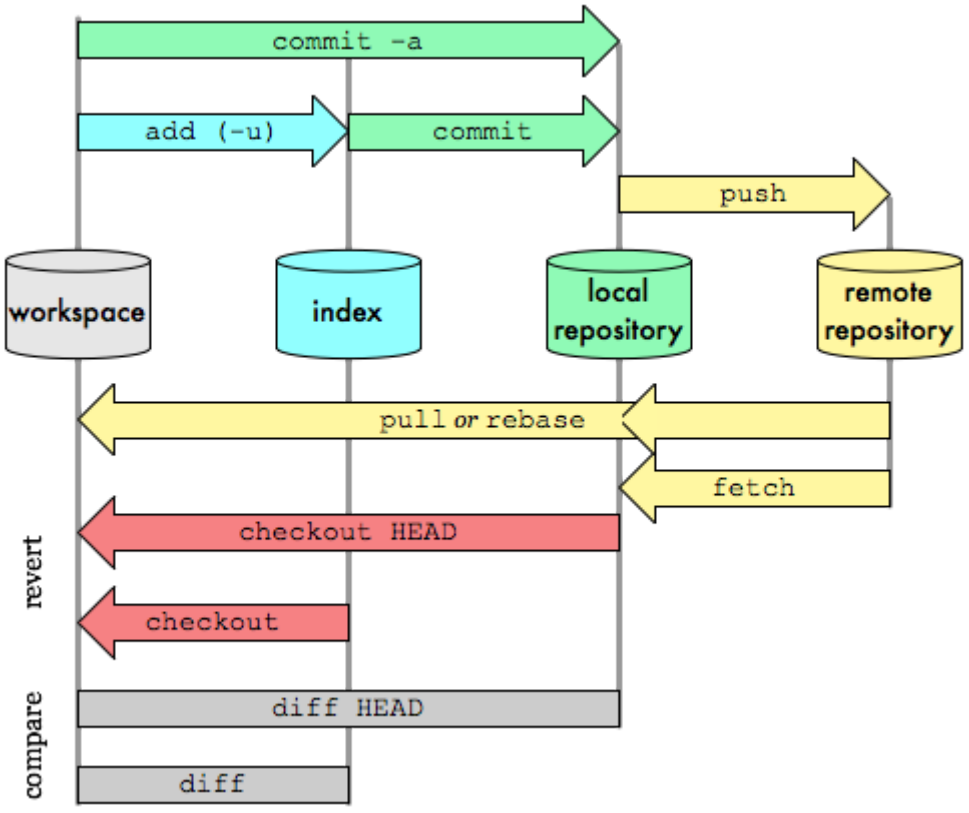


Figura 1: Imagen de Oliver Steele.

Estados de un archivo en `git`

Un archivo puede estar en alguno de los siguientes estados:

- Sin seguimiento (*untracked*)
- Modificado (*modified*)
- Preparado (*staged*)
- Confirmado (*commited*)

El siguiente diagrama muestra en qué sección se puede encontrar cada archivo en función de su estado.

```
+-------------+  +-------------+  +-------------+  +-------------+
|  Working    |  |   Staging   |  |    Local    |  |    Remote   |
|  Directory  |  |     Area    |  |  Repository |  |  Repository |
+------+------+  +------+------+  +------+------+  +------+------+
       |                |                |                |
   Untracked            |                |                |
       |                |                |                |
   Modified          Staged          Commited             |
       |                |                |                |
       +                +                +                +
```

Para consultar el estado de los archivos usamos el comando:

```
git status
```

Este comando es muy usado ya que es fundamental conocer el estado de los archivos de nuestro repositorio.

Utilizando distintos comandos podemos pasar los archivos de una sección a otra y cambiar su estado. A continuación veremos los comandos básicos que nos permitirán una utilización básica de git usando como repositorio remoto GitHub.

Para ir recordando los distintos comandos que vayamos aprendiendo es muy recomendable utilizar un "cheatsheet" o ir creando el nuestro propio. Aquí tenéis uno que podéis utilizar. Podéis ir marcando con un subrayador los comandos que vais aprendiendo.

Crear un proyecto

Crear un programa "Hola Mundo"

Creamos un directorio donde colocar el código

```
$ mkdir curso-de-git
$ cd curso-de-git
```

Creamos un fichero `hola.php` que muestre Hola Mundo.

```php
<?php
echo "Hola Mundo\n";
?>
```

Crear el repositorio

Para crear un nuevo repositorio se usa la orden `git init`

```
$ git init
Initialized empty Git repository in /home/admin/curso-de-git/.git/
```

Comprueba con un `ls -la` que hay un nuevo directorio oculto .git donde git guardará toda la información que necesite de forma transparente al usuario.

Al inicializar nuestro proyecto, el archivo `hola.php` estará en el Workspace o Working Directory.

Añadir la aplicación

Vamos a almacenar el archivo que hemos creado en el repositorio para poder trabajar, después explicaremos para qué sirve cada orden.

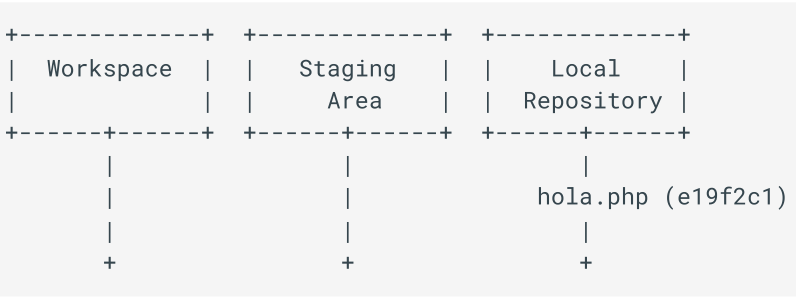
```
$ git add hola.php
```

Al ejecutar el git add, el archivo pasará a la "Staging area" o área de preparación. Podemos ejecutar el siguiente comando tras cada orden para ir comprobando el estado del proyecto:

```
    $ git status
```

```
$ git commit -m "Creación del proyecto"
[master (root-commit) e19f2c1] Creación del proyecto
 1 file changed, 3 insertions(+)
 create mode 100644 hola.php
```

El archivo pasará al "Local Ropository" y se le asignará un hash o código de inserción.



Comprobar el estado del repositorio

Con la orden `git status` podemos ver en qué estado se encuentran los archivos de nuestro repositorio.

```
$ git status
On branch master
nothing to commit (working tree clean)
```

Si modificamos el archivo `hola.php`:

```
<?php
@print "Hola {$argv[1]}\n";
?>
```

Y volvemos a comprobar el estado del repositorio:

```
    $ 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:   hola.php

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

Nos indica que `hola.php` vuelve a estar en el working directory, porque lo hemos modificado. Y nos dice que podemos usar `git add hola.php` para volverlo a pasar a la "Staging Area" o bien `git restore hola.php` para descartar los cambios en el Working Directory y recuperar la version anterior en "Local Repository". En nuestro esquema veremos más arriba las versiones más recientes y más abajo las más antiguas, que es como nos lo mostrarán los

comandos git para ver históricos. Es como una pila en la que la nueva versión queda sobre las anteriores.

```
+-------------+  +-------------+  +-------------+
|  Working    |  |   Staging   |  |    Local    |
|  Directory  |  |     Area    |  |  Repository |
+------+------+  +------+------+  +------+------+
       |                |                |
   hola.php             |                |
       |                |             hola.php (e19f2c1)
       |                |                |
       +                +                +
```

Añadir cambios

Con la orden `git add` indicamos a git que prepare los cambios para que sean almacenados.

```
    $ git add hola.php
    $ git status
    On branch master
    Changes to be committed:
      (use "git restore --staged <file>..." to unstage)

      modified:   hola.php
```

Hemos vuelto a pasar `hola.php` a la "Staging area"

```
+-------------+  +-------------+  +-------------+
|  Working    |  |   Staging   |  |    Local    |
|  Directory  |  |     Area    |  |  Repository |
+------+------+  +------+------+  +------+------+
       |                |                |
       |             hola.php            |
       |                |             hola.php (e19f2c1)
       |                |                |
       +                +                +
```

Confirmar los cambios

Con la orden `git commit` confirmamos los cambios definitivamente, lo que hace que se guarden permanentemente en nuestro repositorio.

```
    $ git commit -m "Parametrización del programa"
    [master efc252e] Parametrización del programa
     1 file changed, 1 insertion(+), 1 deletion(-)
    $ git status
    On branch master
    nothing to commit, working tree clean
```

El archivo pasará al "Local Ropository" con un nuevo hash.

```
+-------------+  +-------------+  +-------------+
|  Working    |  |   Staging   |  |    Local    |
|  Directory  |  |     Area    |  |  Repository |
+------+------+  +------+------+  +------+------+
       |                |                |
       |                |             hola.php (efc252e)
       |                |             hola.php (e19f2c1)
       |                |                |
       +                +                +
```

Diferencias entre *workdir* y *staging*.

Modificamos nuestra aplicación para que soporte un parámetro por defecto y añadimos los cambios.

```
<?php
$nombre = isset($argv[1]) ? $argv[1] : "Mundo";
@print "Hola, {$nombre}\n";
?>
```

Al modificarlo volvemos a tenerlo en "Working Directory"

```
+-------------+  +-------------+  +-------------+
|  Working    |  |   Staging   |  |    Local    |
|  Directory  |  |     Area    |  |  Repository |
+------+------+  +------+------+  +------+------+
       |                |                |
     hola.php           |                |
       |                |             hola.php (efc252e)
       |                |             hola.php (e19f2c1)
       |                |                |
       +                +                +
```

Este vez añadimos los cambios a la fase de *staging* pero sin confirmarlos (*commit*).

```
git add hola.php
```

Al hacer el add volvemos a tenerlo en "Staging Area"

```
+-------------+  +-------------+  +-------------+
|  Working    |  |   Staging   |  |    Local    |
|  Directory  |  |     Area    |  |  Repository |
+------+------+  +------+------+  +------+------+
       |                |                |
       |             hola.php            |
       |                |             hola.php (efc252e)
       |                |             hola.php (e19f2c1)
```

```
        |                |                |
        +                +                +
```

Volvemos a modificar el programa para indicar con un comentario lo que hemos hecho.

```
<?php
// El nombre por defecto es Mundo
$nombre = isset($argv[1]) ? $argv[1] : "Mundo";
@print "Hola, {$nombre}\n";
?>
```

Y vemos el estado en el que está el repositorio

```
    $ git status
    On branch master
    Changes to be committed:
      (use "git restore --staged <file>..." to unstage)

      modified:   hola.php

    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:   hola.php
```

Podemos ver como aparecen el archivo `hola.php` dos veces. El primero está preparado para ser confirmado y está almacenado en la "Staging Area" y es el que hicimos el add en primer lugar. El segundo indica que el archivo `hola.php` está modificado otra vez en la zona de trabajo "Working Directory". Como vemos en el gráfico, el mismo archivo está en 2 zonas distintas en este momento, además de las 2 versiones anteriores que hicimos commit.

```
+-------------+  +-------------+  +-------------+
|  Working    |  |   Staging   |  |    Local    |
|  Directory  |  |     Area    |  |  Repository |
+------+------+  +------+------+  +------+------+
       |                |                |
     hola.php        hola.php            |
       |                |            hola.php (efc252e)
       |                |            hola.php (e19f2c1)
       |                |                |
       +                +                +
```

Warning

Si volvieramos a hacer un `git add hola.php` sobreescribiríamos los cambios previos que había en la zona de *staging*.

Almacenamos los cambios por separado:

```
$ git commit -m "Se añade un parámetro por defecto"
[master 3283e0d] Se añade un parámetro por defecto
 1 file changed, 2 insertions(+), 1 deletion(-)

$ 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:   hola.php

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

Con el commit hemos pasado el hola.php que estaba en Staging Area al Local Repository quedando en el "Working Directory" el último que habíamos editado.

```
+-------------+  +-------------+  +-------------+
|  Working    |  |   Staging   |  |    Local    |
|  Directory  |  |     Area    |  |  Repository |
+------+------+  +------+------+  +------+------+
       |                |                |
     hola.php           |                |
       |                |             hola.php (3283e0d)
       |                |             hola.php (efc252e)
       |                |             hola.php (e19f2c1)
       |                |                |
       +                +                +
```

Si ahora hacemos un git add:

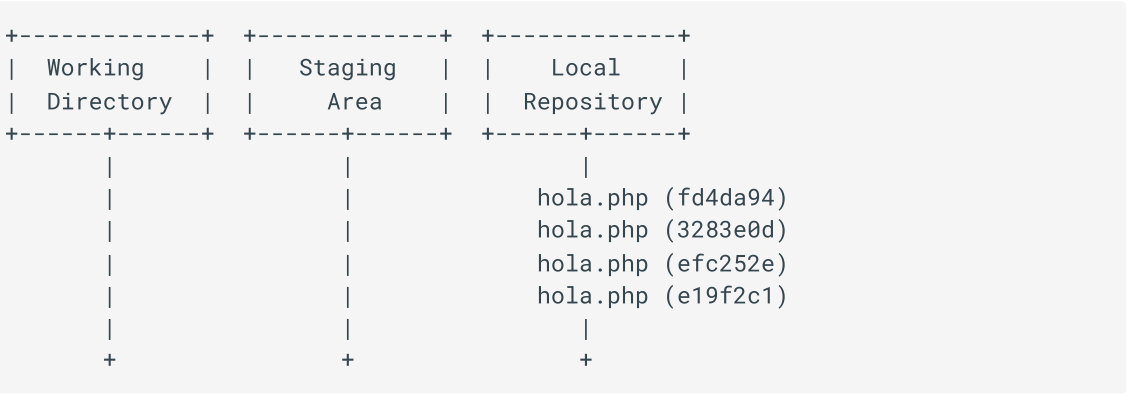
```
$ git add .
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)

  modified:   hola.php
```

El archivo en Working Directory pasa a Staging Area.

```
+-------------+  +-------------+  +-------------+
|  Working    |  |   Staging   |  |    Local    |
|  Directory  |  |     Area    |  |  Repository |
+------+------+  +------+------+  +------+------+
       |                |                |
       |             hola.php            |
       |                |             hola.php (3283e0d)
       |                |             hola.php (efc252e)
       |                |             hola.php (e19f2c1)
       |                |                |
       +                +                +
```

Y si ahora hacemos un commit.

```
$ git commit -m "Se añade un comentario al cambio del valor por defecto"
[master fd4da94] Se añade un comentario al cambio del valor por defecto
 1 file changed, 1 insertion(+)
```



Info

El valor "." despues de `git add` indica que se añadan todos los archivos de forma recursiva.

Warning

Cuidado cuando uses `git add .` asegúrate de que no estás añadiendo archivos que no quieres añadir.

Ignorando archivos

La orden `git add .` o `git add nombre_directorio` es muy cómoda, ya que nos permite añadir todos los archivos del proyecto o todos los contenidos en un directorio y sus subdirectorios. Es mucho más rápido que tener que ir añadiéndolos uno por uno. El problema es que, si no se tiene cuidado, se puede terminar por añadir archivos innecesarios o con información sensible.

Por lo general se debe evitar añadir archivos que se hayan generado como producto de la compilación del proyecto, los que generen los entornos de desarrollo (archivos de configuración y temporales) y aquellos que contentan información sensible, como contraseñas o tokens de autenticación. Por ejemplo, en un proyecto de `C/C++`, los archivos objeto no deben incluirse, solo los que contengan código fuente y los make que los generen.

Para indicarle a *git* que debe ignorar un archivo, se puede crear un fichero llamado *.gitignore*, bien en la raíz del proyecto o en los subdirectorios que queramos. Dicho fichero puede

contener patrones, uno en cada línea, que especiquen qué archivos deben ignorarse. El formato es el siguiente:

```
# .gitignore
dir1/           # ignora todo lo que contenga el directorio dir1
!dir1/info.txt  # El operador ! excluye del ignore a dir1/info.txt (sí se
guardaría)
dir2/*.txt      # ignora todos los archivos txt que hay en el directorio dir2
dir3/**/*.txt   # ignora todos los archivos txt que hay en el dir3 y sus
subdirectorios
*.o             # ignora todos los archivos con extensión .o en todos los
directorios
```

Cada tipo de proyecto genera sus ficheros temporales, así que para cada proyecto hay un `.gitignore` apropiado. Existen repositorios que ya tienen creadas plantillas. Podéis encontrar uno en https://github.com/github/gitignore

Ignorando archivos globalmente

Si bien, los archivos que hemos metido en `.gitignore`, deben ser aquellos ficheros temporales o de configuración que se pueden crear durante las fases de compilación o ejecución del programa, en ocasiones habrá otros ficheros que tampoco debemos introducir en el repositorio y que son recurrentes en todos los proyectos. En dicho caso, es más útil tener un *gitignore* que sea global a todos nuestros proyectos. Esta configuración sería complementaria a la que ya tenemos. Ejemplos de lo que se puede ignorar de forma global son los ficheros temporales del sistema operativo (`*~`, `.nfs*`) y los que generan los entornos de desarrollo.

Para indicar a *git* que queremos tener un fichero de *gitignore* global, tenemos que configurarlo con la siguiente orden:

```
git config --global core.excludesfile $HOME/.gitignore_global
```

Ahora podemos crear un archivo llamado `.gitignore_global` en la raíz de nuestra cuenta con este contenido:

```
# Compiled source #
###################
*.com
*.class
*.dll
*.exe
*.o
*.so

# Packages #
############
# it's better to unpack these files and commit the raw source
# git has its own built in compression methods
```

```
*.7z
*.dmg
*.gz
*.iso
*.jar
*.rar
*.tar
*.zip

# Logs and databases #
######################
*.log
*.sql
*.sqlite

# OS generated files #
######################
.DS_Store
.DS_Store?
._*
.Spotlight-V100
.Trashes
ehthumbs.db
Thumbs.db
*~
*.swp

# IDEs               #
######################
.idea
.settings/
.classpath
.project
```

Trabajando con el historial

Observando los cambios

Con la orden `git log` podemos ver todos los cambios que hemos hecho. Antes de nada vuelve al directorio `curso-de-git` en el que estábamos trabajando:

```
    $ git log
    commit fd4da946326fbe8b24e89282ad25a71721bf40f6  (HEAD -> master)
    Author: Sergio Gómez <sergio@uco.es>
    Date:   Sun Jun 16 12:51:01 2013 +0200

        Se añade un comentario al cambio del valor por defecto

    commit 3283e0d306c8d42d55ffcb64e456f10510df8177
    Author: Sergio Gómez <sergio@uco.es>
    Date:   Sun Jun 16 12:50:00 2013 +0200

        Se añade un parámetro por defecto
```

```
    commit efc252e11939351505a426a6e1aa5bb7dc1dd7c0
    Author: Sergio Gómez <sergio@uco.es>
    Date:   Sun Jun 16 12:13:26 2013 +0200

        Parametrización del programa

    commit e19f2c1701069d9d1159e9ee21acaa1bbc47d264
    Author: Sergio Gómez <sergio@uco.es>
    Date:   Sun Jun 16 11:55:23 2013 +0200

        Creación del proyecto
```

Para salir escribe `q`.

Recuerda los distintos hash que se habían generado cada vez que hacíamos un commit. Fíjate que antes vimos solo los primeros caracteres hexadecimales del hash.

```
+-------------+  +-------------+  +-------------+
|  Working    |  |   Staging   |  |    Local    |
|  Directory  |  |     Area    |  |  Repository |
+------+------+  +------+------+  +------+------+
       |                |                |
       |                |             hola.php (fd4da94)
       |                |             hola.php (3283e0d)
       |                |             hola.php (efc252e)
       |                |             hola.php (e19f2c1)
       |                |                |
       +                +                +
```

También es posible ver versiones abreviadas o limitadas, dependiendo de los parámetros:

```
    $ git log --oneline
    fd4da94 (HEAD -> master) Se añade un comentario al cambio del valor por
defecto
    3283e0d Se añade un parámetro por defecto
    efc252e Parametrización del programa
    e19f2c1 Creación del proyecto
```

Prueba estas otras opciones y comprobarás lo que hace cada una.

```
    git log --oneline --max-count=2
    git log --oneline --since='5 minutes ago'
    git log --oneline --until='5 minutes ago'
    git log --oneline --author=sergio   # Cambia sergio por tu nombre de
usuario
    git log --oneline --all
```

Una versión muy útil de `git log` es la siguiente, pues nos permite ver en que lugares está master y HEAD, entre otras cosas:

```
    $ git log --pretty=format:'%h %ad | %s%d [%an]' --graph --date=short
    * fd4da94 2013-06-16 | Se añade un comentario al cambio del valor por
```

```
defecto (HEAD, master) [Sergio Gómez]
    * 3283e0d 2013-06-16 | Se añade un parámetro por defecto [Sergio Gómez]
    * efc252e 2013-06-16 | Parametrización del programa [Sergio Gómez]
    * e19f2c1 2013-06-16 | Creación del proyecto [Sergio Gómez]
```

Crear alias

Como estas órdenes son demasiado largas, Git nos permite crear alias para crear nuevas órdenes parametrizadas. Para ello podemos configurar nuestro entorno con la orden `git config` de la siguiente manera:

```
git config --global alias.hist "log --pretty=format:'%h %ad | %s%d [%an]' --
graph --date=short"
```

Ahora basta con ejecutar:

```
git hist
```

Example

Puedes configurar incluso alias para abreviar comandos. Algunos ejemplos de alias útiles:

```
git config --global alias.br branch
git config --global alias.co checkout
git config --global alias.ci commit
git config --global alias.st "status -u"
git config --global alias.cane "commit --amend --no-edit"
```

Recuperando versiones anteriores

Cada cambio es etiquetado por un hash, para poder regresar a ese momento del estado del proyecto se usa la orden `git checkout`. Prueba con el hash de tu primer commit.

```
    $ git checkout e19f2c1
    Note: switching to 'e19f2c1'.

    You are in 'detached HEAD' state. You can look around, make experimental
    changes and commit them, and you can discard any commits you make in this
    state without impacting any branches by switching back to a branch.

    If you want to create a new branch to retain commits you create, you may
    do so (now or later) by using -c with the switch command. Example:

    git switch -c <new-branch-name>

    Or undo this operation with:
```

```
    git switch -

    urn off this advice by setting config variable advice.detachedHead to
false

    HEAD is now at e19f2c1 Parametrización del programa
```

Hemos vuelto a aquí:

```
+-------------+  +-------------+  +-------------+
|  Working    |  |   Staging   |  |    Local    |
|  Directory  |  |     Area    |  |  Repository |
+------+------+  +------+------+  +------+------+
       |                |                |
       |                |             hola.php (e19f2c1)
       |                |                |
       +                +                +
```

El aviso que nos sale nos indica que estamos en un estado donde no trabajamos en ninguna rama concreta. Eso significa que los cambios que hagamos podrían "perderse" porque si no son guardados en una nueva rama, en principio no podríamos volver a recuperarlos. Hay que pensar que Git es como un árbol donde un nodo tiene información de su nodo padre, no de sus nodos hijos, con lo que siempre necesitaríamos información de dónde se encuentran los nodos finales o de otra manera no podríamos acceder a ellos. Vamos a comprobarlo.

Edita `hola.php` y añade un comentario. Haz un commit y comentalo como "Prueba en detached HEAD". Después haz un log para comprobar la rama.

Antes de continuar desharemos con

```
$ git switch -
```

Comprueba que todos los cambios que hiciste en el "detached HEAD" se han perdido.

Volver a la última versión de la rama master.

Ya hemos tratado el concepto de HEAD, pero no lo hemos definido formalmente. HEAD hace referencia al puntero que señala a la referencia actual de la rama activa o commit en el que estás trabajando. En otras palabras, es un indicador que te dice en qué commit estás situado en ese momento.

Vamos a ver otra forma de llevar nuevamente el HEAD a nuestro primer commit, como hicimos en el punto anterior. ¿Recuerdas cómo lo hicimos?

A continuación usamos `git checkout` indicando el nombre de la rama:

```
$ git checkout master
Previous HEAD position was e19f2c1... Creación del proyecto
```

Comprueba en qué posición estás con alguno de los comandos que aprendiste anteriormente.

Etiquetando versiones

Para poder recuperar versiones concretas en la historia del repositorio, podemos etiquetarlas, lo cual es más facil que usar un hash. Para eso usaremos la orden `git tag`.

```
$ git tag v1
```

Ahora vamos a etiquetar la versión inmediatamente anterior como v1-beta. Para ello podemos usar los modificadores `^` o `~` que nos llevarán a un ancestro determinado. Las siguientes dos órdenes son equivalentes:

```
$ git checkout v1^
$ git checkout v1~1
```

Asignamos ahora el tag v1-beta a la versión anterior.

```
$ git tag v1-beta
```

Vuelve al estado final con

```
$ git switch -
```

Si ejecutamos la orden sin parámetros nos mostrará todas las etiquetas existentes.

```
$ git tag
v1
v1-beta
```

Y para verlas en el historial:

```
$ git hist master --all
* fd4da94 2013-06-16 | Se añade un comentario al cambio del valor por defecto
(tag: v1, master) [Sergio Gómez]
* 3283e0d 2013-06-16 | Se añade un parámetro por defecto (HEAD, tag: v1-beta)
[Sergio Gómez]
* efc252e 2013-06-16 | Parametrización del programa [Sergio Gómez]
* e19f2c1 2013-06-16 | Creación del proyecto [Sergio Gómez]
```

En nuestro esquema:

```
+-------------+  +-------------+  +-------------+
|  Working    |  |   Staging   |  |    Local    |
|  Directory  |  |     Area    |  |  Repository |
+------+------+  +------+------+  +------+------+
```

```
        |                |                |
        |                |             hola.php (fd4da94) tag: v1
        |                |             hola.php (3283e0d) tag: v1-beta
        |                |             hola.php (efc252e)
        |                |             hola.php (e19f2c1)
        |                |                |
        +                +                +
```

Borrar etiquetas

Para borrar etiquetas:

```
git tag -d nombre_etiqueta
```

De momento no borres las que hemos creado.

Visualizar cambios

Para ver los cambios que se han realizado en el código usamos la orden `git diff`. La orden sin especificar nada más, mostrará los cambios que no han sido añadidos aún, es decir, todos los cambios que se han hecho antes de usar la orden `git add`. Después se puede indicar un parámetro y dará los cambios entre la versión indicada y el estado actual. O para comparar dos versiones entre sí, se indica la más antigua y la más nueva. Ejemplo:

```
$ git diff v1-beta v1
diff --git a/hola.php b/hola.php
index a31e01f..25a35c0 100644
--- a/hola.php
+++ b/hola.php
@@ -1,3 +1,4 @@
 <?php
+// El nombre por defecto es Mundo
 $nombre = isset($argv[1]) ? $argv[1] : "Mundo";
 @print "Hola, {$nombre}\n";
 ?>
```

La salida del comando git diff v1-beta v1 muestra las diferencias entre dos puntos en la historia del repositorio de Git, específicamente entre las versiones "v1-beta" y "v1". Aquí está el desglose de la salida:

- diff --git a/hola.php b/hola.php: Esto indica que se está comparando el archivo hola.php entre las dos versiones. La letra "a/" indica la versión original (en este caso, la versión de "v1-beta"), y la letra "b/" indica la versión modificada (en este caso, la versión de "v1").
- index a31e01f..25a35c0 100644: Los valores del índice (hash) para las dos versiones que están siendo comparadas. En este caso, el commit original (v1-beta) tiene el hash

a31e01f y el commit modificado (v1) tiene el hash 25a35c0. El número 100644 es el modo de archivo.

- --- a/hola.php: Indica que el archivo original (hola.php en la versión de "v1-beta") tiene el contenido que sigue.
- +++ b/hola.php: Indica que el archivo modificado (hola.php en la versión de "v1") tiene el contenido que sigue.
- @@ -1,3 +1,4 @@: Muestra la sección modificada del archivo. En este caso, indica que desde la línea 1 hasta la línea 3 en la versión original y desde la línea 1 hasta la línea 4 en la versión modificada hay diferencias.
- <?php: Este es el contenido original de la línea 1.
- +// El nombre por defecto es Mundo: Esta línea fue agregada en la versión modificada.
- $nombre = isset($argv[1]) ? $argv[1] : "Mundo";: Esta línea está presente tanto en la versión original como en la versión modificada, por lo que no hay cambios aquí.
- @print "Hola, {$nombre}\n";: Esta línea está presente tanto en la versión original como en la versión modificada.

En resumen, la salida indica que se ha agregado un comentario en la línea 2 de la versión "v1", y la diferencia en el contenido de la línea 3 se debe a la adición del comentario en la versión "v1".

No borres lo que hemos hecho hasta aquí. Seguiremos con este ejemplo en el siguiente apartado.