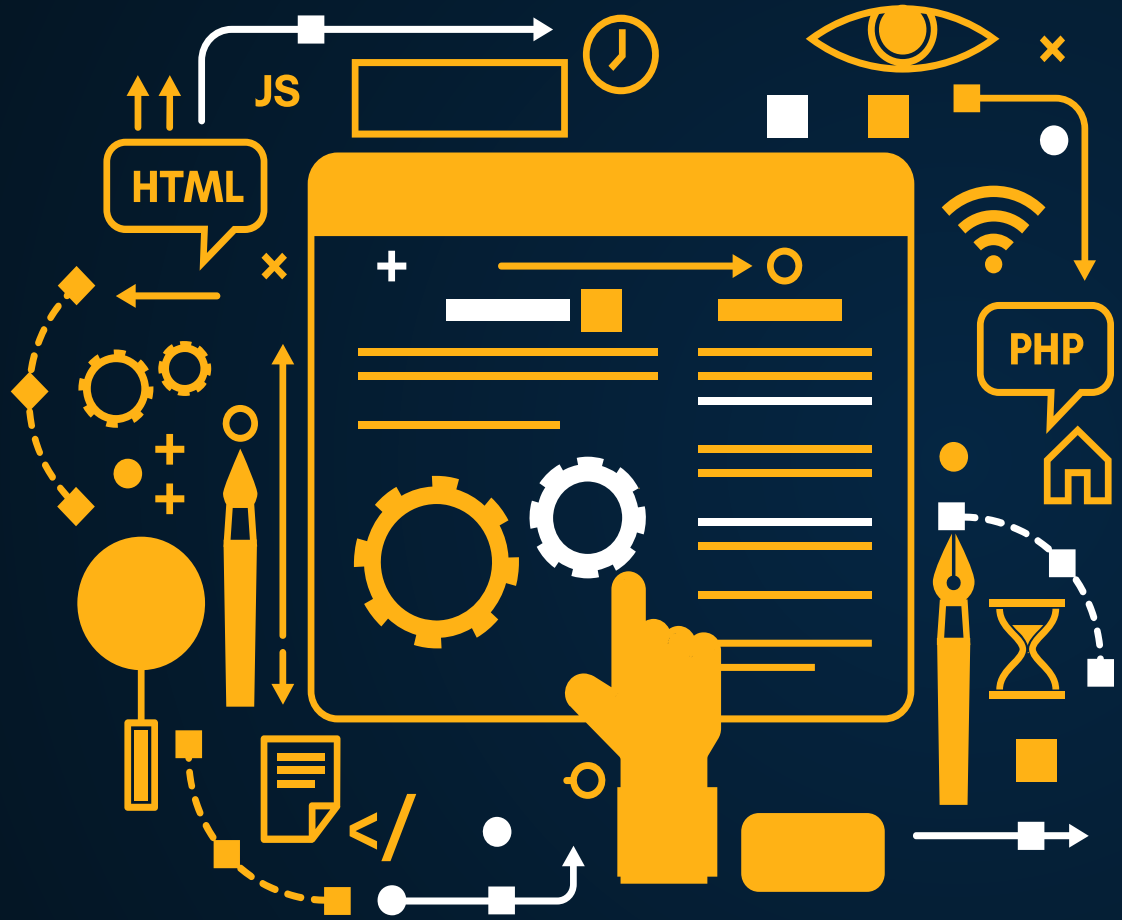


ENTORNOS DE DESARROLLO

UD4. Sistemas de control de versiones

Contenidos

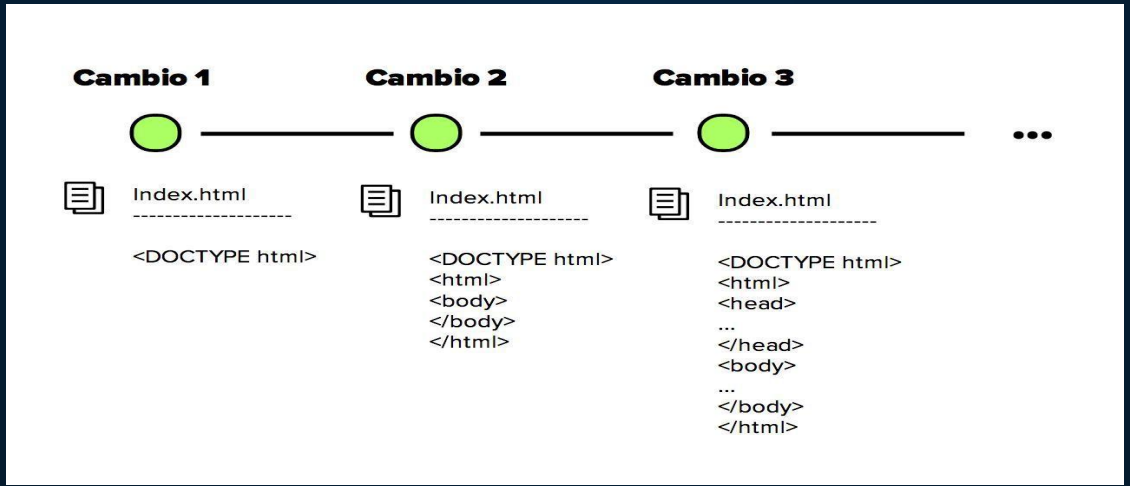
- Sistemas de Control de Versiones.
- Estructura de las herramientas de Control de Versiones.
- Repositorio.
- Herramientas de control de versiones: GIT.
- Control de versiones en la nube: GitHub.
- Clientes de control de versiones integrados en el Entorno de Desarrollo.

Historia

Paso de ficheros
Subversion
Git & Mercurial (HG)

Sistema de control de versiones

Un sistema de Control de versiones guarda los cambios en un fichero o conjunto de ficheros durante el tiempo que se trabaja con ellos de modo que es posible recuperar una versión específica con posterioridad



Estructura del sistema de control de versiones

El sistema de control de versiones está formado por un conjunto de componentes:
Repositorio (repository): directorio donde se almacenan los datos de los proyectos.
Módulo (module): conjunto de directorios/archivos del repositorio que pertenecen a un proyecto común
Revisión (revision): versión parcial o cambios en los archivos o repositorio completo. La última versión recibe el nombre de HEAD
Etiqueta (tag): título que se añade a una versión
Rama (branch): bifurcación con revisiones paralelas de un módulo para efectuar cambios sin tocar la evolución principal. Se usan para pruebas o mantener los cambios en versiones antiguas.

Estado de los archivos

Git tiene tres estados principales en los que se pueden encontrar tus archivos: confirmado (**committed**), modificado (**modified**), y preparado (**staged**).

Confirmado: los datos están almacenados de manera segura en tu base de datos local.
Modificado: has modificado el archivo pero todavía no lo has confirmado a tu base de datos.
Preparado: has marcado la versión actual de un archivo modificado para que se añada a tu próxima confirmación.

Repositorio - Secciones

Un proyecto GIT tiene las siguientes secciones:

El directorio de Git (Git directory o repositorio): donde se almacenan los metadatos y la base de datos de objetos para tu proyecto. Es lo que se copia cuando clonas un repositorio desde otra computadora.

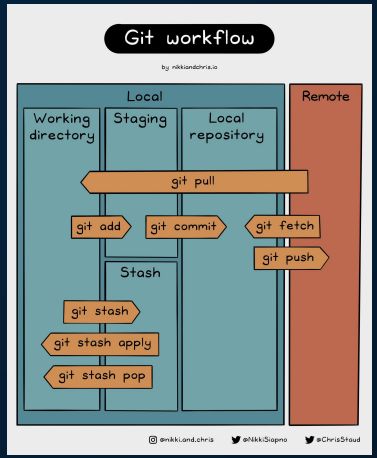
El directorio de trabajo (working directory) es una copia de una versión del proyecto. Estos archivos se sacan de la base de datos comprimida en el directorio de Git, y se colocan en disco para que los puedas usar/modificar.

El área de preparación (staging area) es un archivo en tu directorio de Git que almacena información de lo que se añade a tu próxima confirmación. También se denomina índice (“**index**”).

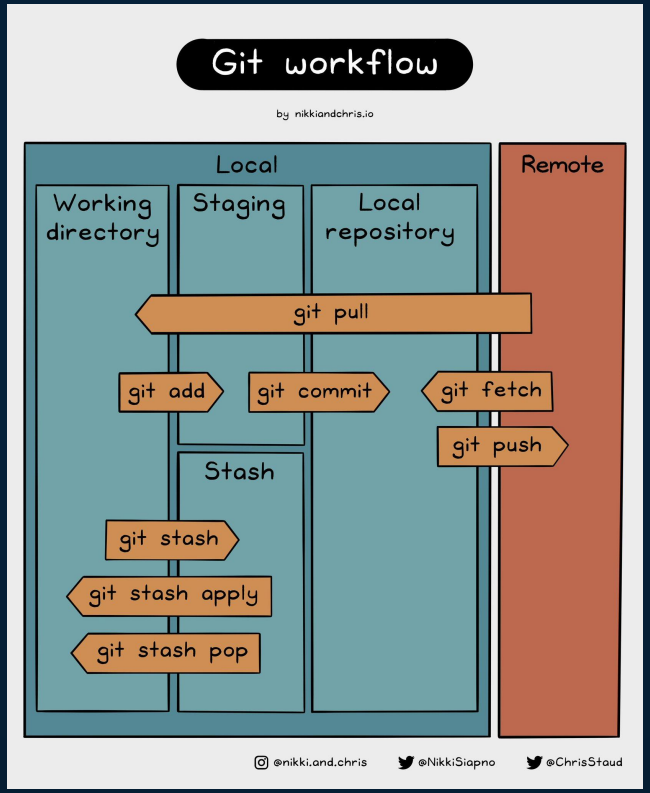
Flujo de trabajo entre secciones

El flujo de trabajo entre las secciones es el siguiente:

- Copiamos (clonamos) el proyecto desde Git Directory a Working Directory
- Modificamos archivos y los añadimos a Staging Area
- Confirmamos los archivos de Staging Area para almacenarlos en Git Directory
- Remoto:
  - Subir cambios
  - Descargar cambios
  - Comprobar si existen cambios



Flujo de trabajo entre secciones



Instalación y configuración de Git

- Instalamos GIT desde el terminal con el siguiente comando:

```
sudo apt install git
```

La herramienta git config realiza la configuración de Git que se puede almacenar en los siguientes ficheros:

1. **/etc/gitconfig** se aplica a todos los usuarios del sistema y todos sus repositorios. Se lee y escribe específicamente en este archivo con la opción --system.
2. **~/.gitconfig o ~/.config/git/config** es específico del usuario. Se lee y escribe específicamente en este archivo con la opción --global.
3. **.git/config** en el directorio de Git es específico del repositorio actual.

Cada nivel sobrescribe los valores del nivel anterior, esto es, los valores de .git/config tienen preferencia sobre los de /etc/gitconfig.

Configuración de Git

La configuración básica pasa por establecer el nombre de usuario y el email (se indica global para que sea la configuración para todo el sistema):

```
git config --global user.name "Antonio Calabuig"
git config --global user.email "antonio.calabuig@ieslasenia.org"
```

Opcionalmente puedes configurar otras opciones como el editor de texto a usar (si no se especifica, éste podría ser vim, por defecto en muchos sistemas):

```
git config --global core.editor (nano|emacs|vim)
```

Puedes comprobar la configuración con la opción list:

```
git config --list
```

O bien indicando la clave específica para obtener el valor configurado:

```
git config user.name
```

Repositorio de Git

Podemos obtener un proyecto Git de dos formas:

- **creando** `(git init)` un repositorio en un directorio existente
- **clonando** `(git clone)` un repositorio existente. Capitulo 2.1 de git-scm

En ambos casos deberemos tener creada la carpeta en la que queramos alojar el proyecto.

Además, también en ambas opciones debemos indicar a GIT que dicho directorio va a ser nuestro Working Directory ejecutando la sentencia git init.

A continuación veremos el proceso de crear un directorio GIT vacío y trabajar con los archivos que creemos en él. Más tarde veremos la opción de clonar un repositorio existente con git clone.

Primeros pasos repositorio GIT

Comenzaremos creando la carpeta de nuestro proyecto, en mi caso “pruebagit”:

```
mkdir pruebagit
```

Mostramos que la carpeta no contiene nada todavía:

```
~/Sites/ed/pruebagit ls -la
total 0
drwxr-xr-x  2 imacbuig  staff  64 Jan 10 12:47 .
drwxr-xr-x  3 imacbuig  staff  96 Jan 10 12:47 ..
```

Indicamos que esa carpeta va a ser nuestro Working Directory de GIT: `git init`

Observamos que en la carpeta se ha creado un directorio ,git para el seguimiento de los archivos:

```
~/Sites/ed/pruebagit git init
Initialized empty Git repository in /Users/imacbuig/Sites/ed/pruebagit/.git/
~/Sites/ed/pruebagit git master ls -la
total 0
drwxr-xr-x  3 imacbuig  staff   96 Jan 10 12:48 .
drwxr-xr-x  3 imacbuig  staff   96 Jan 10 12:47 ..
drwxr-xr-x  9 imacbuig  staff  288 Jan 10 12:48 .git
~/Sites/ed/pruebagit git master
```

Primeros pasos repositorio GIT

Creamos los primeros ficheros file1.txt y file2.txt. Comprobamos la situación de los ficheros:

`git status`

```
~/Sites/ed/pruebagit  master  touch file1.txt file2.txt
~/Sites/ed/pruebagit  master ?2  git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        file1.txt
        file2.txt

nothing added to commit but untracked files present (use "git add" to track)
~/Sites/ed/pruebagit  master ?2
```

Da un aviso que hay ficheros sin seguimiento de versiones, por lo que los añadiremos al seguimiento: **`git add`**

Observamos con **`git status`** que se han añadido al área de Staging pero nos avisa que no se ha hecho commit, se quedan preparados:

```
~/Sites/ed/pruebagit  master ?2  git add file1.txt file2.txt
~/Sites/ed/pruebagit  master +2  git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   file1.txt
        new file:   file2.txt

~/Sites/ed/pruebagit  master +2
```

Nota: podemos añadir ficheros individuales con git add fich1.txt, en el ejemplo hemos puesto el directorio actual (.)

Primeros pasos repositorio GIT

Confirmamos los cambios en los ficheros ya preparados para almacenarlos en el directorio git o repositorio. Se añade un mensaje o etiqueta para identificar los cambios confirmados (para realizar el commit, se debe tener configurado el usuario):

```
git commit -m "Primeros cambios en el proyecto"
```

```
~/Sites/ed/pruebagit  master +2  git commit -m "Primeros cambios en el proyecto"
[master (root-commit) 970fef0] Primeros cambios en el proyecto
 2 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 file1.txt
 create mode 100644 file2.txt
~/Sites/ed/pruebagit  master
```

Se puede ver los cambios realizados y el hash del commit. Con `git log [-p]` obtenemos más información:

`git log -p` nos muestra:

- ➔ El hash del commit
- ➔ El autor de los cambios
- ➔ La diferencia de los ficheros actuales (+++) con las versiones anteriores (---)

```
commit 970fef0b6b04843f2e8c73ea57c5020ebdb4a94c (HEAD -> master)
Author: Antonio Calabuig Puigvert <antonio.calabuig@gmail.com>
Date:   Tue Jan 10 12:56:54 2023 +0100

    Primeros cambios en el proyecto

diff --git a/file1.txt b/file1.txt
new file mode 100644
index 0000000..e69de29
diff --git a/file2.txt b/file2.txt
new file mode 100644
index 0000000..e69de29
```

Primeros pasos repositorio GIT

Creamos otros dos ficheros file3.txt y file4.txt, los añadimos y los guardamos con el commit “Segunda revisión”:

```
~/Sites/ed/pruebagit  master  touch file3.txt file4.txt
~/Sites/ed/pruebagit  master ?2  git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        file3.txt
        file4.txt

nothing added to commit but untracked files present (use "git add" to track)
~/Sites/ed/pruebagit  master ?2  git add file3.txt file4.txt
~/Sites/ed/pruebagit  master +2  git commit -m "Segunda revision"
[master 938fa1e] Segunda revision
 2 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 file3.txt
 create mode 100644 file4.txt
~/Sites/ed/pruebagit  master
```

```
commit 938fa1ebfc95894ab8cc646f6364f316835901d3 (HEAD -> master)
Author: Antonio Calabuig Puigvert <antonio.calabuig@gmail.com>
Date:   Tue Jan 10 14:31:14 2023 +0100

    Segunda revision

 file3.txt | 0
 file4.txt | 0
 2 files changed, 0 insertions(+), 0 deletions(-)

commit 970fef0b6b04843f2e8c73ea57c5020ebdb4a94c
Author: Antonio Calabuig Puigvert <antonio.calabuig@gmail.com>
Date:   Tue Jan 10 12:56:54 2023 +0100

    Primeros cambios en el proyecto

 file1.txt | 0
 file2.txt | 0
 2 files changed, 0 insertions(+), 0 deletions(-)
```

Experimentando con ramas

Una rama permite trabajar en un Working Directory distinto al principal (master) de modo que se pueden realizar desarrollos en paralelo. Para el ejemplo crearemos una rama nueva_rama con una carpeta demo y dentro un fich5.txt

```
~/Sites/ed/pruebagit on git main git branch nueva_rama
~/Sites/ed/pruebagit on git main git checkout nueva_rama
Switched to branch 'nueva_rama'
~/Sites/ed/pruebagit on git nueva_rama mkdir demo
~/Sites/ed/pruebagit on git nueva_rama cd demo
~/S/ed/pruebagit/demo on git nueva_rama touch file5.txt
~/S/ed/pruebagit/demo on git nueva_rama ?1
```

Realizamos el commit del fich5.txt en nueva_rama:

```
~/Sites/ed/pruebagit on git nueva_rama ?1 git status
On branch nueva_rama
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        demo/

nothing added to commit but untracked files present (use "git add" to track)
~/Sites/ed/pruebagit on git nueva_rama ?1 git add demo
~/Sites/ed/pruebagit on git nueva_rama +1 git status
On branch nueva_rama
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   demo/file5.txt

~/Sites/ed/pruebagit on git nueva_rama +1 git commit -m "Tercera revision"
[nueva_rama 42860b3] Tercera revision
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 demo/file5.txt
```

```
git branch nueva_rama
```

```
git checkout -b nueva_rama
```

Volver a una rama por su nombre:

```
git checkout main|master
```

Experimentando con ramas

Regresamos a master y fusionamos con nueva_rama:

```
~/Sites/ed/pruebagit on git nueva_rama git checkout main
Switched to branch 'main'
~/Sites/ed/pruebagit on git main git merge nueva_rama
Updating 01da686..42860b3
Fast-forward
 demo/file5.txt | 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 demo/file5.txt
~/Sites/ed/pruebagit on git main
```

Borramos la rama:

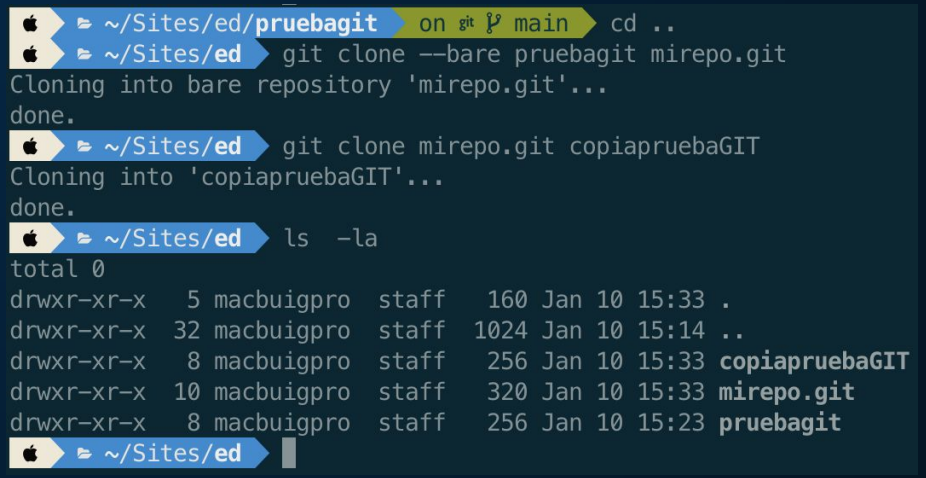
```
~/Sites/ed/pruebagit on git main git branch -d nueva_rama
Deleted branch nueva_rama (was 42860b3).
```

Clonar repositorios

Vamos a clonar un repositorio en local en otra carpeta **copiapruebaGIT**. Para ello salimos de nuestra carpeta y creamos un repositorio vacío con la copia del de **pruebagit** (mi repo.git) que hará de servidor local y lo clonamos en **PRUEBAGIT**:

```
git clone --bare pruebagit mirepo.git
```

```
git clone mirepo.git copiapruebaGIT
```



Clonar repositorios

Creamos y confirmamos el file6.txt en **copiapruebaGIT**.

Con git status nos avisa que hay cambios respecto a master.

```
~/Sites/ed cd copiapruebaGIT
~/Sites/ed/copiapruebaGIT on git main touch file6.txt
~/Sites/ed/copiapruebaGIT on git main ?1 git add file6.txt
~/Sites/ed/copiapruebaGIT on git main +1 git commit -m "Agregado fichero6 desde copiapruebaGIT"
[main 76a1ac2] Agregado fichero6 desde copiapruebaGIT
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 file6.txt
~/Sites/ed/copiapruebaGIT on git main ↑1 git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
~/Sites/ed/copiapruebaGIT on git main ↑1
```

Enmendar un commit (fichero olvidado)

Creamos otra_rama para crear file7.txt y confirmarlo:

```
~/Sites/ed/copiapruebaGIT on git main ↑1 git branch otra_rama
~/Sites/ed/copiapruebaGIT on git main ↑1 git checkout otra_rama
Switched to branch 'otra_rama'
~/Sites/ed/copiapruebaGIT on git otra_rama touch file7.txt
~/S/ed/copiapruebaGIT on git otra_rama ?1 git add file7.txt
~/S/ed/copiapruebaGIT on git otra_rama +1 git commit -m "Agregado fichero7 con otra_rama"
[otra_rama ef461bd] Agregado fichero7 con otra_rama
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 file7.txt
~/Sites/ed/copiapruebaGIT on git otra_rama
```

Una vez confirmado, recordamos que nos falta añadir un fichero file8.txt que creamos y añadimos a staging, ahora enmendamos el commit para añadirlo: **`git commit --amend`**

```
~/Sites/ed/copiapruebaGIT on git otra_rama touch file8.txt
~/S/ed/copiapruebaGIT on git otra_rama ?1 git add file8.txt
~/S/ed/copiapruebaGIT on git otra_rama +1 git commit --amend
[otra_rama 01ce10b] Agregado fichero7 con otra_rama y también el fichero8 que se nos había olvidado
 Date: Tue Jan 10 15:43:21 2023 +0100
 2 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 file7.txt
 create mode 100644 file8.txt
```

Deshacer cambios

Si queremos recuperar cambios desde repositorio a Working Directory:

- `git checkout -- .` (todos los ficheros)
- `git checkout HEAD fich1.txt` (el fichero llamado fich1.txt del último commit)
- `git checkout HEAD~N` (los ficheros de N commit antes del último)

Si queremos recuperar cambios desde Staging Area a Working Directory:

- `git revert HashCommitID` (indicando el hash del commit a recuperar)
- `git reset HashCommitID` (indicando el hash del commit a recuperar)
- `git reset  HEAD fich1.txt` (el fichero llamado fich1.txt del último commit)
- `git reset  --hard HEAD~N` (los ficheros de N commit antes del último. OJO que se pierden todos los datos de Staging Area y Working Directory que no estuvieran confirmados)

Mostrando cambios

Podemos necesitar comparar los cambios producidos entre commits como el último y los dos anteriores: `git diff HEAD HEAD~2`

```
diff --git a/file6.txt b/file6.txt
deleted file mode 100644
index e69de29..0000000
diff --git a/file7.txt b/file7.txt
deleted file mode 100644
index e69de29..0000000
diff --git a/file8.txt b/file8.txt
deleted file mode 100644
index e69de29..0000000
(END)
```

También podemos comparar cambios entre el Working Directory y Staging Area tras haber añadido fich9.txt: `git diff --staged`

```
diff --git a/file9.txt b/file9.txt
new file mode 100644
index 0000000..e69de29
(END)
```

```
~/Sites/ed/copiapruebaGIT on git main ↑2 touch  file9.txt
~/S/ed/copiapruebaGIT on git main ↑2 ?1 git status
On branch main
Your branch is ahead of 'origin/main' by 2 commits.
  (use "git push" to publish your local commits)

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        file9.txt

nothing added to commit but untracked files present (use "git add" to track)
~/S/ed/copiapruebaGIT on git main ↑2 ?1 git add file9.txt
~/S/ed/copiapruebaGIT on git main ↑2 +1 git diff --staged
~/S/ed/copiapruebaGIT on git main ↑2 +1 git diff --staged
~/S/ed/copiapruebaGIT on git main ↑2 +1 git commit -m "Añado fichero9"
[main 52ed0fb] Añado fichero9
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 file9.txt
```

Etiquetas y extensiones a ignorar

Podemos añadir etiquetas a nuestro proyecto para indicar la versión, por ejemplo:

`git tag -a v1.0 -m "versión 1.0"`

`git show v1.0` (Muestra los detalles de la etiqueta además del log de cambios que se ha omitido por espacio)

```
~/Sites/ed/copiapruebaGIT on git main ↑3 git tag -a v1.0 -m "versión 1.0"
~/Sites/ed/copiapruebaGIT on git main ↑3 git show v1.0
```

```
tag v1.0
Tagger: Antonio Calabuig Puigvert <antonio.calabuig@gmail.com>
Date:   Tue Jan 10 18:48:19 2023 +0100

versión 1.0

commit 52ed0fb4495f8d92804f6b246659d940804d0ec8 (HEAD -> main, tag: v1.0)
Author: Antonio Calabuig Puigvert <antonio.calabuig@gmail.com>
Date:   Tue Jan 10 18:42:57 2023 +0100

    Añado fichero9

diff --git a/file9.txt b/file9.txt
new file mode 100644
index 0000000..e69de29
(END)
```

Nota: para mantener las etiquetas se suben al servidor con:

`git push origin --tags`

Etiquetas y extensiones a ignorar

Si queremos que se ignoren archivos de determinadas extensiones, lo indicamos en el fichero .gitignore (debe ser confirmado)

```
touch .gitignore

git add .gitignore

git commit .gitignore -m "Añado el fichero .gitignore"

echo "Fichero temporal" > fich.tmp

git status
```

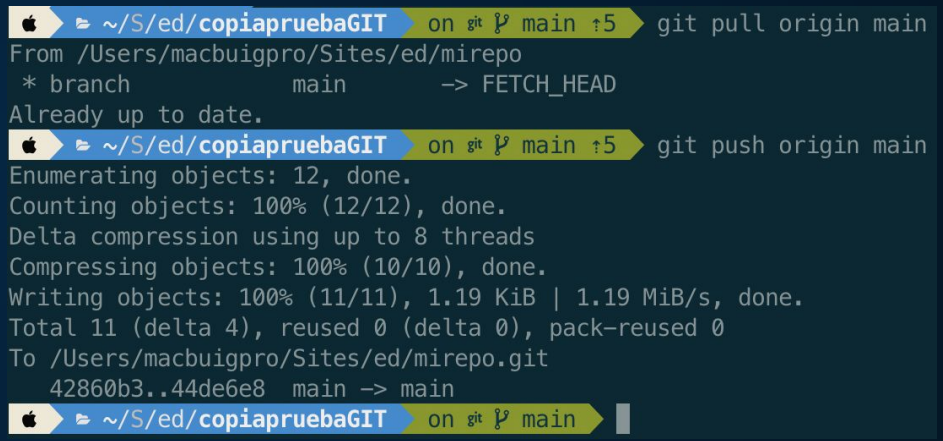
(no indica que se ha creado el fich.tmp, lo ignora)

```
~/S/ed/copiapruebaGIT on git main ↑3 touch .gitignore
~/S/ed/copiapruebaGIT on git main ↑3 ?1 git add .gitignore
~/S/ed/copiapruebaGIT on git main ↑3 +1 git commit -m "Añadido fichero .gitignore"
[main b520a75] Añadido fichero .gitignore
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 .gitignore
~/Sites/ed/copiapruebaGIT on git main ↑4 echo "Fichero tempora" > file.tmp
~/S/ed/copiapruebaGIT on git main ↑4 ?1 vim .gitignore
~/S/ed/copiapruebaGIT on git main ↑4 !1 git status
On branch main
Your branch is ahead of 'origin/main' by 4 commits.
  (use "git push" to publish your local commits)

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:   .gitignore

no changes added to commit (use "git add" and/or "git commit -a")
~/S/ed/copiapruebaGIT on git main ↑4 !1 git  commit -m "Modificado .gitignore"
```

Actualizar cambios en repositorios

Comprobamos que tenemos la última versión del repositorio (copiaPruebaGIT-`[master|main]`) y enviamos nuestros cambios al repositorio que hace de servidor (mirepo.git - origin):

```
git pull origin [master|main]

git push origin [master|main]
```



- Con `git pull origin [master|main]` traemos los posibles cambios realizados por otras personas en **mirepo.git**
- En nuestro caso no los hay, pero si hubiera, deberíamos hacer git merge para incorporar esos cambios a nuestro repositorio para fusionarlos y subir la última versión completa
- Finalmente, subimos los cambios con git push origin master

Nota: `git pull` realiza un `git fetch` y `git merge` implícitos

Clonar repositorio de repo server

Hasta ahora habíamos creado un repositorio llamado **pruebagit**. Tras trabajar con él, creamos un repositorio **mirepo.git** **sin estructura (`bare`)** donde almacenamos todo lo de **pruebagit** (es como una copia de seguridad). Clonamos **mirepo.git** a **copiapruebaGIT** y continuamos trabajando en ese **Working Directory**. Ahora, queremos sincronizar nuestro **WD pruebagit** con **mirepo.git** que nos hace de repositorio "**SERVIDOR**". Para ello, necesitamos clonar **mirepo.git** en **pruebagit**. Sin embargo, debe ser un repositorio vacío así que "borramos" el contenido de **pruebagit** antes de clonar:

- No vamos a borrarlo, lo vamos a renombrar: `mv pruebagit pruebagit.old`
- Ahora ya podemos clonar de nuevo **mirepo.git** en **pruebagit** y listamos para ver el contenido del directorio

```
git clone mirepo.git pruebagit

ls -la
```

```
~/S/ed/copiapruebaGIT on git main cd ..
~/Sites/ed ll
total 0
drwxr-xr-x  14 macbuigpro  staff   448B Jan 10 18:58 copiapruebaGIT
drwxr-xr-x  10 macbuigpro  staff   320B Jan 16 14:17 mirepo.git
drwxr-xr-x   8 macbuigpro  staff   256B Jan 10 15:23 pruebagit
~/Sites/ed mv pruebagit pruebagit.old
~/Sites/ed git clone mirepo.git pruebagit
Cloning into 'pruebagit'...
done.
```

Bibliografía

- 1.1 Getting Started - About Version Control
- Getting a Git Repository