



Anfitrionas: Hablemos de tecnología W4TT



Women for Technical Talks W4TT

<https://women4tt.blogspot.com>

#W4TT
#anfitrionasw4tt

Next Step

 **axazure**

Lola Villalobos Ortiz

Desarrolladora.Net

東京'
TOKIOTA



@lola_wizard



lolavillalobosortiz



Women for Technical Talks W4TT

<https://women4tt.blogspot.com>

#W4TT

#anfitrionasw4tt

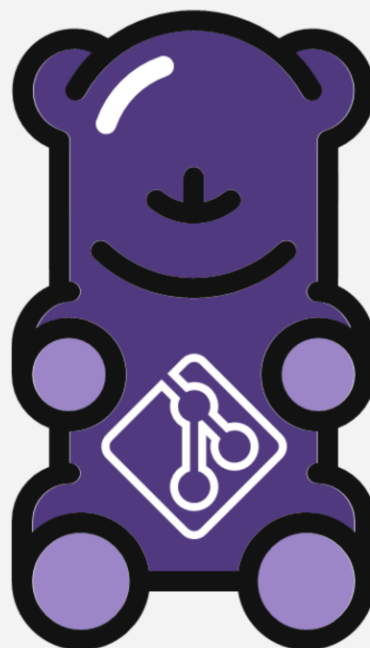
Next Step

axazure

Git for Gummies

O cómo meter cabeza en Git sin
ponerle 2 velas negras a Linus Torvalds

Lola Villalobos Ortiz



Contenido

¿Qué es Git?

Sistema Distribuido
Almacenamiento en Snapshots
Estructura de Grafo

Estructura del entorno de trabajo

Comandos principales

Crear Repositorio	Repositorio remoto
Añadir Cambios	Deshacer cambios
Trabajar con ramas	Utilidades

Herramientas que nos facilitan la vida y repositorios

¿Por qué usar Git?



Women for Technical Talks W4TT

<https://women4tt.blogspot.com>

#W4TT
#anfitrionasw4tt

Next Step

Axazure

¿Qué es Git?

- Sistema de control de versiones (CVS)
- Linus Torvalds (2005)
- Cómo almacena la información:
 - Sistema Distribuido (DCVS).
 - Almacenamiento de *snapshots*.
 - Estructura de grafo.



Women for Technical Talks W4TT

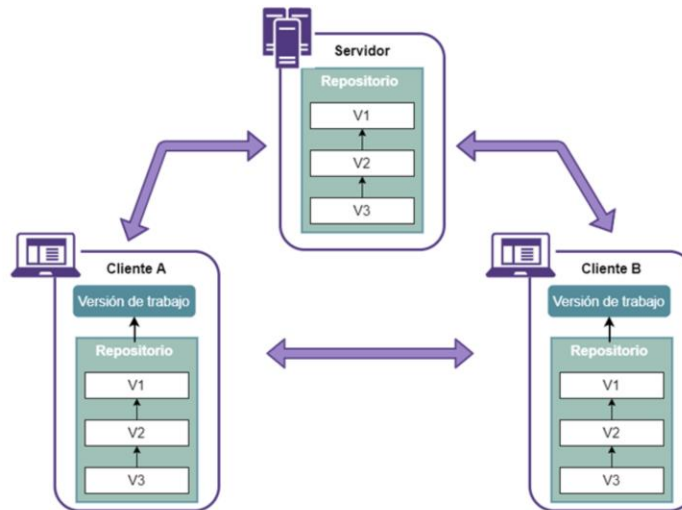
<https://women4tt.blogspot.com>

#W4TT
#anfitrionasw4tt

Next Step

axazure

¿Qué es Git? -> Sistema Distribuido



Women for Technical Talks W4TT

<https://women4tt.blogspot.com>

#W4TT
#anfitrionasw4tt

Next Step



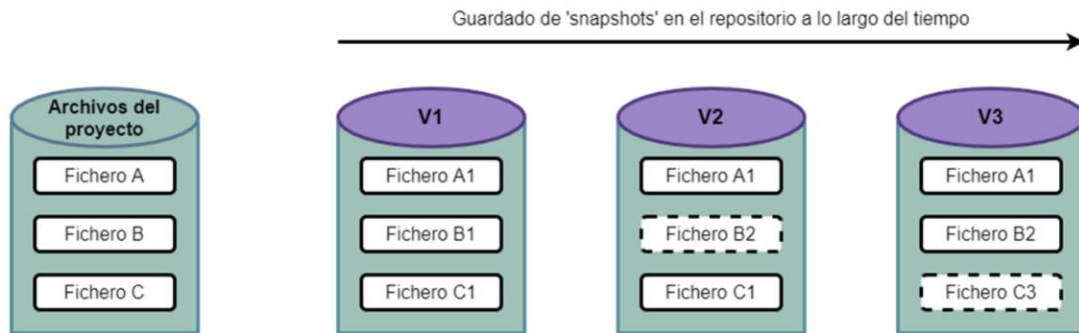
Entre sus principales características están que se trata de un sistema de control de versiones distribuido, es decir, cada cliente se descarga una copia completa del repositorio (la última versión del código más todo su historio).

Esto aporta seguridad, porque no existe el riesgo de perder una única copia centralizada.

Al tener el repositorio completo casi todas las operaciones se pueden hacer en local, sin necesidad de conexión.

Y se agilizan muchas operaciones.

¿Qué es Git? -> Almacenamiento de Snapshots



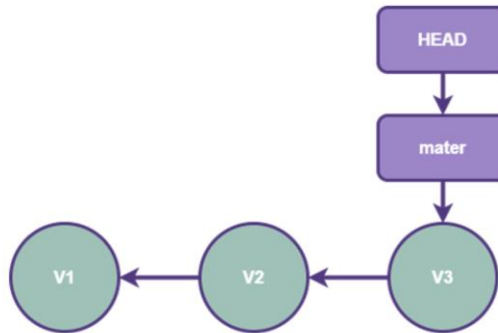
Git almacena los datos como si fueran 'snapshots' o 'instantáneas' con el estado actual de los archivos. Sería el equivalente a un commit.

Es decir, para la versión que vamos a guardar en el repositorio, guarda el conjunto de archivos que lo conforman en ese momento, tanto los que tienen cambios como los que no.

Aunque por razones de eficiencia, en el caso de los archivos que no han sido modificados, lo que se guarda es una referencia a la versión anterior, en lugar del archivo al completo.

Por último, hay que mencionar que, dentro de la estructura interna de los snapshot, aparte de incluir el conjunto de cambios y otra información como autoría, comentarios, etc. incluye una clave hash asociada (SHA-1) generada a partir de su contenido y que, aparte de identificarlo de forma única, permite garantizar la integridad de los datos almacenados en el repositorio, de modo que es imposible que se pierdan o corrompan datos sin que el sistema lo detecte.

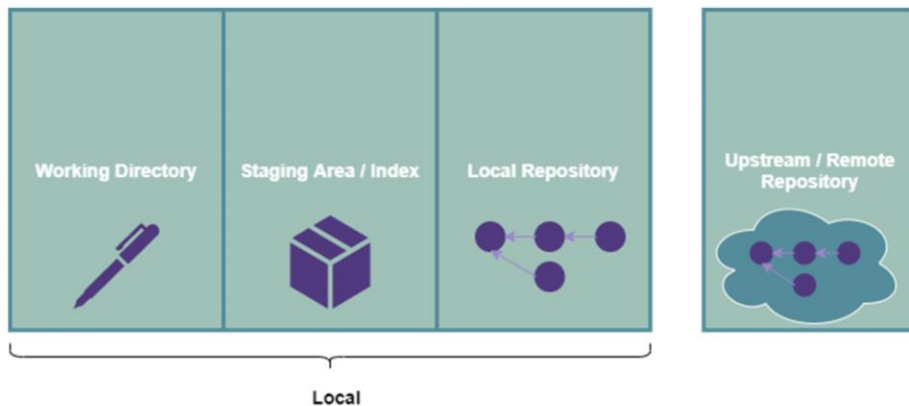
¿Qué es Git? -> Estructura de Grafo



Por último, el histórico de cambios se almacena en forma de grafo, es decir, la información sobre los cambios realizados incluye también la referencia a los cambios anteriores, de forma que podemos seguir la traza de los cambios .

Respecto a las referencias a master y HEAD, se trata de punteros que nos permiten ubicarnos dentro del grafo. Master es el nombre que se le da por defecto a la rama principal del grafo y su puntero siempre hace referencia al último commit de la rama. En el caso de HEAD, se trata del puntero que apunta a la rama en la que nos encontramos trabajando ahora mismo, en este caso, master.

Estructura del entorno de trabajo



Hasta ahora hemos visto cómo almacena Git internamente los cambios en nuestros ficheros.

Sin embargo, Git también nos 'impone' una forma específica de trabajar con el repositorio y determina distintos estados para los ficheros.

En local se distinguen tres áreas distintas o estados en los que se puede encontrar un archivo:

- Por un lado, está el Working Directory, que engloba a todos los archivos con los que estamos trabajando, siempre dentro de la correspondiente carpeta del repositorio en local, ya sean nuevos o modificados. Como su propio nombre indica, es el área de trabajo.
- Por otro está el Staging Area o Index. Cualquier fichero nuevo que queramos añadir al repositorio o cambio que deseemos hacer, debe pasar del Working Directory a esta área. Se puede decir que es un área de preparación, donde especifico qué quiero que vaya en mi siguiente subida al repositorio. Si estáis familiarizados con TFS sería como lo opuesto a excluir un fichero del conjunto de cambios a subir, en este caso, de todos tus cambios, tienes que indicar cuáles quieres subir al repositorio.
- La última área a nivel local sería el Repositorio Local, aquí es donde ya subiría mis cambios y quedarían registrados en el histórico (donde se haría commit), sólo que todavía serían visibles sólo para mí, dado que siguen en local.
- Y por último estaría el repositorio remoto. En realidad, se trata simplemente de otro repositorio con el que sincronizar mi repo local, que puede estar en mi

máquina o en otra ajena, y básicamente es donde mis cambios serían visibles para el resto de clientes de Git. Digamos que, si otro cliente actualiza su código con los últimos cambios subidos, consulta este repositorio y copia todo lo que haya en él a su repositorio local y actualiza su Working Directory con el contenido de dichos cambios.

Metiendo las manos en la masa

Crear repositorio

- Intro a .gitignore

Añadir cambios

- Index
- Repositorio local

Deshacer cambios

- Deshacer cambios en Working Directory/Index
- Deshacer en Repositorio Local

Trabajar con ramas

- Crear rama
- Cambiar de rama
- Merge

Repositorio remoto

- Crear/enlazar repositorio remoto
- Subir/bajar cambios

Utilidades

- Comprobar estado
- Log de cambios
- Diferencias



Women for Technical Talks W4TT

<https://women4tt.blogspot.com>

#W4TT
#anfitrionasw4tt

Next Step

Axazure

Crear repositorio

git init

git status

```
Command Prompt
Microsoft Windows [Version 10.0.19041.264]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\LolaVillalobos>cd c:\PruebasGit

c:\PruebasGit>git init
Initialized empty Git repository in c:/PruebasGit/.git/

c:\PruebasGit>git status
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)

c:\PruebasGit>
```



Women for Technical Talks W4TT

<https://women4tt.blogspot.com>

#W4TT
#anfitrionasw4tt

Next Step

Axazure

El comando git init nos permite crear un repositorio desde en nuestra máquina. Si nos vamos al directorio de carpetas, se puede ver que lo único que ha creado es una carpeta oculta .git, que contendrá la información con la que gestionar el repositorio. Aparte de esa carpeta, dentro podemos incluir los archivos que deseemos incorporar a nuestro repositorio. También está el comando git status, que nos resultará muy útil para averiguar el estado actual de nuestro proyecto, qué cambios hay y dónde se encuentran.

Crear repositorio

.gitignore

```
.gitignore - Notepad
File Edit Format View Help
# Ficheros cache/opciones Visual Studio
.vs/

# Visual Studio Code
.vscode

# Ficheros de usuario
*.suo
*.user

# Directorios y archivos resultantes tras compilar
[Dd]ebug/
[Rr]elease/
[Rr]eleases/
x64/
x86/
build/
bld/
[Bb]in/
[Oo]bj/
[Oo]ut/
```



Women for Technical Talks W4TT

<https://women4tt.blogspot.com>

#W4TT

#anfitrionasw4tt

Next Step

Axazure

Un archivo que resulta útil para gestionar los archivos de mi proyecto es .gitignore. Este fichero, mediante expresiones regulares, nos permitirá indicarle a git que ignore ciertos ficheros de nuestro proyecto (por ejemplo, los resultados de las compilaciones). En este ejemplo ignoro las carpetas obj, debug, release... para no cargar el repositorio con ficheros que no aportan nada y que sólo añadirán ruido a los cambios. En la red podéis encontrar muchos ejemplos típicos para cada tipo de proyecto (C++, C#, Java...).

Añadir cambios -> Index

git add .

git add <file>

```
Command Prompt
c:\PruebasGit>git status
On branch master

No commits yet

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

        .gitignore
        HelloWorld/

nothing added to commit but untracked files present (use "git add" to track)

c:\PruebasGit>git add .
c:\PruebasGit>git status
On branch master

No commits yet

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

        new file:   .gitignore
        new file:   HelloWorld/HelloWorld.sln
        new file:   HelloWorld/HelloWorld/HelloWorld.csproj
        new file:   HelloWorld/HelloWorld/Program.cs
```



Una vez he añadido mi proyecto al repositorio, los ficheros del proyecto que he añadido aparecen como untracked, esto significa que Git todavía no los reconoce para hacerles el seguimiento.

Podéis ver que entre paréntesis git nos hace recomendaciones de los siguientes pasos que podemos dar, en este caso, nos sugiere que añadamos los ficheros al Staging Area o Index para poder 'comitearlos', o sea, llevarlos al repositorio local. Para añadirlos sin tener que ir fichero por fichero, ejecuto git add .

Si comprobamos ahora el estado, los nuevos ficheros figuran como listos para ser comiteados, es decir, ya están incluidos en el Index/Staging Area. Como podéis ver en el listado, al incluir .gitignore, sólo se están subiendo los ficheros que no están incluidos en éste, es decir, está ignorando las carpetas de compilación y ficheros de usuario.

Añadir cambios -> Repositorio local

**git commit -m
<mensaje>**

```
Command Prompt
c:\PruebasGit>git commit -m "Primer commit, subo proyecto HelloWorld"
[master (root-commit) b10ef12] Primer commit, subo proyecto HelloWorld
4 files changed, 66 insertions(+)
create mode 100644 .gitignore
create mode 100644 HelloWorld/HelloWorld.sln
create mode 100644 HelloWorld/HelloWorld/HelloWorld.csproj
create mode 100644 HelloWorld/HelloWorld/Program.cs

c:\PruebasGit>git status
On branch master
nothing to commit, working tree clean

c:\PruebasGit>
```



Women for Technical Talks W4TT

<https://women4tt.blogspot.com>

#W4TT

#anfitrionasw4tt

Next Step

Axazure

Una vez añadidos al Staging Area/Index, podemos ya subirlos al repositorio local con git commit.

Si os fijáis, una vez subido al repositorio local, ya no tenemos cambios pendientes.

Acabamos de subir nuestro primer proyecto a git.

Si queréis comprobar el listado de cambios realizado hasta ahora, podéis usar git log.

Deshacer cambios -> Eliminar del Index

git add .

git diff

```
Command Prompt

C:\PruebasGit>git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   HelloWorld/HelloWorld/Program.cs

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

C:\PruebasGit>git diff
diff --git a/HelloWorld/HelloWorld/Program.cs b/HelloWorld/HelloWorld/Program.cs
index 8168c80..1fb6e6f 100644
--- a/HelloWorld/HelloWorld/Program.cs
+++ b/HelloWorld/HelloWorld/Program.cs
@@ -6,7 +6,7 @@ namespace HelloWorld
     {
         static void Main(string[] args)
         {
             Console.WriteLine("Hello World!");
             Console.WriteLine("Bye World!");
         }
     }
 }

C:\PruebasGit>git add .

C:\PruebasGit>git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   HelloWorld/HelloWorld/Program.cs
```



Women for Technical Talks W4TT

<https://women4tt.blogspot.com>

#W4TT
#anfitrionasw4tt

Next Step



Ya tenemos nuestro proyecto añadido a Git, de modo que cualquier cambio que hagamos en los ficheros será reconocido por Git como una modificación en un fichero ya existente.

Podemos comprobar los cambios realizados con git diff.

Deshacer cambios -> Eliminar del index

git reset HEAD

Warning

git reset -- hard HEAD

```
Command Prompt
c:\PruebasGit>git reset HEAD
Unstaged changes after reset:
M   HelloWorld/HelloWorld/Program.cs

c:\PruebasGit>git status
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   HelloWorld/HelloWorld/Program.cs

no changes added to commit (use "git add" and/or "git commit -a")
c:\PruebasGit>
```



Women for Technical Talks W4TT

<https://women4tt.blogspot.com>

#W4TT
#anfitrionasw4tt

Next Step

Axazure

Vamos a añadir estos cambios al Index igual que hicimos anteriormente. Pero vamos a suponer que nos damos cuenta de que no queremos llevar estos cambios al repositorio, que nos hemos equivocado. Hay varios comandos para deshacer esto, pero el más básico sería git reset.

El comando sería git reset seguido del commit que queremos llevar a nuestro Index, en este caso hemos especificado el último commit (dado que HEAD apunta al último commit realizado).

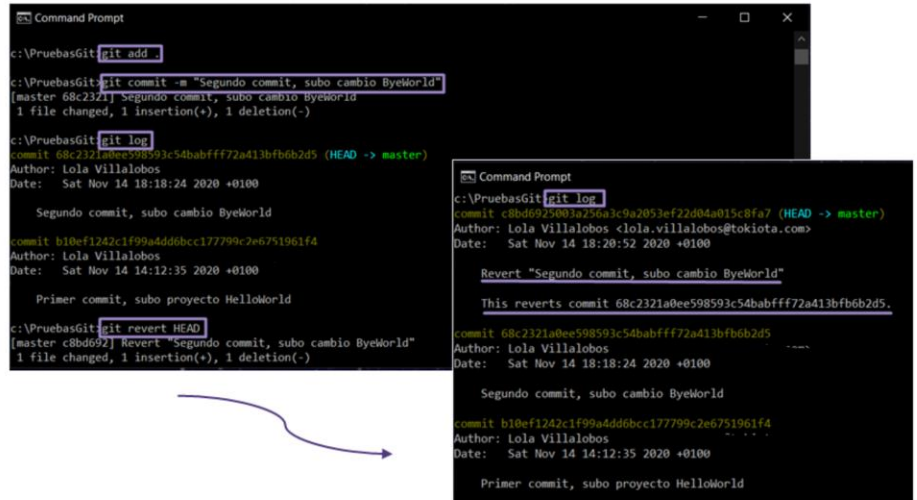
Este comando mueve el puntero HEAD al commit especificado (en este caso se queda donde está) y copia su contenido al Index, en este caso, al especificar el último commit, sería como sacar los cambios del Index.

También podemos ir un paso más allá y usar la opción --hard, con lo que no sólo se eliminan los cambios del Index, sino que también los perdemos de nuestro Working Directory, por lo que hay que tener cuidado con esta opción, ya que perderíamos los cambios sin posibilidad de recuperación.

Deshacer cambios -> Eliminar del repositorio local

git revert HEAD

git log



```
c:\PruebasGit>git add
c:\PruebasGit>git commit -m "Segundo commit, subo cambio ByeWorld"
[master 68c2321] Segundo commit, subo cambio ByeWorld
1 file changed, 1 insertion(+), 1 deletion(-)
c:\PruebasGit>git log
commit 68c2321a0ee598593c54babfff72a413fb6b2d5 (HEAD -> master)
Author: Lola Villalobos
Date: Sat Nov 14 18:18:24 2020 +0100

    Segundo commit, subo cambio ByeWorld

commit b10ef1242c1f99a4dd6bcc177799c2e6751961f4
Author: Lola Villalobos
Date: Sat Nov 14 14:12:35 2020 +0100

    Primer commit, subo proyecto HelloWorld

c:\PruebasGit>git revert HEAD
[master c8bd092] Revert "Segundo commit, subo cambio ByeWorld"
1 file changed, 1 insertion(+), 1 deletion(-)

c:\PruebasGit>git log
commit c8bd0925001a250a3c9a2053ef22d04a015c8fa7 (HEAD -> master)
Author: Lola Villalobos <lola.villalobos@tokiota.com>
Date: Sat Nov 14 18:20:52 2020 +0100

    Revert "Segundo commit, subo cambio ByeWorld"

    This reverts commit 68c2321a0ee598593c54babfff72a413fb6b2d5.

commit 68c2321a0ee598593c54babfff72a413fb6b2d5
Author: Lola Villalobos
Date: Sat Nov 14 18:18:24 2020 +0100

    Segundo commit, subo cambio ByeWorld

commit b10ef1242c1f99a4dd6bcc177799c2e6751961f4
Author: Lola Villalobos
Date: Sat Nov 14 14:12:35 2020 +0100

    Primer commit, subo proyecto HelloWorld
```



Women for Technical Talks W4TT

<https://women4tt.blogspot.com>

#W4TT
#anfitrionasw4tt

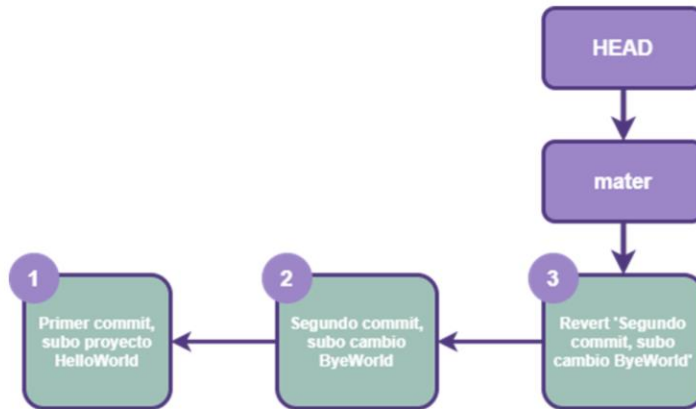
Next Step

QXQZURE

Vamos a suponer que no queremos los últimos cambios en nuestro repositorio. En este caso, una vez están en el repositorio local, podemos usar `git revert HEAD` para deshacer el último commit (al que apunta HEAD).

Para comprobar los commits que llevamos en nuestro repositorio, se puede usar la operación `git log`, con el que saldrá un listado.

Trabajar con ramas -> Estado actual



Otra de las operaciones que nos permite hacer git es la creación de ramas. Estas no dejan de ser un conjunto de commits igual que la rama principal con la que estamos trabajando, y normalmente se crean cuando se realiza algún desarrollo que no nos interesa que interfiera en la rama principal o viceversa.

Con este diagrama quiero representar el estado actual de la única rama con la que estamos trabajando, para ver que la creación de una rama no difiere tanto del esquema actual. Aquí están representados los 3 commits que llevamos hechos hasta ahora y los punteros de nuestra rama (master) y el HEAD

Trabajar con ramas -> Crear rama y moverse a ella

```
git branch <nombre_rama>
```

```
Command Prompt
c:\PruebasGit>git branch ramal
c:\PruebasGit>git status
On branch master
nothing to commit, working tree clean
```

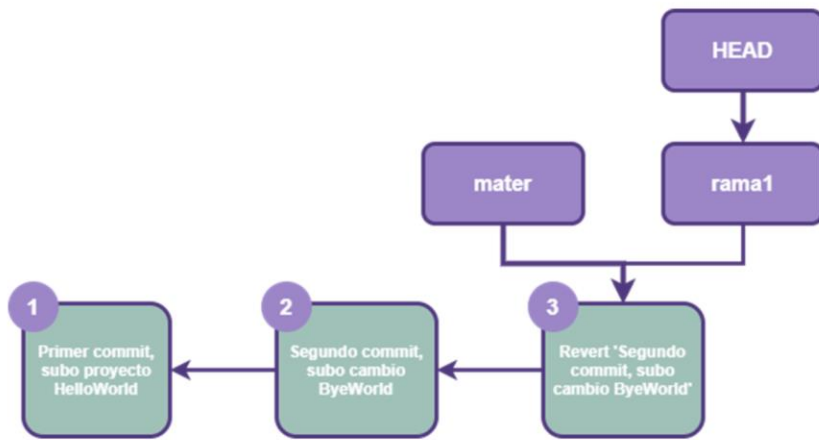
```
git checkout <nombre_rama>
```

```
Command Prompt
c:\PruebasGit>git checkout ramal
Switched to branch 'ramal'
c:\PruebasGit>git status
On branch ramal
nothing to commit, working tree clean
c:\PruebasGit>
```



Con los siguientes comandos podemos crear una rama nueva y movernos a ella. Con `git branch` la creamos, aunque seguimos en la que estábamos, y con `checkout` nos movemos a dicha rama (`checkout` también se suele usar de forma parecida a `reset`, aunque con variaciones).

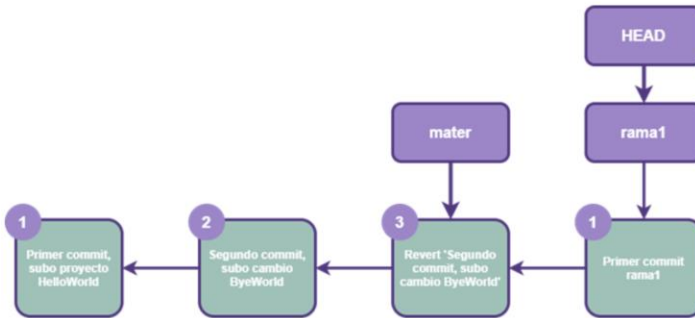
Trabajar con ramas -> Crear rama y moverse a ella



Y con las 2 operaciones que acabamos de hacer, nuestra rama quedaría tal que así. Como se puede ver, la nueva rama, como ocurría con la master, sigue siendo lo mismo, un puntero a un conjunto de commits.

Y conviene fijarse en que, cada vez que cambiamos de rama, también cambia el HEAD, dado que siempre apunta al último commit de la rama en la que estemos.

Trabajar con ramas -> Añadir cambio a mi rama



```
Command Prompt
C:\PruebasGit>git commit -m "Primer commit rama1"
[rama1 aa9a6dd] Primer commit rama1
1 file changed, 1 insertion(+), 1 deletion(-)

C:\PruebasGit>git status
On branch rama1
nothing to commit, working tree clean

C:\PruebasGit>git log
commit aa9a6ddc5037dbf0d4c7212e9df681d490963f2e (HEAD -> rama1)
Author: Lola Villalobos
Date: Sat Nov 14 19:22:36 2020 +0100

    Primer commit rama1

commit c8bd0925003a256a3c9a2053ef22d04a815c8fa7 (master)
Author: Lola Villalobos
Date: Sat Nov 14 18:20:52 2020 +0100

    Segundo commit, subo cambio ByeWorld

Revert "Segundo commit, subo cambio ByeWorld"
This reverts commit 68c2321a0ee598593c54babfff72a413bf6b2d5.

commit 68c2321a0ee598593c54babfff72a413bf6b2d5
Author: Lola Villalobos
Date: Sat Nov 14 18:18:24 2020 +0100

    Segundo commit, subo cambio ByeWorld

commit b10ef1242c1f99a4dd6bcc177799c2e6751961f4
Author: Lola Villalobos
Date: Sat Nov 14 14:12:35 2020 +0100

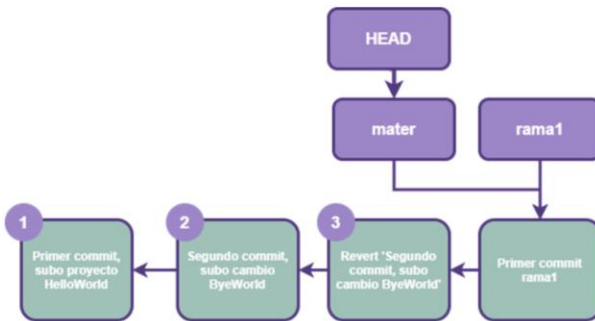
    Primer commit, subo proyecto HelloWorld
```



Y con las 2 operaciones que acabamos de hacer, nuestra rama quedaría tal que así.

Trabajar con ramas -> Merge

`git merge <nombre_rama>`



```
Command Prompt
C:\PruebasGit>git checkout master
Switched to branch 'master'

C:\PruebasGit>git status
On branch master
nothing to commit, working tree clean

C:\PruebasGit>git merge rama1
Updating c8bd692..aa9a6de
Fast-forward
 HelloWorld/HelloWorld/Program.cs | 2 +
 1 file changed, 1 insertion(+), 1 deletion(-)

C:\PruebasGit>git log
commit aa9a6dec50370b10d4c7212e9df6814d90963f2e (HEAD -> master, rama1)
Author: Lola Villalobos
Date: Sat Nov 14 19:22:36 2020 +0100

    Primer commit rama1

commit c8bd6925003a256a3c9a2053ef22d04a015c8fa7
Author: Lola Villalobos
Date: Sat Nov 14 18:20:52 2020 +0100

    Revert "Segundo commit, subo cambio ByeWorld"

    This reverts commit 68c2321a0ee598593c54babfff72a413bfb6b2d5.

commit 68c2321a0ee598593c54babfff72a413bfb6b2d5
Author: Lola Villalobos
Date: Sat Nov 14 18:18:24 2020 +0100

    Segundo commit, subo cambio ByeWorld
```



Dado que la motivación principal detrás de la creación de ramas es aislar desarrollos, llega un momento en el que se deseará llevar los cambios de una a otra y unificar dichos desarrollos. Para esto está la operación `git merge`, que se encarga de llevar los cambios de la rama especificada a la rama en la que nos encontramos ahora. De modo que, para poder llevar nuestros cambios en `rama1` a la `master`, una vez situados en la rama `master`, hacemos merge de `rama1`.

Repositorio remoto -> Crear repositorio remoto

git init --bare

(carpeta de repositorio remoto)

```
Command Prompt
c:\CharlaGit\PruebaGit_remoto>git init --bare
Initialized empty Git repository in c:/CharlaGit/PruebaGit_remoto/
```

**git remote add
<alias_repo> <ruta>**

(carpeta de repositorio local)

```
Command Prompt
c:\CharlaGit\PruebaGit_local>git log
commit 97e60d86422bee48d887692ed4403df437c33cf7 (HEAD -> master)
Author: Lola Villalobos
Date: Sat Nov 14 23:47:08 2020 +0100

    Primer commit, subo proyecto HelloWorld

c:\CharlaGit\PruebaGit_local>git remote add repoRemoto c:\CharlaGit\PruebaGit_remoto
```



Women for Technical Talks W4TT

<https://women4tt.blogspot.com>

#W4TT

#anfitrionasw4tt

Next Step

Axazure

Hasta ahora hemos trabajado únicamente en local, pero en algún momento habrá que conectarse con un repositorio remoto para poder compartir nuestros cambios. Para crearlo directamente en local o incluso nuestro servidor, necesitamos crear un repositorio ligeramente diferente, con la opción `--bare`, que genera el repositorio vacío, sin directorio de trabajo ni `Index`, de forma que no puedes añadir archivos mediante `add` o `commits`, sólo usando `push` y `pull`, que son los comandos propios de los repositorios remotos. Una vez generado, habría que enlazar nuestro repositorio local a éste mediante `git remote`.

Repositorio remoto -> Subir cambios

`git push <alias_repo> <rama>`

```
Command Prompt

c:\CharlaGit\PruebaGit_local>git remote
repoRemoto

c:\CharlaGit\PruebaGit_local>git push repoRemoto master
Enumerating objects: 8, done.
Counting objects: 100% (8/8), done.
Delta compression using up to 8 threads
Compressing objects: 100% (8/8), done.
Writing objects: 100% (8/8), 1.29 KiB | 659.00 KiB/s, done.
Total 8 (delta 0), reused 0 (delta 0)
To c:\CharlaGit\PruebaGit_remoto
 * [new branch]      master -> master

c:\CharlaGit\PruebaGit_local>git log
commit 97e60d86422bee48d887692ed4403df437c33cf7 (HEAD -> master, repoRemoto/master)
Author: Lola Villalobos <lola.villalobos@axazure.com>
Date: Sat Nov 14 23:47:08 2020 +0100

    Primer commit, subo proyecto HelloWorld
```



Y una vez enlazado, ya podríamos subir nuestros cambios al repositorio remoto.

Repositorio remoto -> Enlazar con repositorio remoto

`git remote add <alias_repo> <ruta>`

```
Command Prompt
c:\CharlaGit\PruebaGit_local2>git init
Initialized empty Git repository in c:/CharlaGit/PruebaGit_local2/.git/
c:\CharlaGit\PruebaGit_local2>git remote add gitHub https://github.com/lolaWizard/pruebaGit.git
```

`git clone <ruta>`

```
c:\CharlaGit\PruebaGit_local3>git clone https://github.com/lolaWizard/pruebaGit.git
Cloning into 'pruebaGit'...
remote: Enumerating objects: 8, done.
remote: Counting objects: 100% (8/8), done.
remote: Compressing objects: 100% (8/8), done.
remote: Total 8 (delta 0), reused 8 (delta 0), pack-reused 0
Unpacking objects: 100% (8/8), done.
```



Si ya existe el repositorio remoto y lo que queremos es descargarlo para empezar a trabajar con él. Tenemos 2 alternativas:

- Tener un repositorio ya existente creado con git init y asociarlo con git remote, como ya hicimos al crear el remoto desde cero.
- Usar git clone, que directamente crea un repositorio vacío y se descarga desde el remoto.

Repositorio remoto -> Bajar cambios

```
git fetch <alias_repo><rama>
```

```
c:\CharlaGit\PruebaGit_local2>git fetch gitHub master
Logon failed, use ctrl+c to cancel basic credential prompt.
Username for 'https://github.com': 
Password for 'https://github.com': 
remote: Enumerating objects: 8, done.
remote: Counting objects: 100% (8/8), done.
remote: Compressing objects: 100% (8/8), done.
remote: Total 8 (delta 0), reused 8 (delta 0), pack-reused 0
Unpacking objects: 100% (8/8), done.
From https://github.com/lolawizard/pruebaGit
 * branch      master      -> FETCH_HEAD
 * [new branch] master      -> gitHub/master
```

```
git pull <alias_repo><rama>
```

```
Command Prompt

c:\CharlaGit\PruebaGit_local2>git pull gitHub master
Logon failed, use ctrl+c to cancel basic credential prompt.
Username for 'https://github.com': 
Password for 'https://github.com': 
From https://github.com/lolawizard/pruebaGit
 * branch      master      -> FETCH_HEAD
```



Y para todo lo demás...

```
git --help
```



Women for Technical Talks W4TT

<https://women4tt.blogspot.com>

#W4TT
#anfitrionasw4tt

Next Step

 **axazure**

Herramientas que nos facilitan la vida



Git Extensions



Fork



Women for Technical Talks W4TT

<https://women4tt.blogspot.com>

#W4TT

#anfitrionasw4tt

Next Step

Azure

Para Git existen múltiples aplicaciones de escritorio, muchas de ellas gratuitas, como GitExtensions o SourceTree, que son las más conocidas. También existen otras como Fork, pero son de pago (aunque he probado la versión gratuita sin aparentes limitaciones). Y Visual Studio también incluye un menú para poder trabajar con Git.

Repositorios gratuitos



GitHub



Bitbucket



Women for Technical Talks W4TT

<https://women4tt.blogspot.com>

#W4TT
#anfitrionasw4tt

Next Step

 **axazure**

¿Por qué usar Git?

Ventajas

Seguridad

Mi copia, la del becario, la del analista...

Trabajo sin conexión

3 áreas de trabajo en local vs. 1 repositorio remoto

Rapidez

Trabajar en local es más rápido que con la wifi del vecino

Inconvenientes

Curva de aprendizaje

Si has llegado hasta aquí sin resoplar ¡Impostora!, ya sabes Git, no vengas a trollear.

[....]

Rellénalo con lo que se te ocurra, yo me he quedado sin ideas



Women for Technical Talks W4TT

<https://women4tt.blogspot.com>

#W4TT

#anfitrionasw4tt

Next Step

axazure

Y después de todo lo que hemos visto llega la gran pregunta: por qué dejar que Git entre en mi vida.

Pues como ya se ha comentado en algunas diapositivas anteriores, las principales ventajas que aporta Git son:

- Seguridad. Al tratarse de un sistema distribuido, hay una copia completa del repositorio en cada uno de los clientes y, a más copias, menos posibilidades de perder mi repo. Y para ilustrar la importancia de esto, tenemos la historia de Toy Story, que casi nos quedamos sin película después de que alguien hiciese un delete 'demasiado' bien hecho y la película se salvó gracias a que una de las trabajadoras teletrabajaba desde casa y tenía una copia.
- Se puede trabajar sin conexión. Al tener en local una copia de todo el repositorio, puedo trabajar en local la mayor parte del tiempo, operaciones como:
 - Hacer commit
 - Ver el histórico de cambios
 - Operar con ramas: crear, cambiar de rama, hacer un merge, borrar...
 - O revertir commitspueden hacerse perfectamente en local. Sólo necesito conexión cuando necesite compartir mis cambios con los compañeros o descargar los suyos.
- Rapidez. Por el mismo motivo que antes. Aunque la operación de clone es más lenta que en otros sistemas, ya que nos descargamos el repositorio completo con su histórico, otras operaciones como commit, merge, diff, blame o log son mucho más rápidas, ya que se hacen en local.

Y respecto a los inconvenientes:

- Git está muy bien pensado para aportar seguridad y rapidez al control de versiones, pero no es fácil a la primera, tiene una curva de aprendizaje un poco dura. Esto ocurre sobre todo si vienes de otros entornos, como TFS, donde pasas de tener un solo 'Check in' a tener un 'Commit' y un 'Push', o no tienes 'Staging Area', que sería el equivalente a incluir o no los cambios en la subida que haces al repositorio. Pero no te estreses, al final se le coge cariño, sólo necesita tiempo y dedicación, como todo en esta vida.



W4TT Anfitrionas:
Hablemos de tecnología

Gracias

Next Step  axazure

Recursos

Git

Libro 'Pro Git', de Scott Chacon, Ben Straub

Disponible gratuitamente: <https://git-scm.com/book/en/v2>

Tutoriales

<https://git-scm.com/doc>

<https://www.atlassian.com/git/tutorials>

Tutorial interactivo, muy divertido ;-)

<https://learngitbranching.js.org/>

Cheatsheets

<https://ndpsoftware.com/git-cheatsheet.html>

<https://www.atlassian.com/git/tutorials/atlassian-git-cheatsheet>

Ventajas de usar Git vs. TFS

<https://michaelscodingspot.com/life-changed-moving-tfvctfs-git/>

Gráficos

Iconos

www.freepik.com

www.rawpixel.com

Diagramas

Elaborados con diagrams.net



Women for Technical Talks W4TT

<https://women4tt.blogspot.com>

#W4TT

#anfitrionasw4tt

Next Step

Axazure