Instituto Tecnológico Superior de Cajeme
Ingeniería en Sistemas Computacionales

git

GitHub

Manual

Elaborado por:
José Luis Beltrán Márquez
lbeltran@itesca.edu.mx

Cd. Obregón, Sonora, México. Agosto de 2018.

Índice

Capítulo 1

git

git es un software de control de versiones diseñado por Linus Torvalds, pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando éstas tienen un gran número de archivos de código fuente. Su propósito es llevar registro de los cambios en archivos de computadora y coordinar el trabajo que varias personas realizan sobre archivos compartidos.

Al principio, Git se pensó como un motor de bajo nivel sobre el cual otros pudieran escribir la interfaz de usuario o front end como Cogito o StGIT. 3 Sin embargo, Git se ha convertido desde entonces en un sistema de control de versiones con funcionalidad plena. 4 Hay algunos proyectos de mucha relevancia que ya usan Git, en particular, el grupo de programación del núcleo Linux.

El mantenimiento del software Git está actualmente (2009) supervisado por Junio Hamano, quien recibe contribuciones al código de alrededor de 280 programadores. En cuanto a derechos de autor Git es un software libre distribuible bajo los términos de la versión 2 de la Licencia Pública General de GNU.

1.1 fc (file compare en windows), diff (diferencia en Mac y Linux)

El comando ***diff -u arhivo1 archivo2***

donde -u es para ver en el formato unificado estándar permite ver las diferencias entre los 2 archivos.

Pone los siguientes signos al principio de cada línea:

esta igual en ambos archivos

- solo está en el primer archivo

+ solo esta en el segundo archivo

1.2 Algunas formas de Controlar las Versiones de un proyecto

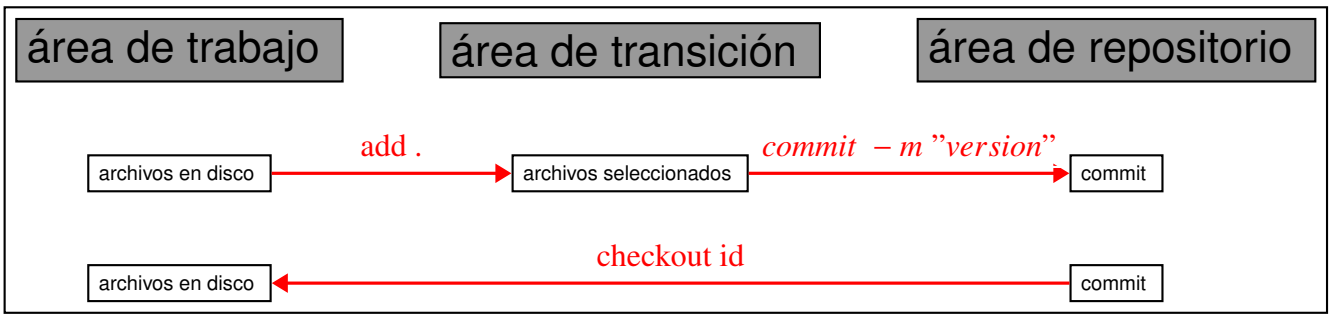
Se trata de guardar los avances de los diferentes documentos, por si hay necesidad de recuperar una versión anterior; algunas opciones son:

- Una versión primitiva sería usar diferentes carpetas para guardar manualmente cada versión, con la desventaja del uso excesivo del disco duro
- Usar programas de terceros como Dropbox o Google Docs que guardan las diferentes versiones de cada archivo, aunque solo por tiempo limitado y dependiendo del tipo de contrato.
- git
- CVS (Concurrent Version Control)
- SVN (SubVersion)
- Mercurial (hg)

Aunque hay que responder lo siguiente para cada una de las opciones:

- ¿puedes usar cualquier editor?
- ¿puedes respaldar sin estar conectado a internet?
- ¿puedes escoger el punto de respaldo (commit)?

1.3 Diferentes areas usadas en git



1.4 área de trabajo

Es la carpeta raíz donde se realiza la inicialización del git, es donde están los archivos fuente que a la postre se respaldan como diferentes versiones en el repositorio en forma de commits. Esta área no tiene id

1.5 área de transición (staging area)

Es la relación de los archivos que juntos se guardarán como un solo commit en el repositorio y también es una copia del último commit, aunque carece de id.

1.6 commit

Es el respaldo que se hace en el repositorio, cada commit tiene un id y puede ser uno o varios archivos según la configuración del área de transición.

Cuando hacemos un commit, el git nos pide un nombre para el commit.

El id lo asigna automáticamente el git al commit.

1.7 Repositorio

El repositorio es la carpeta que contiene todos los commits que se han realizado.

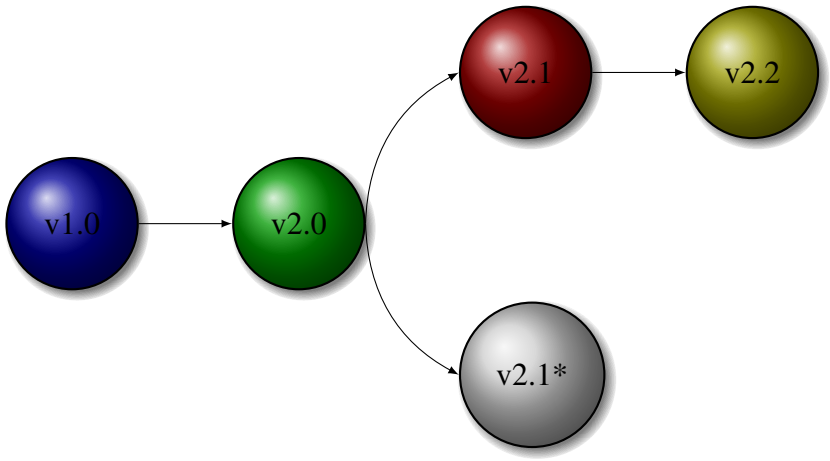
Es una carpeta oculta (ya que su nombre comienza con un punto) de nombre **.git** y allí es donde se guardan los commits.

1.8 rama (branch)

Es posible hacer diferentes versiones con ligeros cambios a la línea principal, por ejemplo manejar idiomas, a esas variantes se les conoce como ramas o branch en inglés.

Una bifurcación (en inglés fork), en el ámbito del desarrollo de software, es la creación de un proyecto en una dirección distinta de la principal u oficial tomando el código fuente del proyecto ya existente. Como resultado de la bifurcación se pueden llegar a generar proyectos diferentes que cubren necesidades distintas aunque similares. El término también puede ser usado para representar la ramificación de cualquier trabajo. Un ejemplo de bifurcación es la Enciclopedia Libre escindida de la Wikipedia en español.

Las bifurcaciones de proyectos de software libre surgen de un cisma en los objetivos o un choque de personalidades. En una bifurcación, ambos lados asumen derechos de autor idénticos pero típicamente solo el grupo de mayor tamaño, o el que contiene al arquitecto original, retendrá el nombre original completo y la comunidad de usuarios asociada. Por ello existe una penalización asociada con la bifurcación.



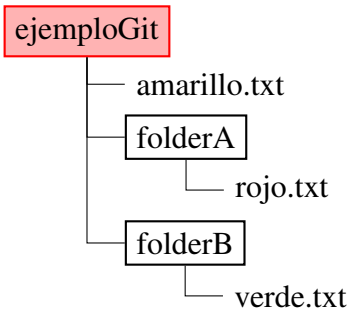
1.9 parámetros del comando git

- git init (para inicializar la carpeta y el servicio)
- git config - -global user.email ”usuario@serverMail.com”
- git config - -global user.name ”nombre usuario”
- git config - -global user.pass ”contraseña”
- git remote add origin https://github.com/belmarq/Ejemplo1.git
- git add archivo (para agregar archivos o carpetas al control de versiones)
- git rm <archivos> (quita los archivos dados del área de transición)
- git commit -a (para hacer una fotografía de todos los archivos considerados usando un asistente)
- git commit -m ”nombre de la versión”
- git log (lista los diferentes commits del repositorio)
- git log - -stat (lista los diferentes commits del repositorio y sus diferencias)
- git log - -graph - -oneline master (muestra resumidos los cambios)
- git status (muestra los archivos incluidos en el área de transición y los no incluidos también)
- git diff (muestra las diferencias entre el proyecto de trabajo y el área de transición)
- git diff <id commit 1> <id commit 2> (muestra las diferencias entre el commit 1 y el commit 2)

- git diff - -staged (muestra las diferencias entre el repositorio y el área de transición)
- git checkout <rama> (carga el último commit de la rama indicada, y hace a esa rama la predeterminada para los siguientes commits)
- git checkout <id commit> (carga el commit indicado)
- git branch (muestra las diferentes ramas del proyecto)
- git branch <rama> (crea una nueva rama de nombre rama)
- git clone
- git push -u origin master (copia el repositorio local .git al repositorio en línea)
- git pull

1.10 Ejemplo

1.10.1 Crear carpeta de trabajo y documentos de inicio

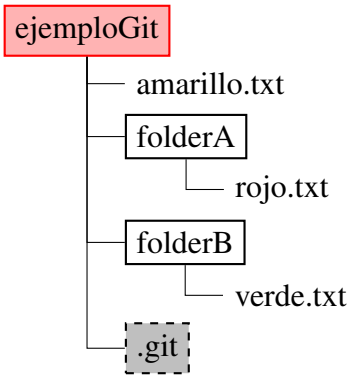


1. En la pantalla de comandos (o desde el explorador de archivos), en la carpeta home, hacer una carpeta de nombre ejemploGit
2. Dentro de la carpeta ejemploGit, crear una carpeta de nombre folderA, otra de nombre folderB
3. Dentro de ejemploGit crear un archivo de texto de nombre amarillo.txt con el contenido: Amarillo no me pongo, amarillo es mi color
4. Dentro de folderA crear un archivo de texto de nombre rojo.txt con el contenido: El rojo es pasión
5. Dentro de folderB crear un archivo de texto de nombre verde.txt con el contenido: Que solo los caminos queden sin sembrar

1.10.2 Inicializar git

1. Estando en la carpeta ejemploGit, crear el repositorio (*carpeta .git*) con el comando **[git init]**

2. Puedes verificar si se creo la carpeta oculta .git, usando el explorador de windows con la opción de mostrar archivos ocultos activa, o con el comando **[dir /a]**, en linux es **ls -s**



3. Los siguientes comandos son para registrar tus datos en la computadora, en este caso tu usuario y contraseña de github que usaremos posteriormente para hacer respaldos remotos; el usuario y contraseña es el que pusiste en github cuando creaste tu usuario, NOTA: no es el usuario y contraseña de tu email.

   - git config - -global user.email ”usuario@tuServerMail”
   - git config - -global user.name ”nombre usuario (de github)”
   - git config - -global user.pass ”contraseña (de github)”

1.10.3 Agregar archivos a el área de transición

1. con el comando **[git add .]** agregamos todos los archivos de la carpeta y subcarpetas al area de transición

2. Podemos verificar usando el comando **[git status]** el cual nos indica los archivos considerados para hacer el commit, sin embargo nos dice que actualmente no hay commit

3. Otra forma de hacerlo es con el comando **[git log]** que también nos dice que no hay commits

1.10.4 Hacer un commit

1. El commit respalda los archivos que agregaste a el área de transición mediante el comando add.

2. Una opción es usar el asistente con **[git commit -a]**, por eso usamos la opción -a, el cual nos da oportunidad de escribir el nombre del commit y ver el resumen del commit en una ventana emergente, en caso de windows es con Esc, shift ZZ

3. Otra opción es hacerlo manual, sin asistente con **[git commit -m** "*nombreVersion*"**]**

4. poner nombre de v1.0

5. Verificar que se hizo el commit con el comando **[git log]**, o el comando **[git log - - stat]**

6. el número gigantesco que aparece a la derecha de commit es el id del commit

1.10.5 Hacer cambios a los archivos y comparar con los respaldados

1. El área de transición tiene una copia sin id del último commit

2. Checamos si hay diferencias entre lo que hay guardado en la carpeta de trabajo y el área de transición, usando el comando **[git diff]**, en este caso, no nos muestra nada, ya que son iguales los contenidos.

3. Hacemos un cambio a el archivo amarillo.txt, y agregamos otra línea con el contenido de: Quien de amarillo se viste, o de su belleza confía o de sinvergüenza se pasa

4. Guardamos el archivo, si hacemos la comparación de contenidos, veremos que si hay diferencias ya que agregamos una linea nueva. **[git diff]**

5. Si usamos el comando **[git add .]** agregamos al área de transición los archivos modificados

6. si volvemos a ver las diferencias entre el área de trabajo y el área de transición, veremos ahora que ya no hay diferencia, ya que actualizamos el área de transición. comando **[git diff]**

7. Sin embargo, la información almacenada del commit si es diferente al área de trabajo; podemos checar la diferencia entre el área de transición y el repositorio con el comando **[git diff - -staged]**

8. Como vemos, si hay diferencias entre el commit del repositorio y el área de transición

1.10.6 Hacer un respaldo nuevamente y hacer comparaciones

1. Hay que nuevamente hacer un respaldo con los cambios realizados

2. git commit -m "*v*2.0"

3. Verificar que se hizo el commit con el comando **[git log]**, o el comando **[git log - - stat]**

4. Verificar que se hizo el commit con el comando **[git log]**, o el comando **[git log -p]**

5. Podemos verificar que no hay diferencia entre el área de trabajo y el repositorio con el comando **[git diff - -staged]**

6. Ahora tenemos 2 commits, y para ver las diferencias entre ellos, podemos usar el comando **[git diff <id commit v1.0> <id commit v2.0>]**

1.10.7 Hacer nuevamente cambios y agregar los cambios como rama experimental

1. Hacemos un cambio a el archivo rojo.txt, y reemplazamos la primer línea con el contenido de: El rojo es confianza en sí mismo, coraje, valentía y una actitud optimista ante la vida.

2. Guardamos el archivo; si hacemos la comparación de contenidos, veremos que si hay diferencias ya que reemplazamos una linea. el comando **[git diff]** nos muestra las diferencias entre el área de trabajo y el área de transición

3. Si usamos el comando **[git add .]** agregamos al área de transición los archivos modificados

4. si volvemos a ver las diferencias entre el área de trabajo y el área de transición, veremos ahora que ya no hay diferencia, con el comando **[git diff]**

5. Sin embargo, la información almacenada del commit si es diferente al área de trabajo; podemos checar la diferencia entre el área de trabajo y el repositorio con el comando **[git diff - -staged]**

6. Como vemos, si hay diferencias entre el commit del repositorio y el área de trabajo

7. Hay que crear una rama experimental, así que usamos el comando **[git branch]** para listar las ramas actuales, solo aparece la rama master.

8. Creamos la rama experimental con el comando **[git branch exp]**

9. Volvemos a usar el comando **[git branch]** para listar las ramas actuales, ahora debe aparecer la rama master y la exp, sin embargo el asterisco en master indica que la rama está activada como predeterminada.

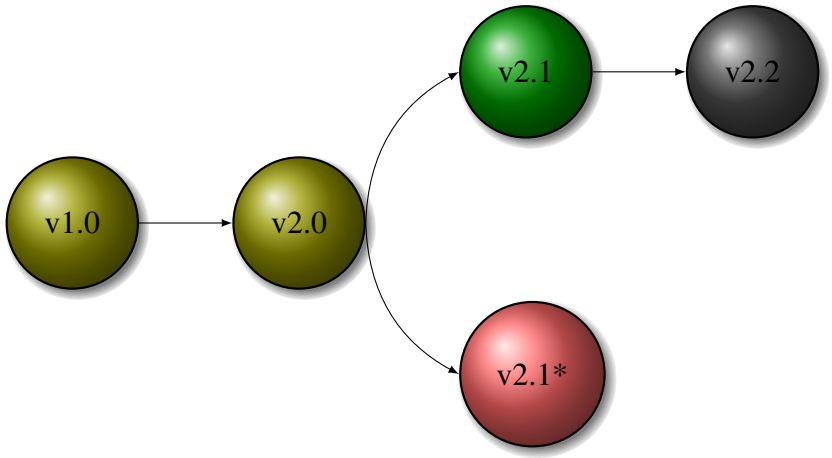
10. Para cambiar la rama predeterminada a exp, usamos el comando **[git checkout exp]**

11. Volvemos a usar el comando **[git branch]** para listar las ramas actuales, ahora debe aparecer la rama master y la exp, sin embargo el asterisco en exp indica que la rama está activada como predeterminada.

12. Hay que nuevamente hacer un respaldo con los cambios realizados, para que se guarde el commit en la nueva rama exp

13. git commit -m ”$v2.1*$ ”

14. Podemos comprobar usando el comando **[git log]**, además vemos que el HEAD ahora esta en exp



1.10.8 Validar los cambios y guardarla como versión liberada

1. Hacemos un cambio a el archivo verde.txt, y agregamos otra línea con el contenido de: El verde es: naturaleza, armonía, crecimiento, exuberancia, fertilidad, frescura, estabilidad, resistencia, dinero.

2. Guardamos el archivo, si hacemos la comparación de contenidos entre el área de trabajo y el área de transición, veremos que si hay diferencias ya que agregamos una linea nueva. **[git diff]**

3. Si usamos el comando **[git add .]** agregamos al área de transición los archivos modificados

4. si volvemos a ver las diferencias entre el área de trabajo y el área de transición, veremos ahora que ya no hay diferencia, ya que actualizamos el área de transición. comando **[git diff]**

5. Sin embargo, la información almacenada del commit si es diferente al área de trabajo; podemos checar la diferencia entre el área de trabajo y el repositorio con el comando **[git diff - -staged]**

6. Como vemos, si hay diferencias entre el commit del repositorio y el área de trabajo

7. Hay que crear una rama de versiones liberadas, así que usamos el comando **[git branch]** para listar las ramas actuales.

8. Creamos la rama validada con el comando **[git branch validada]**

9. Volvemos a usar el comando **[git branch]** para listar las ramas actuales, ahora debe aparecer la rama master, exp y validada, sin embargo el asterisco en exp indica que la rama está activada como predeterminada.

10. Para cambiar la rama predeterminada a validada, usamos el comando **[git checkout validada]**

11. Volvemos a usar el comando **[git branch]** para listar las ramas actuales, ahora debe aparecer la rama validada como predeterminada.

12. Hay que nuevamente hacer un respaldo con los cambios realizados, para que se guarde el commit en la nueva rama exp

13. git commit -m ”*v*2.1”

14. Podemos comprobar usando el comando **[git log]**, además vemos que el HEAD ahora esta en validada

15. También podemos usar el comando **[git log - -stat]**

16. También podemos usar el comando **[git log -p]**

Capítulo 2

GitHub

GitHub es una plataforma de desarrollo colaborativo para alojar proyectos utilizando el sistema de control de versiones Git. Se utiliza principalmente para la creación de código fuente de programas de computadora. El software que opera GitHub fue escrito en Ruby on Rails. Desde enero de 2010, GitHub opera bajo el nombre de GitHub, Inc. Anteriormente era conocida como Logical Awesome LLC. El código de los proyectos alojados en GitHub se almacena típicamente de forma pública, aunque utilizando una cuenta de pago, también permite hospedar repositorios privados.

El 4 de junio de 2018, Microsoft compró GitHub por la cantidad de 7.500 millones de dólares.

2.1 Ejemplo

2.1.1 Crear una cuenta de GitHub

Entrar a la página de github: *https* : //*github.com*/ y crear una cuenta.

2.1.2 Crear un repositorio remoto y subir mi repositorio local

1. Agregar un repositorio nuevo. En mi caso le puse **2018Android**

2. Al crear el repositorio, la página te regresa la liga completa y un recordatorio de la instrucciones que tienes que hacer en tu computadora para subir tu repositorio local, al repositorio remoto de github:

```
...or create a new repository on the command line:

echo "# 2018Android" >> README.md
git init
git add README.md
git commit -m "first commit"
```

```
git remote add origin https://github.com/belmarq/2018Android.git
git push -u origin master

...

...or push an existing repository from the command line
git remote add origin https://github.com/belmarq/2018Android.git
git push -u origin master
```

3. El comando donde está el parámetro remote del comando git, significa que le estamos diciendo a git la dirección del repositorio remoto

4. Nos va a pedir nuevamente el usuario y la contraseña de github

5. El comando donde está el parámetro push del comando git, significa que queremos subir la rama master del repositorio local, al repositorio remoto recién indicado

2.1.3 Descargar un repositorio remoto a un repositorio local

1. Hay 2 opciones: git clone, y git pull

2. **git clone** es la forma de obtener una copia local de un repositorio existente para trabajar. Por lo general, solo se utiliza una vez para un repositorio determinado, a menos que desee tener varias copias de trabajo. (O quiere obtener una copia limpia después de estropear su local ...)

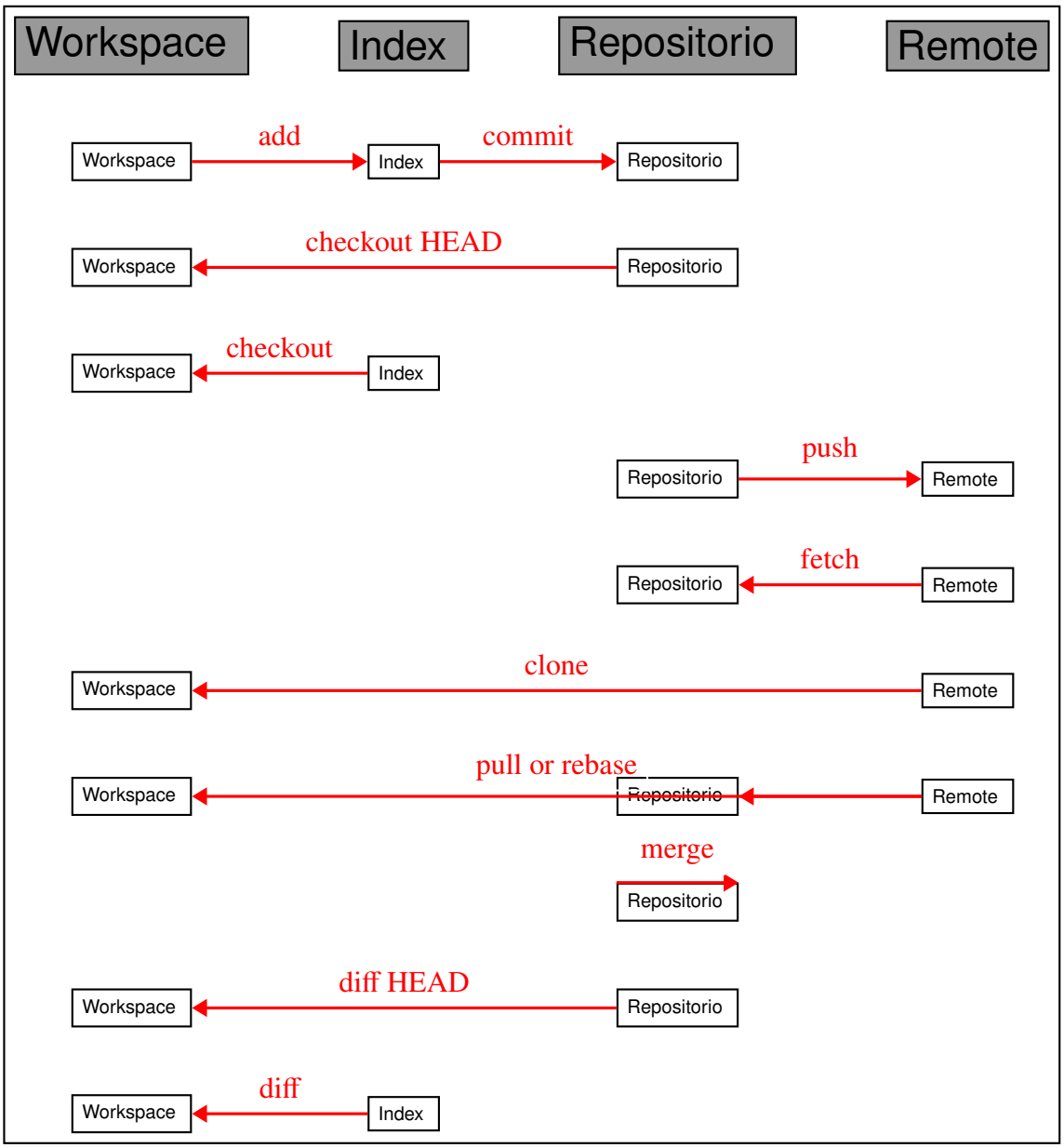
3. **git pull** (o git fetch + git merge) es la manera de actualizar esa copia local con nuevas confirmaciones del repositorio remoto. Si colaboras con otros, es un comando que ejecutarás con frecuencia

Capítulo 3

Resumen

Ver la página oficial de documentación

`https://git-scm.com/docs/`


Workspace
Index
Repositorio
Remote
add
commit
Workspace
Index
Repositorio
checkout HEAD
Workspace
Repositorio
checkout
Workspace
Index
push
Repositorio
Remote
fetch
Repositorio
Remote
clone
Workspace
Remote
pull or rebase
Workspace
Repositorio
Remote
merge
Repositorio
diff HEAD
Workspace
Repositorio
diff
Workspace
Index