Ceiba
software

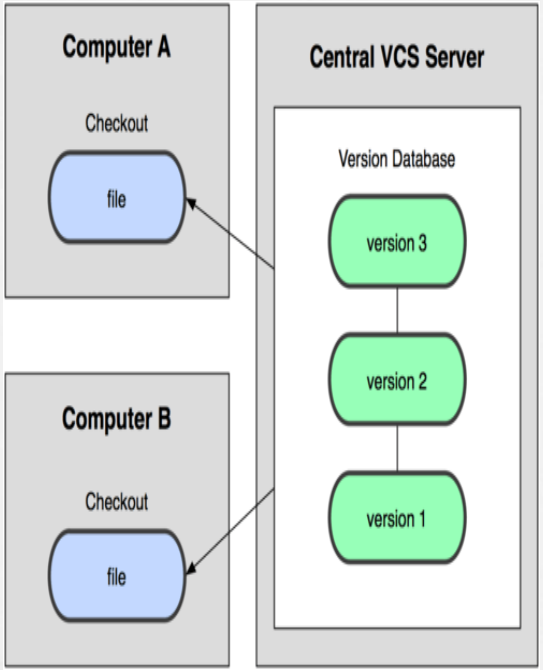
¿QUE ES GIT?

Es un software rastreador. Le da seguimiento a todos los cambios que se ejecutan sobre un archivo o carpeta. Cada cambio que hagas en un directorio, GIT se da cuenta y lo registra. Así de simple.

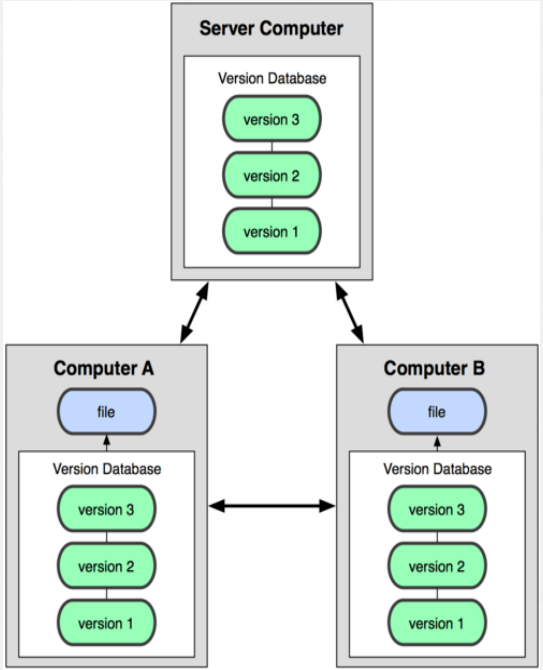
Ceiba
software

Diferencias sistemas control versiones

Sistemas de control de versiones centralizados

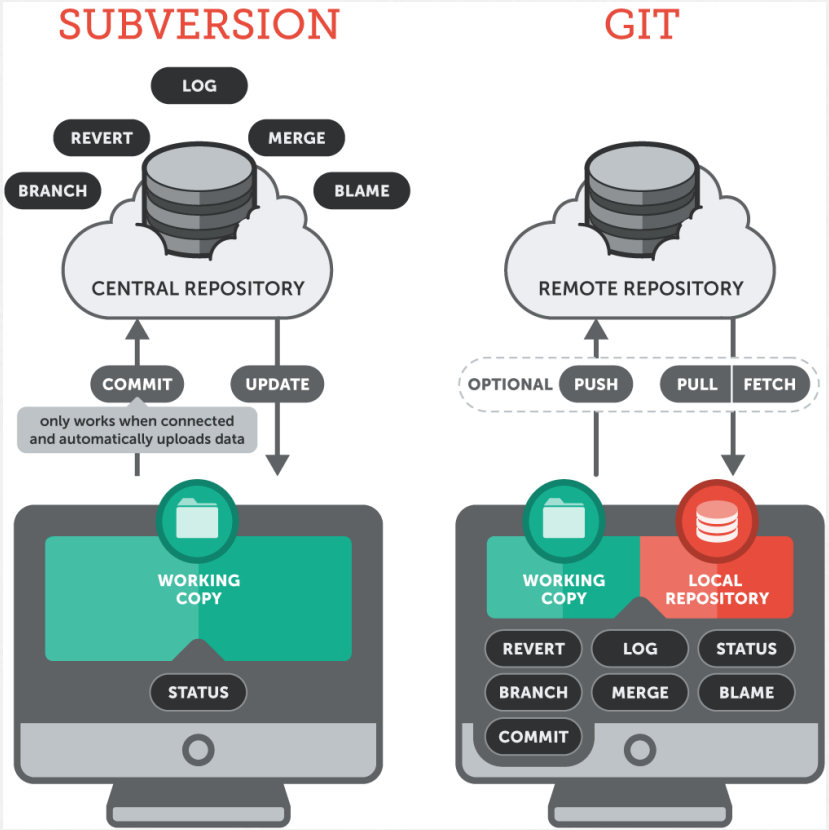


Sistemas de control de versiones distribuidos



Ceiba software

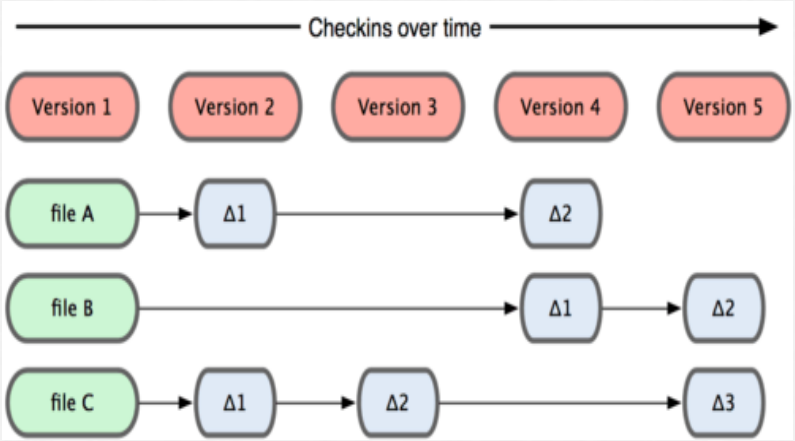
Diferencias sistemas control versiones



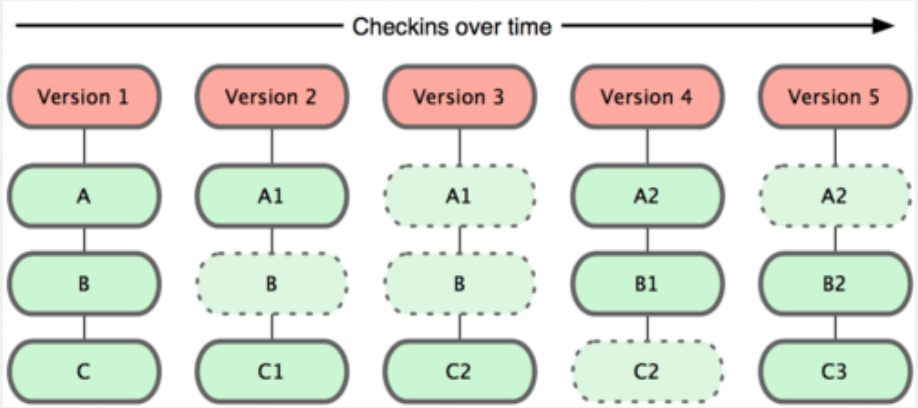
Ceiba software

Diferencias sistemas control versiones

Sistemas de control de versiones centralizados



Sistemas de control de versiones distribuidos

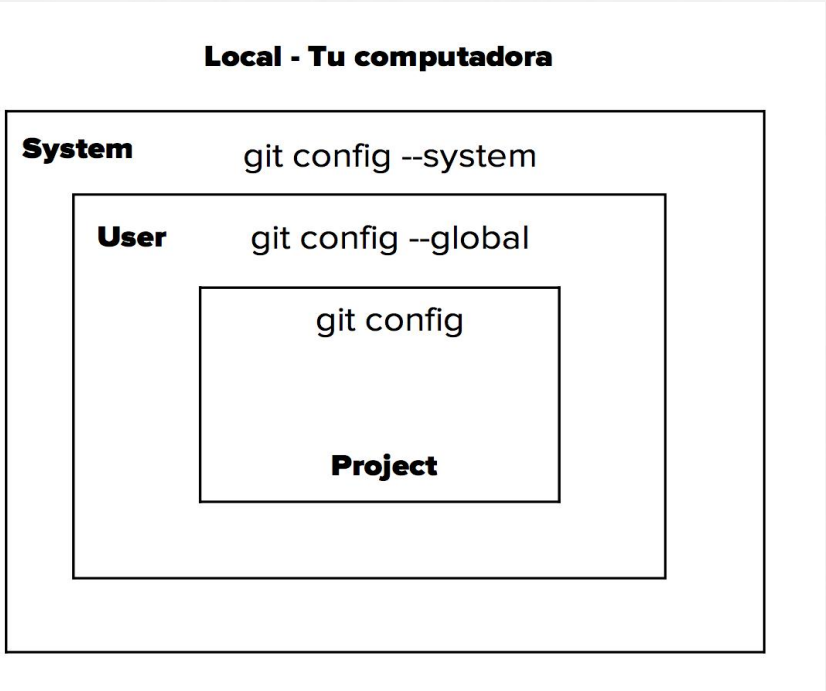


Ceiba software

GIT HELP

```
$ git help # Muestra lista con los comandos existentes

$ git help comando # Ayuda sobre comando especificado

$ git help add # Ayuda sobre el comando add

$ git add --help # Equivalente a anterior

$ man git-add # Equivalente a anterior
```

Configuración de GIT



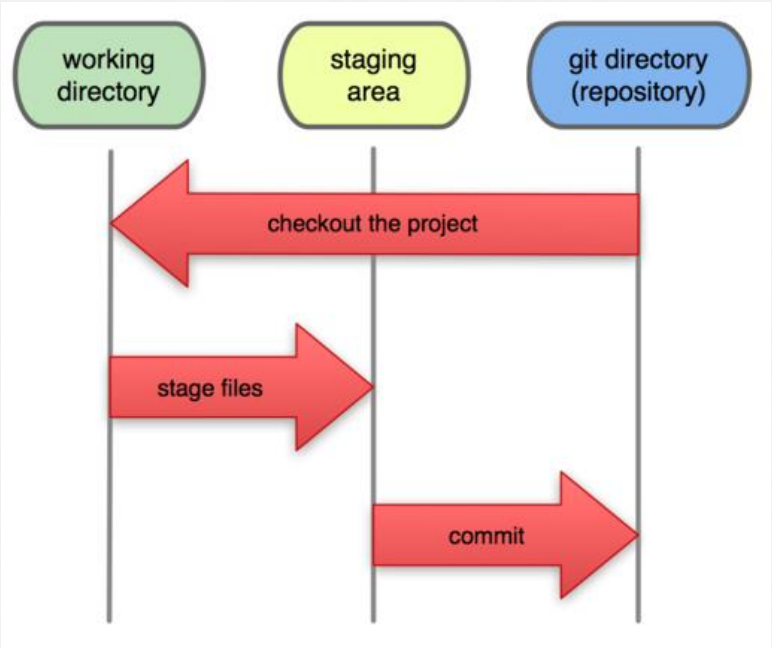
$ git config --global user.name “Nelson M“

$ git config --global user.email nelson@gmail.com

$ git config --list

Ceiba software

Secciones de proyecto GIT



★ **Directorio de trabajo**
Contiene los ficheros con los que estamos trabajando.

★ **Área de cambios (staging area, index)**
Contiene información sobre las modificaciones realizadas en los ficheros de directorio de trabajo que se guardarán en el próximo commit

★ **Directorio GIT (el repositorio)**
Guarda todas las versiones del proyecto.
- ficheros, metadatos

Ceiba software

Proyecto GIT desde cero

Ceiba
software

Datos importantes

El proyecto gestionado por GIT debe estar contenido en un directorio.

El directorio de trabajo: contiene los ficheros y subdirectorios del proyecto.

Los comandos GIT deben invocarse dentro del directorio de trabajo

Ceiba software

Iniciando Git

```
# moverse a la raíz del directorio de proyecto (trabajo)
cd .../miproyecto

# init: crea un nuevo proyecto GIT vacío en directorio miproyecto

$ git init

# Se crea el subdirectorio .git con los ficheros del repositorio
```

Información almacenada en .git

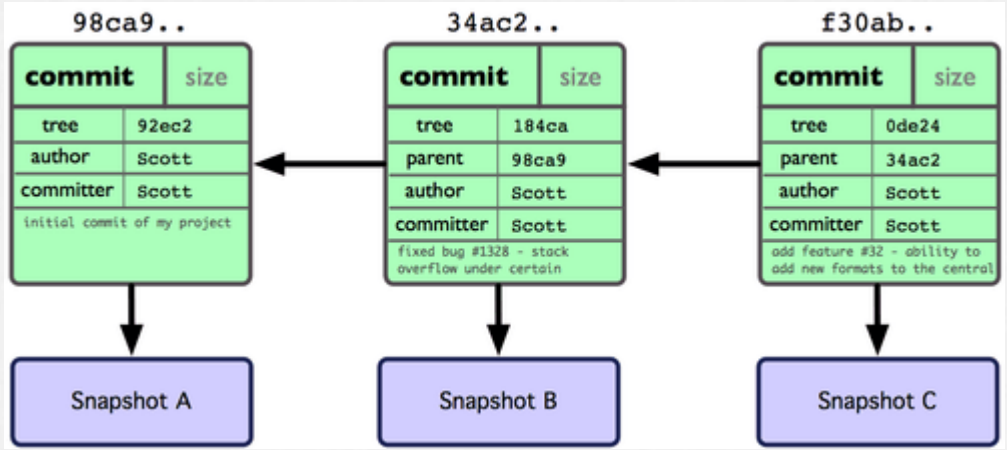
- Objetos que representan los ficheros, los directorios, los commit,...
- Referencias a repositorios remotos, ramas, ...
- Configuraciones

Ceiba software

Commit

La historia commit es un árbol.

- **Cada commit apunta a sus padres**



Ceiba software

Recomendaciones de commit

- Hacer commits con más frecuencias
- Revisar el código antes de hacer un commit
- Evitar commits con códigos a la mitad
- Escribir buenos mensajes en los commits
- hay que compartir código completo

Ceiba software

Primeros pasos en el proyecto


```
$ git add readme.txt # add: añade readme.txt al index
$ git add *.java # add: añade todos los java al index
$ git add . # add: añade todos los ficheros modificados al index

$ git commit -m 'Primera version'
# commit: congela una versión del proyecto en el repositorio

# A partir de este momento:
# - Hacer nuevos cambios en el directorio de trabajo:
# - Crear nuevos ficheros, editar ficheros existentes, borrar ficheros, ...
# - Añadir (add) los cambios en el index
# - Congelar (commit) una nueva versión con los cambios metidos en el index
```

Ceiba software

.gitignore

 # .gitignore es un fichero que informa de los ficheros que no debe gestionar GIT.
 # - git status no los presentará como ficheros untracked.
 # - git add . no los añadira al staging area.

Los ficheros **.gitignore** pueden crearse en cualquier directorio del proyecto,
y afectan a ese directorio y a sus subdirectorios.

 # Su contenido: líneas con patrones de nombres.
 # - Puede usarse los comodines * y ?
 # - Patrones terminados en / indican directorios
 # - Un patron que empiece con ! indica negación
 # - Se ignoran líneas en blanco y que comiencen con #
 # - [abc] indica cualquiera de los caracteres entre corchetes
 # - [a-z] indica cualquiera de los caracteres en el rango especificado

Ceiba software

.gitignore


```
# Ejemplo
private.txt          # excluir los ficheros con nombre "private.txt"
*.class              # excluir los ficheros bytecode de java
*.[oa]               # excluir ficheros acabados en ".o" y ".a"
!lib.a               # no excluir el fichero "lib.a"
*~                   # excluir ficheros acabados en "~"
testing/             # excluir los directorios llamados "testing"
```

Ceiba software

Quitar archivos del Area de Staging

 # Para eliminar del staging área las modificaciones de un fichero:
git reset HEAD <file>

Ceiba software

Deshacer modificación de un archivo

 # Si deseas deshacer los cambios que realizaste sobre un archivo y volver al estado anterior

git checkout -- nombreArchivo

Ceiba software

Modificar el ultimo commit

 # Para rehacer el último commit realizado usaremos git commit --amend
 # Para cambiar el mensaje de log.
 # Para añadir una modificación olvidada

git commit --amend -m “mensaje“

IMPORTANTE: no realizar --amend sobre un commit que se haya publicado

Ceiba software

git reset

 # git reset se usa para restaurar HEAD a otro estado.

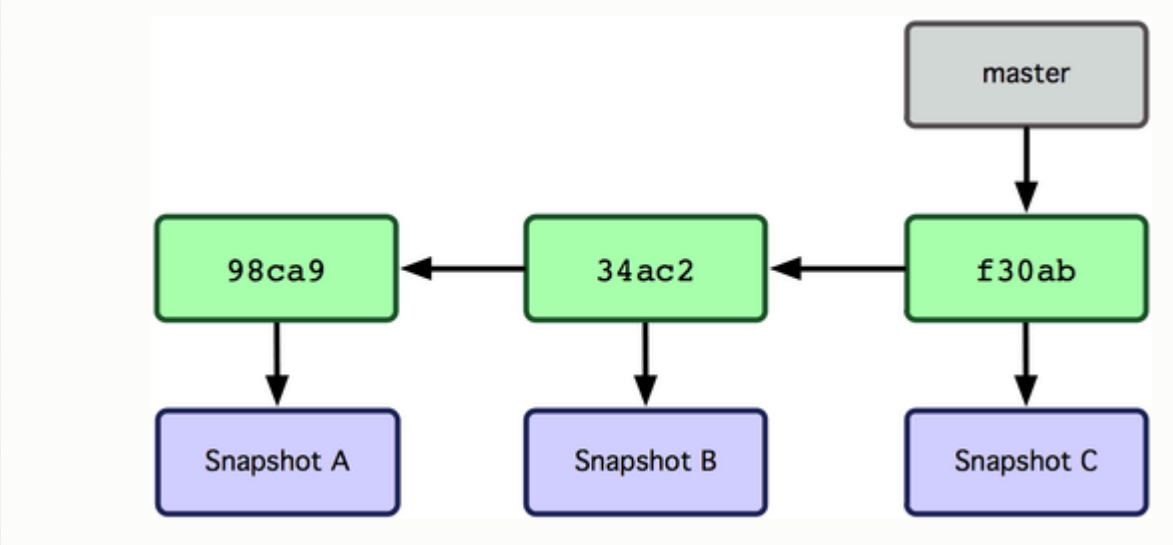
git reset <commit> Cambia HEAD para que apunte al commit dado.
-Se restaura el staging area al nuevo estado.
-Se pierden los cambios del staging area.
-Los ficheros del directorio de trabajo no se modifican.
-No se pierden las modificaciones existentes en ellos.

git reset --soft <commit> Se cambia HEAD para que apunte al commit dado.
- No se borran los cambios creados en el staging area.
-El staging area tambien reflejara los cambios existentes entre las
-versiones apuntadas por el HEAD anterior y el nuevo HEAD.
-No se modifican el directorio de trabajo.

git reset --hard <commit>Cambia HEAD para que apunte al commit dado
- restaura el staging area y restaura el directorio de trabajo eliminando todos los cambios existentes

¿Que es una rama?

- Una rama es un puntero a un commit.
- Este puntero se reasigna al crear nuevos commits.
- En GIT suele tenerse una rama principal llamada master.

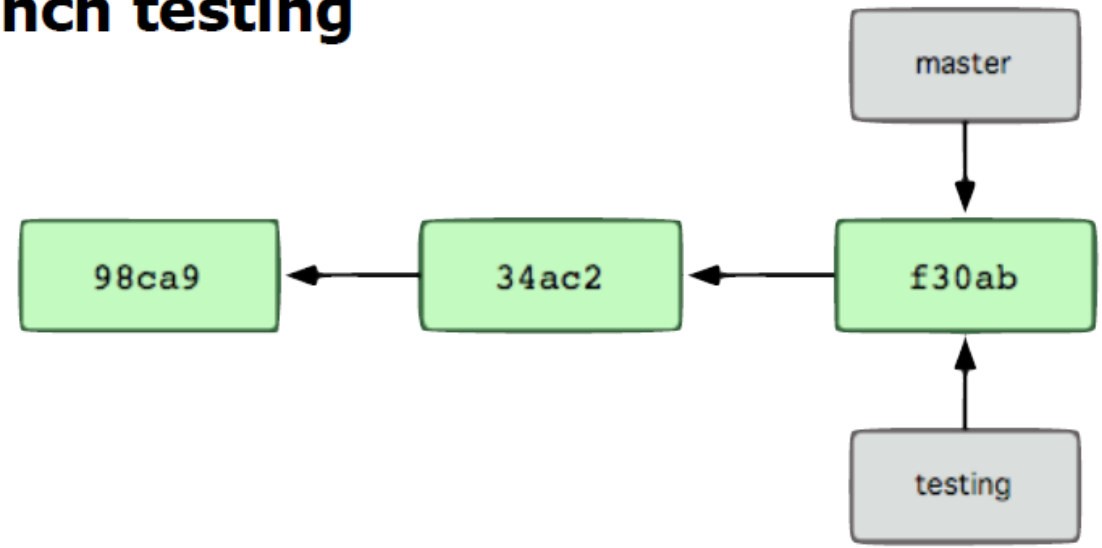


Ceiba software

Crear una rama

Para crear nuevas ramas: **git branch** <nombre>

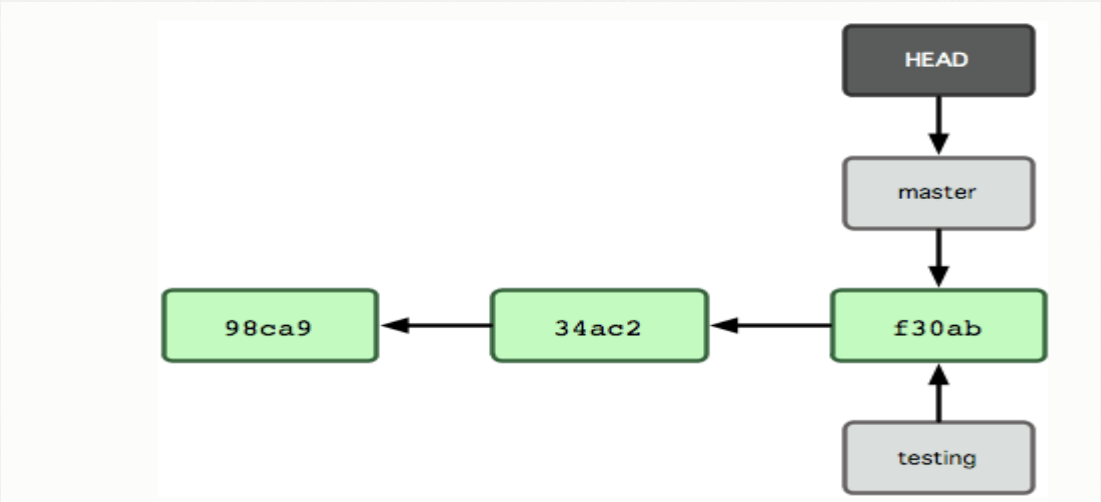
```
$ git branch testing
```



Ceiba software

Head

Es un apuntador a la rama en la cual estas trabajando en el momento, entonces es el que permite a git indicarnos en que rama estamos ubicados.



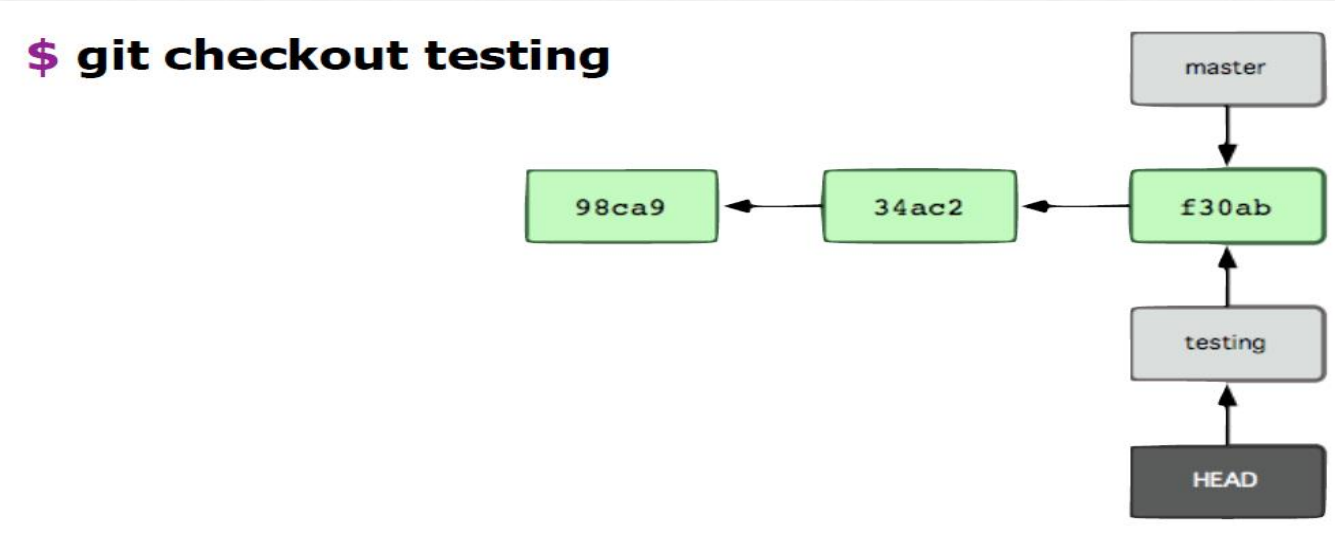
Ceiba software

Cambiar de rama

Para cambiar de rama se utiliza el comando **: git checkout** <nombre_rama>

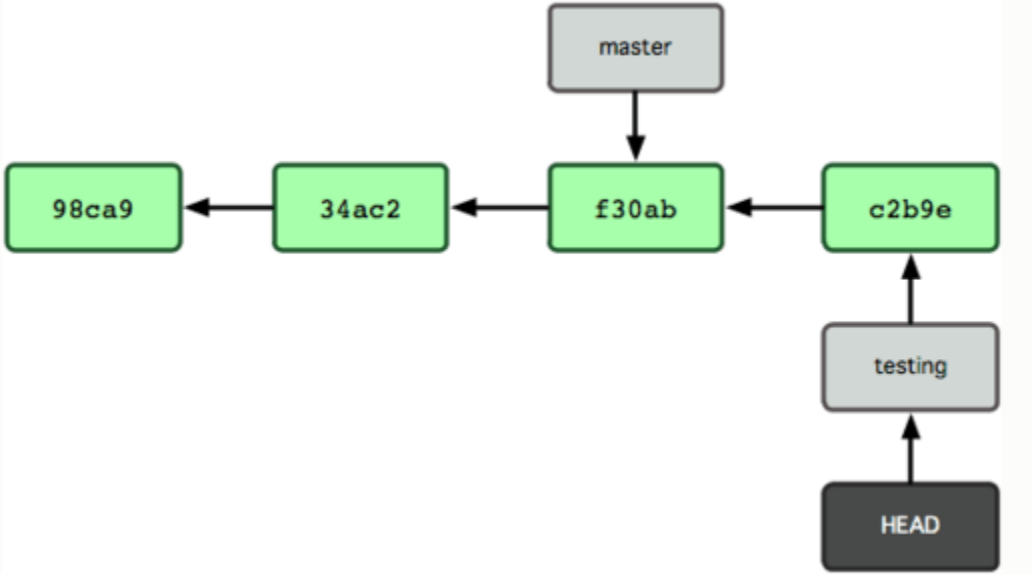
HEAD apuntará a la nueva rama.

Se actualizan el working directory y el staging area.



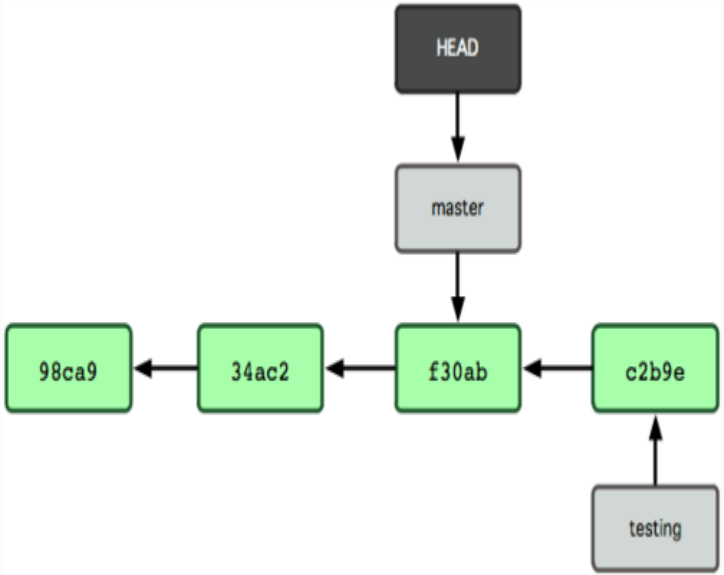
Ceiba software

Commit después de cambiar de rama

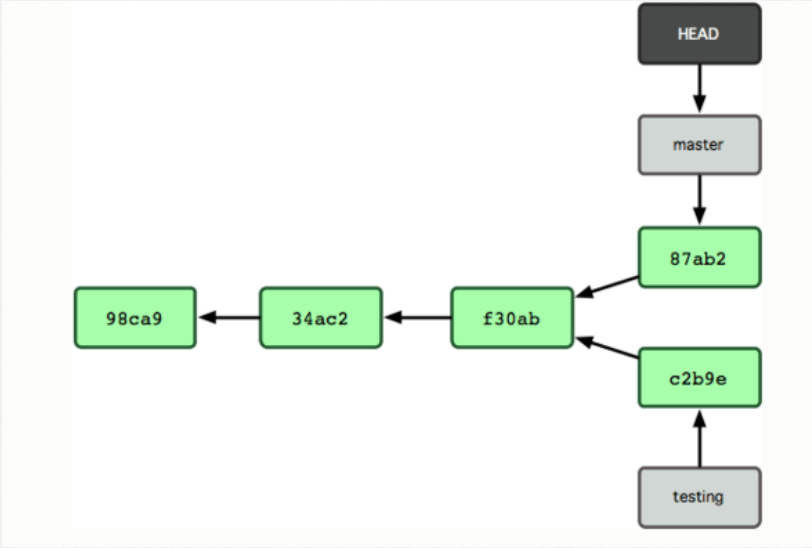


Trabajar de nuevo en master

git checkout master



Si hacemos un nuevo commit: avanza la rama apuntada por HEAD *master



Ceiba software

Crear ramas y cambiar de rama


```
# Para crear una rama nueva en el punto de trabajo
$ git branch <nombre>

# Para crear una rama nueva en un commit dado
$ git branch <nombre> <commit>

# Con checkout -b también crear una rama y cambiarse a ella
$ git checkout -b <nombre>


# Para crear una rama nueva de una rama existente
$ git branch <nombre> <nombreramaexistente>
```

Ceiba software

Detached HEAD Branch

#Es una rama que se crea sobre un commit que no tiene un nombre de rama apuntándole

$ git checkout 34ac2

 # Luego de ubicarnos en el commit podemos crear una rama a partir de este

$ git checkout -b <nombre_rama>

Ceiba software

Fusionar ramas(merge)

#Para incorporar en la rama actual los cambios realizados en otra rama
#internamente GIT analiza la historia de commits para calcular como hacer la mezcla

git merge <rama>

$ git checkout master # Nos ubicamos la rama a la cual llevaremos los cambios

$ merge develop #indicamos las rama de la cual deseamos traer los cambios

Ceiba software

Ceiba
software

Conflictos

#Al hacer el merge pueden aparecer conflictos las dos ramas han modificado las mismas líneas de un fichero.

#Si hay conflictos:

- no se realiza el commit
- las zonas conflictivas se marcan

```
<<<<<<< HEAD:index.html
<div id="footer">contact : email.support@github.com</div>
=======
<div id="footer">contact us at support@github.com</div>
>>>>>>> iss53:index.html
```

```
$ git status   # lista los ficheros con conflictos como unmerged.
```

#Para resolver el conflicto :

- editar el fichero para resolver el conflicto.
- y ejecutar git add y git commit.

Ceiba software

Borrar Ramas

#Una ver terminado el trabajo con una rama

 # la borraremos con

$ git branch -d <nombre>

 # Se elimina el puntero al commit.

#Si la rama a borrar no ha sido mezclada con la rama Actual se muestra un mensaje de error y no se borra.

Para borrar la rama independientemente de si ha sido mezclada o no, usar la opción -D en vez de -d.

$ git branch -D <nombre>

Ceiba software

Repositorios remotos

Adicionar un repositorio remoto

 # Para crear un remote se usa el comando git remote add shortname URL

 # shortname es el nombre corto que damos al remote

 # URL es la URL del repositorio remoto

!

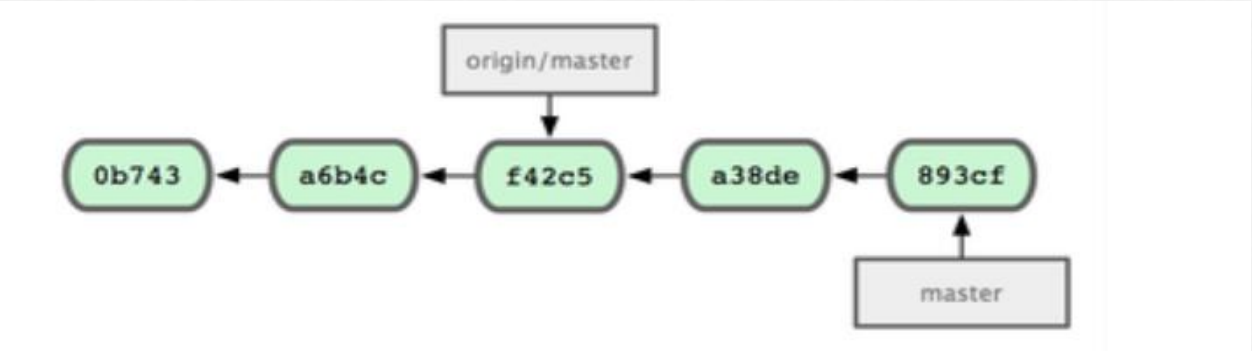
$ git remote add origin <urlrepo>

 # Borrar un remote: git remote rm nombre_del_remote

Ramas remotas

#Una rama remota es un puntero a un commit. Indica cual era la posición de una rama en un repositorio remoto la última vez que nos conectamos con él.

#Se nombran como: <remote>/<rama>.



✦Este puntero no lo podemos mover manualmente.

★ Se mueve cuando actualizamos con el repositorio remoto

Ceiba software

Tracking Branch

#Es una rama local emparejada con una rama remota

para que estén sincronizadas y hacer un seguimiento de los cambios realizados en ellas

#git checkout -b <branchname> <remotename>/<branchname>

$ git checkout -b master origin/master

#git checkout --track <remotename>/<branchname>

$ git checkout --track origin/master

Descargar datos de un remote

```
# Bajarse los datos de un remoto git fetch

# Este comando actualiza el repositorio con los datos existentes en el remote,
# pero no modifica los ficheros del directorio de trabajo.
# Las operaciones de merge las deberemos invocar explícitamente.


# Bajarse los datos que aun no tengo del repositorio del que me cloné:
$ git fetch origin demo
!
# Ahora mezclo mi rama actual con la rama demo de origin:
$ completar
$ git merge origin/demo
```

Descargar datos y mezclar

```
# Bajarse los datos de un remoto y aplicar merge:
git pull nombre_del_remote

# Si la rama actual es una tracking branch:
# El comando git pull [nombre_del_remote], actualiza la rama actual con los
# cambios realizados en la rama asociada del repositorio remoto.

$ git pull origin # Actualiza rama actual con los cambios en origin.
$ git pull # Por defecto se realiza un pull de origin.
!
# Este comando ejecuta un fetch con los argumentos dados, y después realiza
# un merge en la rama actual con los datos descargados.
```

Subir datos a un remote

De forma general, el comando para subir cambios a un remote es:git push nombremote

El comando git push [nombre_del_remote], sube los cambios desarrollados en

la rama local actual a la rama asociada del repositorio remoto.

$ git push origin master # Subir los cambios en la rama master local a origin.

$ git push origin # Subir los cambios de la rama local actual o origin.

$ git push # Por defecto el remote es origin.

La operación push sólo puede realizarse si:

- Se tiene permiso de escritura en el repositorio remoto.
- Nadie ha subido nuevos cambios al repositorio remoto, es decir,# estamos actualizados.
- Si en el remote hay actualizaciones que no tenemos, deberemos hacer un pull antes de poder subir nuestros cambios.
- No pueden subirse actualizaciones que puedan producir conflictos.

Subir datos a un remote

De forma general, el comando para subir cambios a un remote es:git push nombremote

El comando git push [nombre_del_remote], sube los cambios desarrollados en

la rama local actual a la rama asociada del repositorio remoto.

$ git push origin master # Subir los cambios en la rama master local a origin.

$ git push origin # Subir los cambios de la rama local actual o origin.

$ git push # Por defecto el remote es origin.

La operación push sólo puede realizarse si:

- Se tiene permiso de escritura en el repositorio remoto.
- Nadie ha subido nuevos cambios al repositorio remoto, es decir,# estamos actualizados.
- Si en el remote hay actualizaciones que no tenemos, deberemos hacer un pull antes de poder subir nuestros cambios.
- No pueden subirse actualizaciones que puedan producir conflictos.

Tags

#Se usan para etiquetar commits importantes de la historia (o ficheros importantes).

-Para consultar los tags existentes:

★ **git tag # lista todos los tags existentes**

★ **git tag -l <patron> # lista los tags que encajan con el patron dado.**

★ **Ejemplo:**

$ git tag -l v1.*

v1.0

v1.1

v1.2

Crear Tags

#Crear un tag ligero: git tag <nombre> [<commit>]

- Asigna al commit dado un nombre de tag

-Si no se proporciona un commit, se asigna al último commit.

- Se suele usar para etiquetar temporalmente un commit.

#Crear tag anotado: git tag -a <nombre> [<commit>]

-Se crea un nuevo commit para el tag con toda la información que tienen los commit: mensaje, autor del tag, fecha,checksum, etc.

- Ejemplo:

$ git tag -a v1.4 -m "Version 1.4 de la aplicacion" 345ab1ac

-Crea un tag anotado con el nombre v1.4 y el mensaje dado para el commit 345ab1ac

- Si no especificamos la opción -m se lanzará un editor

Compartir Tags

#El comando git push NO transfiere automáticamente los tag creados localmente a los repositorios remotos.

Para copiar un tag local en un repositorio remoto hay que indicarlo explícitamente en el comando push:

```
$ git push origin [tagname]
```

#Usando la opción --tags se transfieren todos los tags que existen en nuestro repositorio local.

```
$ git push origin --tags
```

Bibliografía

http://git-scm.com/book/en/v2

http://chris.beams.io/posts/git-commit