Tu socio tecnológico

nunsys®

COMUNICACIONES · SISTEMAS · SOFTWARE · MARKETING · FORMACIÓN

GOLDCAR rental

Sesión formación Git

16 de marzo 2016

Orden del día

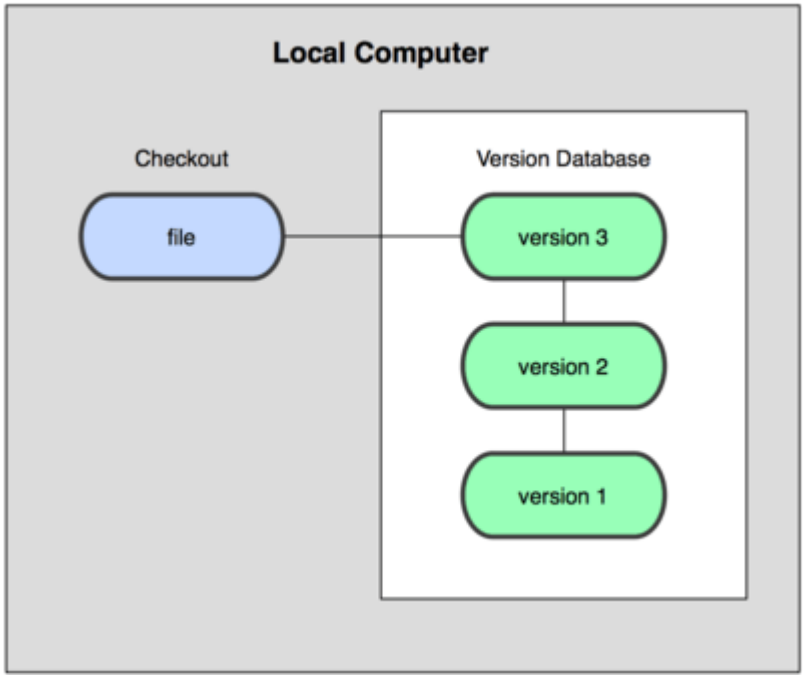
- Introducción
- Instalación y configuración
- Comandos básicos
- Zona de Stash
- El puntero HEAD
- Gestión de ramas
- SourceTree
- Integrando ramas – Merge vs Rebase
- Trabajando con repositorios remotos
- Gitflow
- Migración de un repositorio Subversion

Introducción

"El control de versiones es un sistema que registra los cambios realizados sobre un archivo o conjunto de archivos a lo largo del tiempo, de modo que puedas recuperar versiones específicas más adelante. "

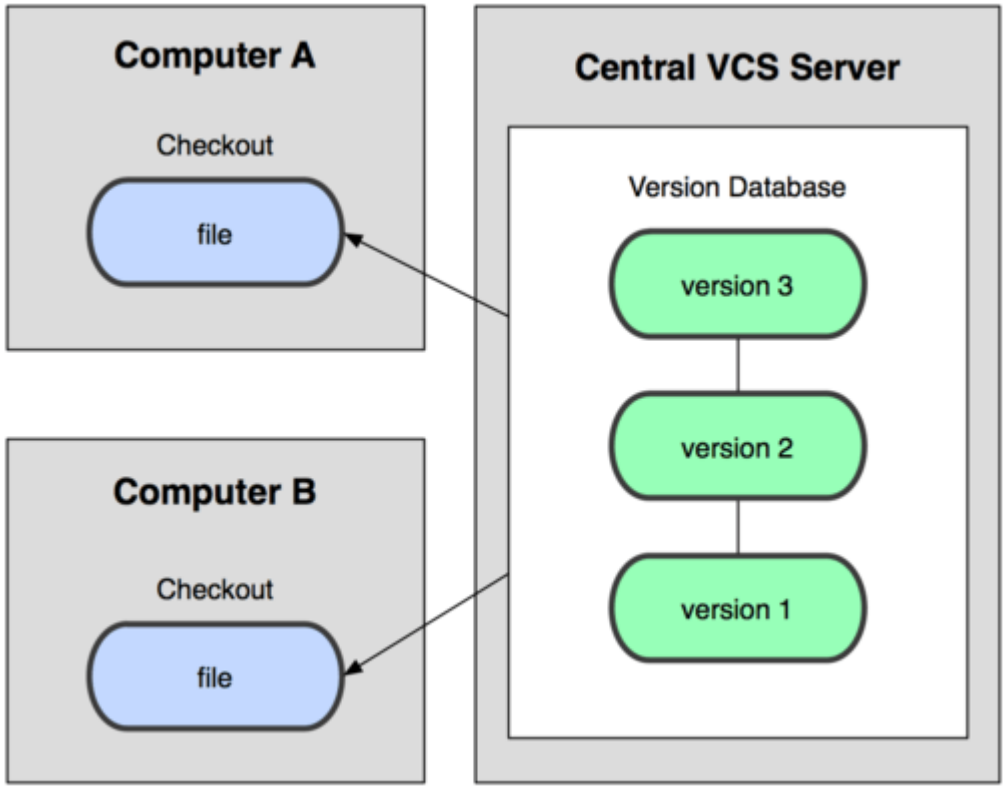
https://git-scm.com/book/es/v1/Empezando-Acerca-del-control-de-versiones

Introducción – Sistema local



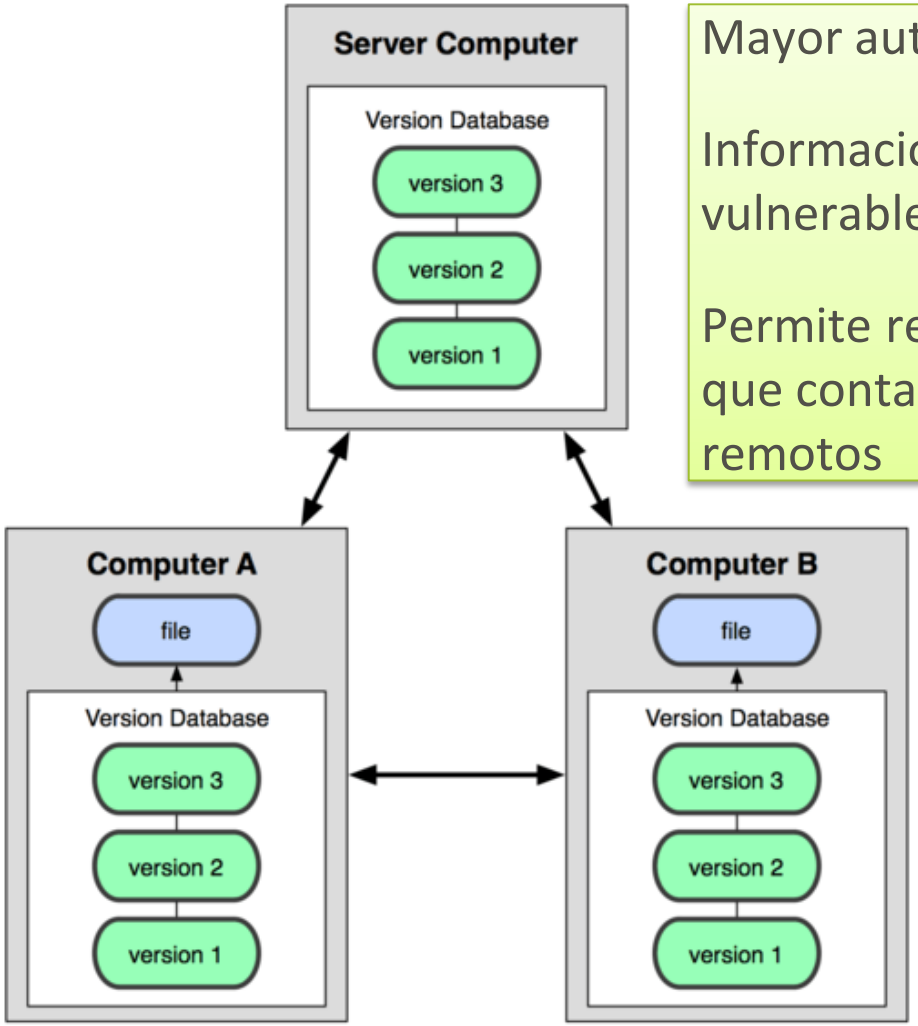
Sistema no colaborativo

Introducción – Sistema centralizado



Dependencia con un servidor central

Introducción – Sistema distribuido



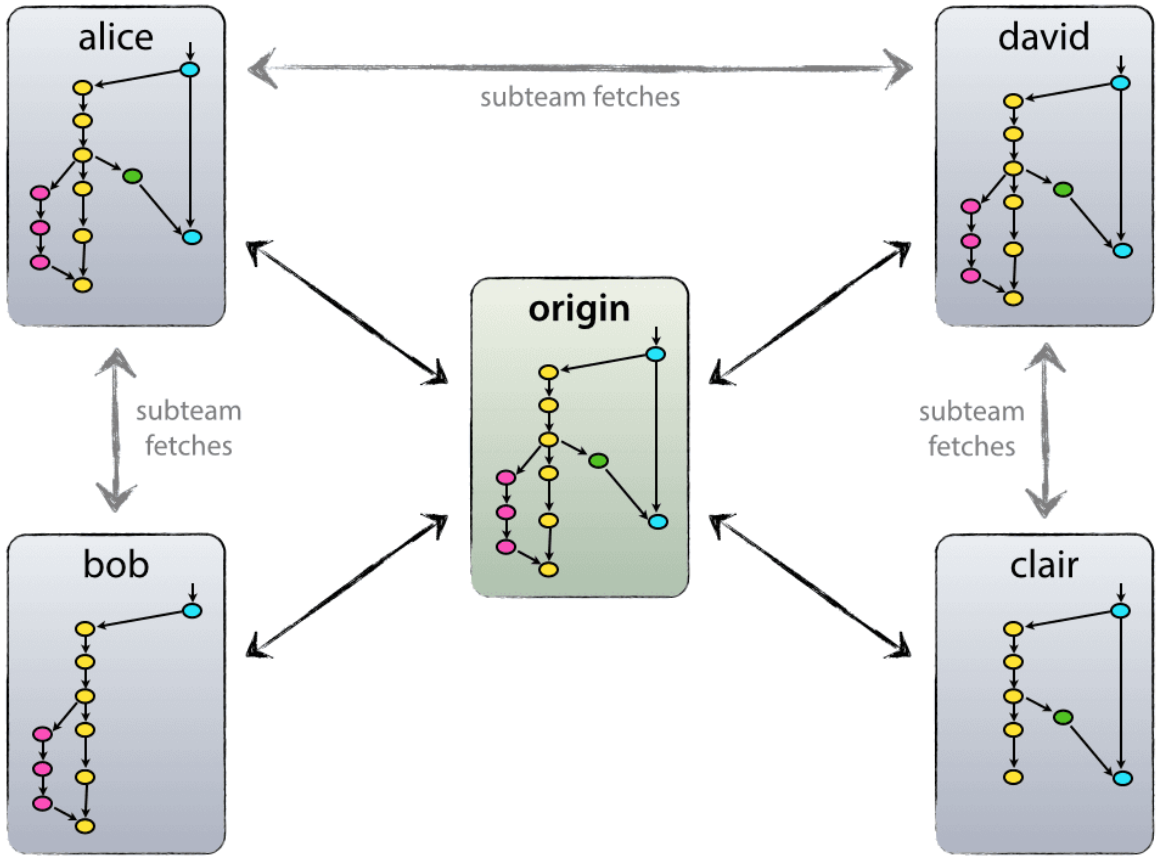
Mayor autonomía

Información replicada, sistema menos vulnerable

Permite realizar pruebas locales sin tener que contaminar el/los servidor (es) remotos

Introducción – Sistema distribuido

Sistema distribuido pero centralizado



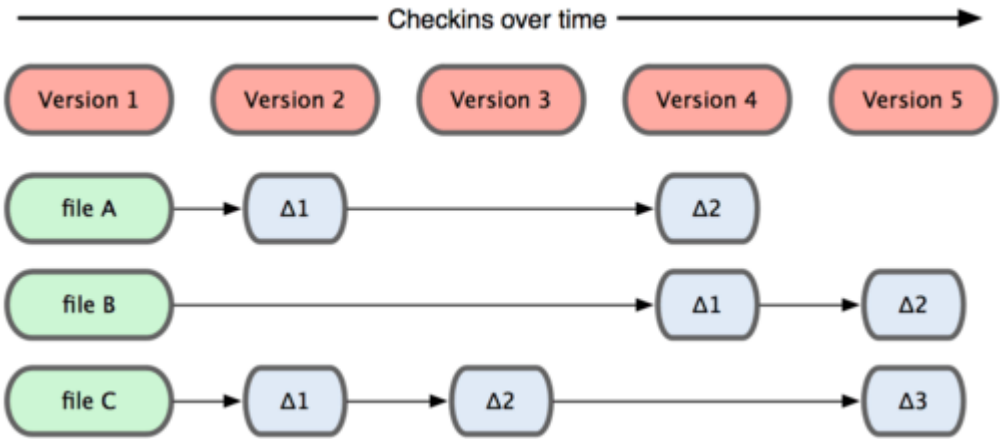
Introducción – Características de Git

- Creado por Linux Torvalds

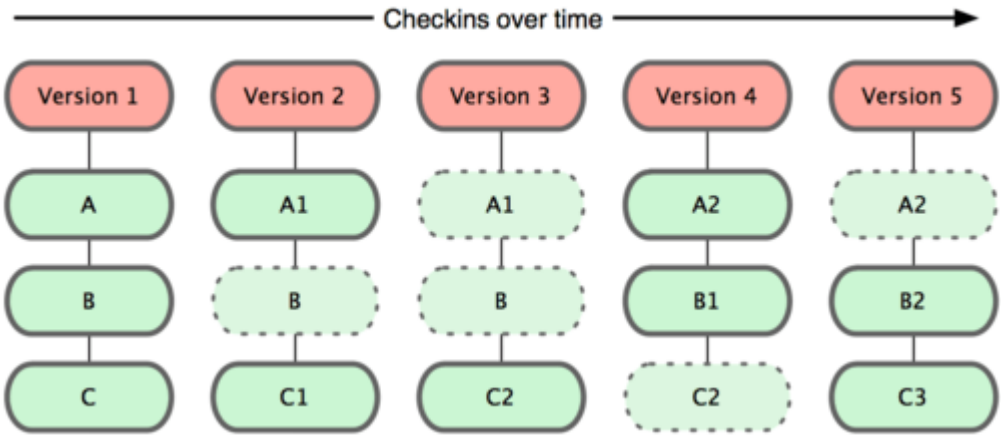
- Objetivos iniciales:
  - Rápido
  - Sencillo
  - Multi rama
  - Distribuido
  - Grandes proyectos

Introducción – Características de Git

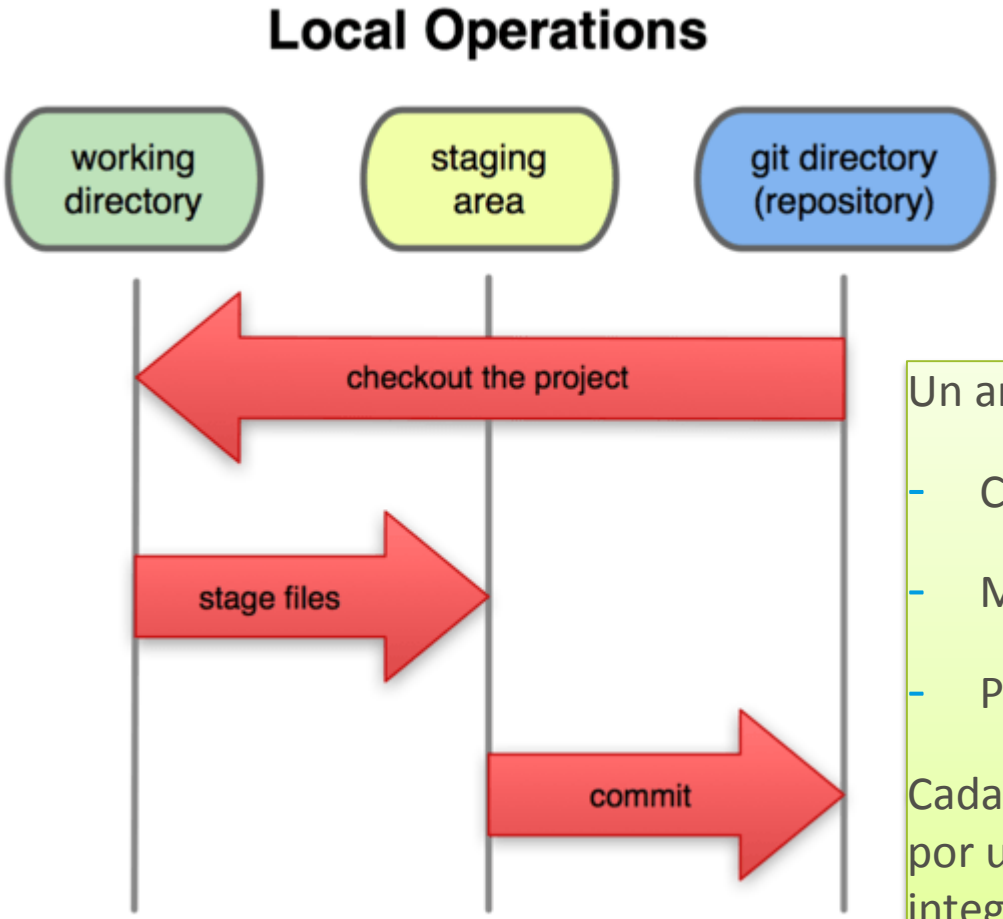
Subversion



Git



Introducción – Características de Git



Un archivo puede estar en tres estados:

- Confirmado (committed)
- Modificado (modified)
- Preparado (staged)

Cada versión de un archivo se identifica por un hash SHA-1, esto garantiza la integridad del contenido del repositorio.

Instalacíón y configuración

Instalación

Windows: https://git-scm.com/download/win

Mac: https://git-scm.com/download/mac

Linux: https://git-scm.com/download/linux

Configuración inicial

```
git config --global user.name "Mi nombre"
git config --global user.email "miemail@dominio.xx"
```

Comprobar la configuración

```
git config --list
```

Comandos básicos

Obtener ayuda

```
git help <comando>
git <comando> --help
```

Crear un repositorio

```
git init
```

Añadir ficheros al área de staging

```
git add fichero.txt
git add *.txt
git add .
```

Commit

```
git commit –m "Añadimos fichero de ejemplo"
```

Añadir cambios al anterior commit

```
git commit --ammend
```

Dos comandos en uno add+commit

```
git commit –am "Añadimos fichero de ejemplo"
```

Comandos básicos

Para ver el histórico de los commit

```
git log
git log --oneline
git log -p
git log -p -2
git log --decorate --graph --oneline --all
```

Para ver el contenido de un commit

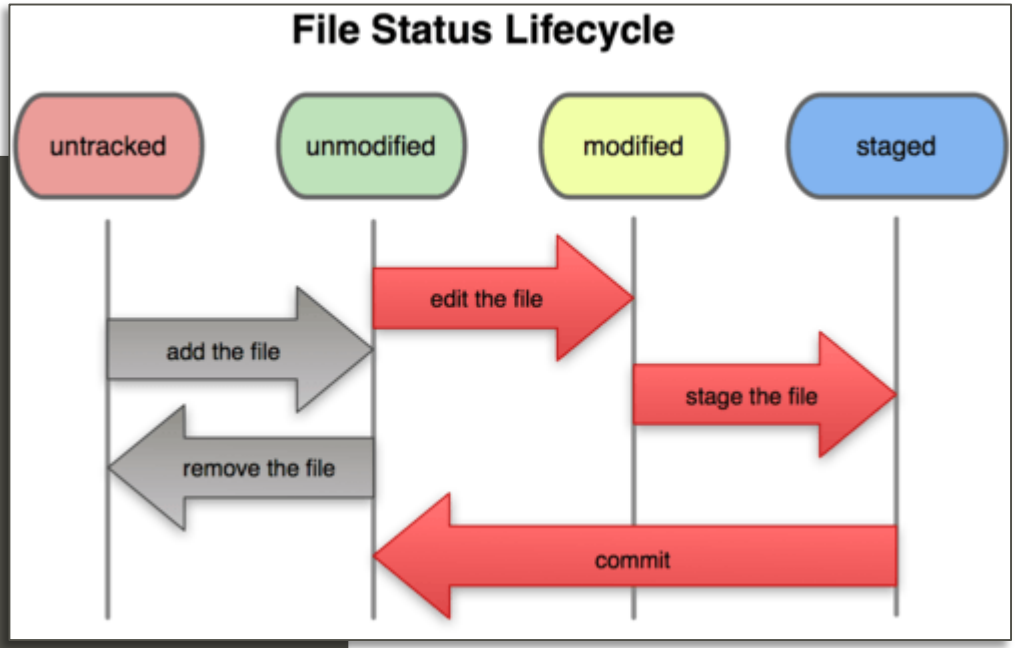
```
git show <id commit>
```

```
$ git log --oneline
67badad Mi primer commit


$ git show 67badad
commit 67badad588ea470a36d232208b5cf7b9360a8a51
Author: Manuel Alagarda <manuel.alagarda@nunsys.com>
Date:   Wed Mar 16 13:21:31 2016 +0100

    Mi primer commit

diff --git a/fichero01.txt b/fichero01.txt
new file mode 100644
index 0000000..e69de29
```

Comandos básicos

Para ver el estado de los ficheros

```
git status
```




```
$ git add fichero01.txt
$ git commit -m "Mi primer commit"
[master (root-commit) 67badad] Mi primer commit
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 fichero01.txt
$ git add fichero02.txt
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   fichero02.txt
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       fichero03.txt
```

Comandos básicos

Para ver las diferencias entre el área de staging y el área de trabajo

```
git diff
```

```
$ git log --oneline
67badad Mi primer commit


$ git diff 67badad
diff --git a/fichero02.txt b/fichero02.txt
new file mode 100644
index 0000000..b2af2e8
--- /dev/null
+++ b/fichero02.txt
@@ -0,0 +1 @@
+Cambios de <FA>ltima hora
\ No newline at end of file
```

Comandos básicos

Para deshacer los cambios de uno o varios ficheros

```
git checkout <nombre fichero>
```

Para sacar un fichero de la zona de staging (mantiene los cambios en los ficheros)

```
git reset <nombre fichero>
```

Para deshacer todos los cambios (excepto ficheros nuevos que no están en staging)

```
git reset --hard
```

Para crear etiquetas

```
git tag <nombre etiqueta>
git tag -a <nombre etiqueta> -m "Mi descripción"
```

Para crear etiquetas tardías si conocemos el hash del commit

```
git tag -a <nombre etiqueta> <id commit> -m "Mi descripción"
```

Para eliminar etiqueta

```
git tag -d <id commit>
```

Zona de stash

Según se está trabajando en un apartado de un proyecto, normalmente el espacio de trabajo suele estar en un estado inconsistente. Pero puede que se necesite cambiar de rama durante un breve tiempo para ponerse a trabajar en algún otro tema urgente. Esto plantea el problema de confirmar cambios en un trabajo medio hecho, simplemente para poder volver a ese punto más tarde. Y su solución es el comando 'git stash'.

```
git stash
```

Para ver el contenido de la pila de stash

```
git stash list
```

Para recuperar

```
git stash apply
git stash apply <id stash>
```

Para recuperar y eliminar el último elemento de la pila

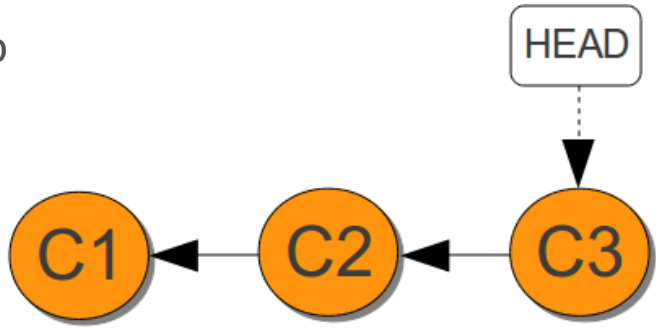
```
git stash pop
```

El puntero HEAD

El puntero HEAD nos indica en que rama y revisión estamos posicionados en el repositorio LOCAL.

Para saber donde esta posicionado el puntero

```
git log --decorate --graph --oneline
git branch -v
```



Para eliminar los dos últimos commits

```
git reset HEAD~2
```

Para hacer un checkout del anterior commit al actual

```
git checkout HEAD~1
```

Para mostrar el contenindo de un commit anterior

```
git show HEAD~1
```

Para fusionar commits que aún no hemos publicado en un repositorio remoto

```
git reset --soft HEAD~6
git commit -m 'mensaje de commit'
```

.gitignore

En este fichero podremos configurar que archivos y carpetas deben ser excluidos del control de versiones, de esta forma podremos evitar por ejemplo que los ficheros temporales que se generan durante una compilación de un proyecto se incluyan en el repositorio.

https://github.com/github/gitignore

Gestión de ramas

Una rama GIT es simplemente un puntero. La rama por defecto de GIT es la rama ‘master‘.

Para **ver** todas las ramas que tenemos en el repositorio local :

```
git branch
```

Cuando **creamos** nosotros una nueva rama, GIT crea un nuevo puntero para que lo puedas gestionar y el comando de creación de una rama es:

```
git branch <nombre de la rama>
```

Al ejecutar el comando anterior lo que hemos hecho es simplemente crear la rama pero seguimos estando en ‘master’. Para **movernos entre ramas**, disponemos del comando:

```
git checkout <nombre de la rama>
```

Para realizar las dos anteriores acciones en un único comando:

```
git checkout –b <nombre de la rama>
```

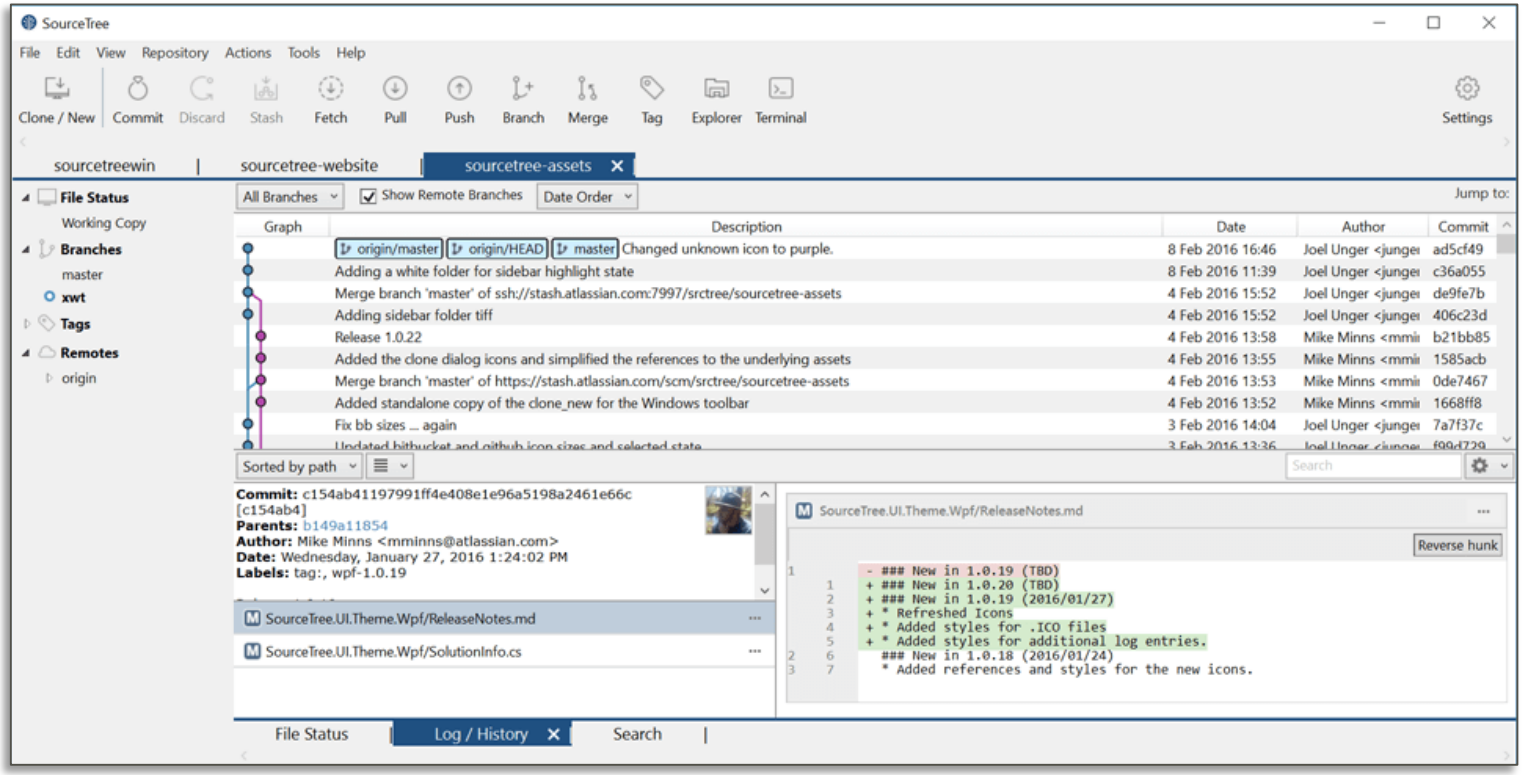
Para **eliminar** una rama

```
git branch –D <nombre de la rama>
```

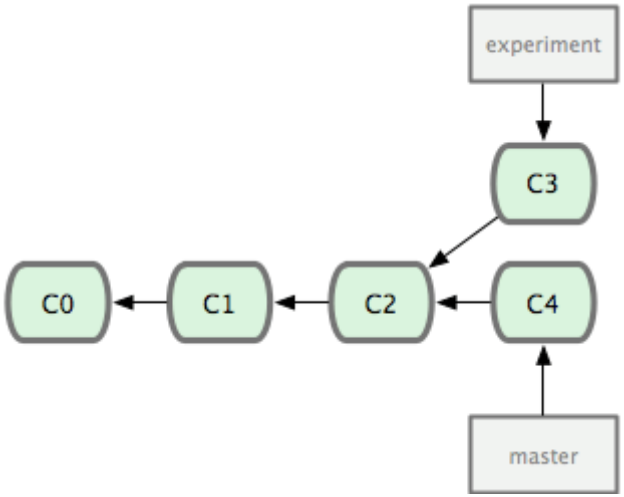
SourceTree

Entorno visual desarrollado por Altassian para gestionar repositorios Git

https://www.sourcetreeapp.com/

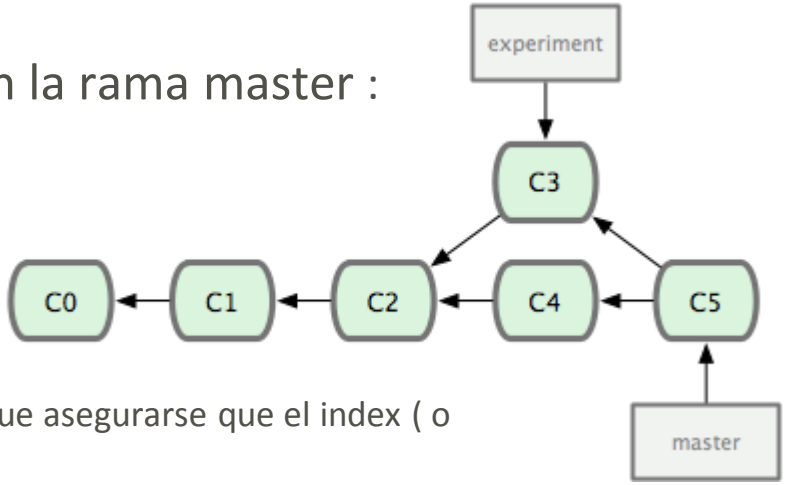


Integrando ramas - Merge



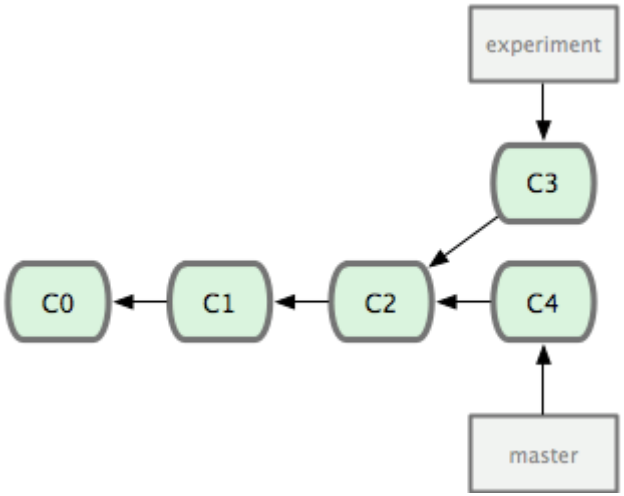
Para integrar la rama experiment en la rama master :

```
git checkout master
git merge experiment
```



¡IMPORTANTE! Antes de hacer un merge hay que asegurarse que el index (o Stagin área está limpio

Integrando ramas - Rebase



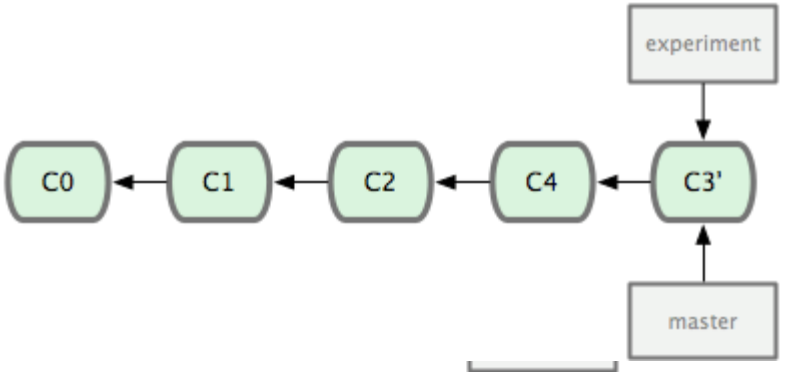
Para integrar la rama experiment en la rama master :

```
git checkout experiment
git rebase master
```

En caso de conflicto podemos:

```
git rebase --continue
git rebase --abort
```

Aconsejable para hacer integración continua



Integrando ramas – Resolución de conflictos

Podemos hacer uso de cualquiera de las herramientas que hay disponibles para resolución de conflictos:

- Kdiff3
- Meld
- WinMerge
- …

Vincular la herramienta de merge con git mediante el comando:

```
git config --global merge.tool kdiff3
```

Lanzar a ejecución la herramienta de resolución de conflictos:

```
git mergetool
```

Trabajando con repositorios remotos

Clonar un repositorio remoto:

```
git clone git://github.com/schacon/grit.git
```

Clonar un repositorio remoto con otro nombre local:

```
git clone git://github.com/schacon/grit.git <nombre_local>
```

Actualizar desde el repositorio remoto:

```
git pull
```

Obtener la lista de ramas remotas:

```
git ls-remote
```

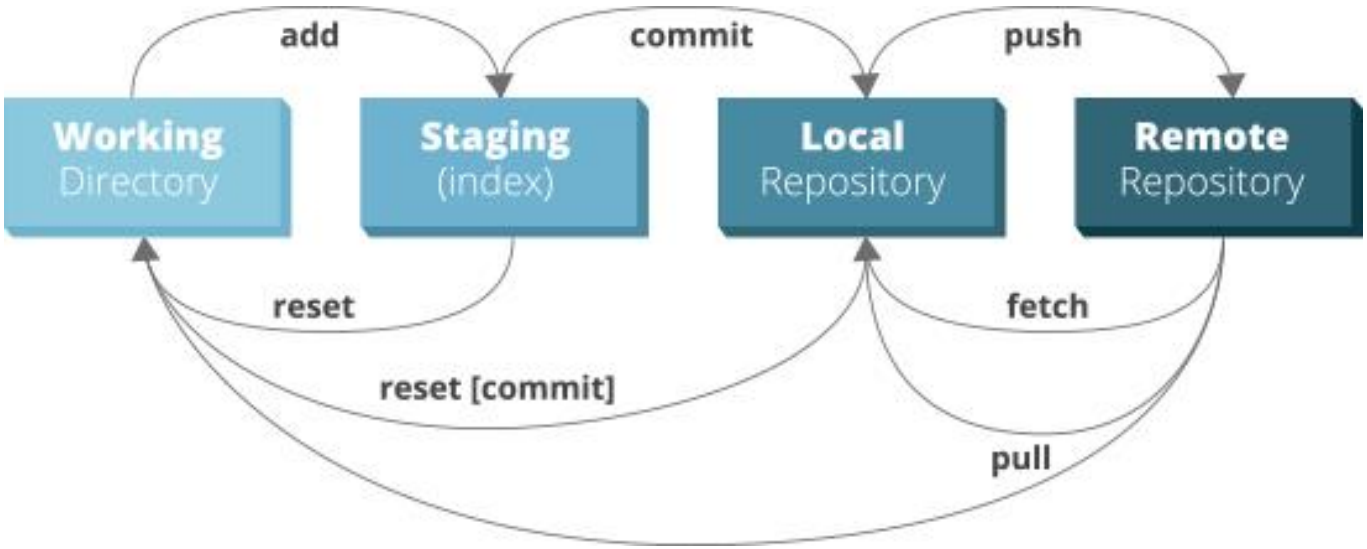
Para descargar una rama remota

```
git checkout –track –b <nombre_rama_remota>
```

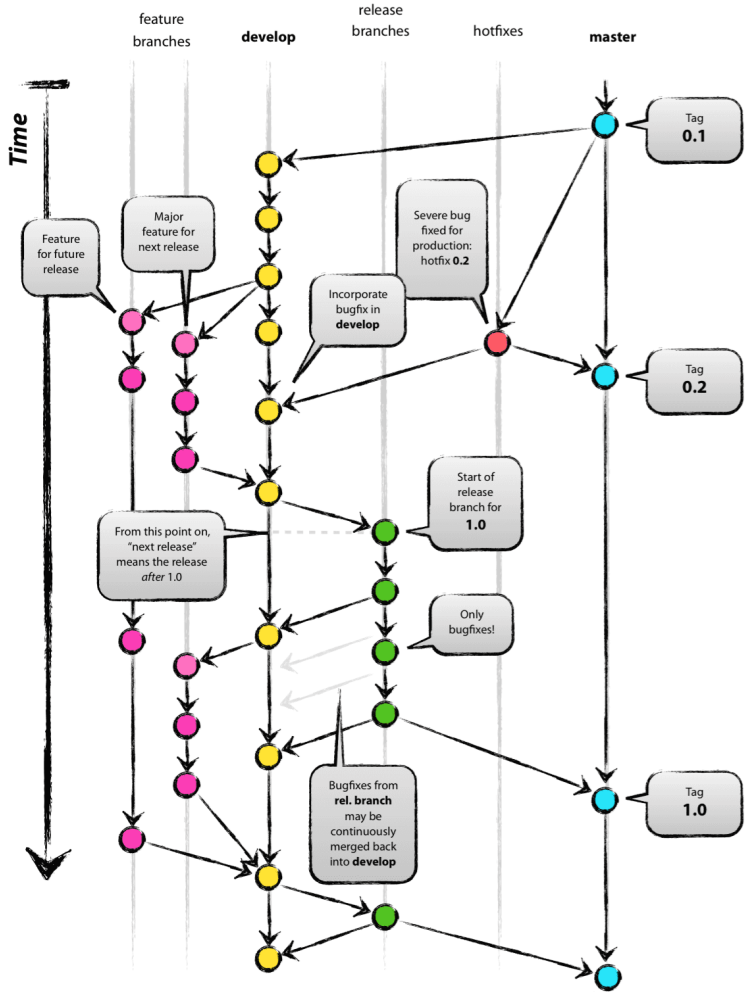
Para actualizar el repositorio remoto con los cambios locales

```
git push
```

Resumen operaciones



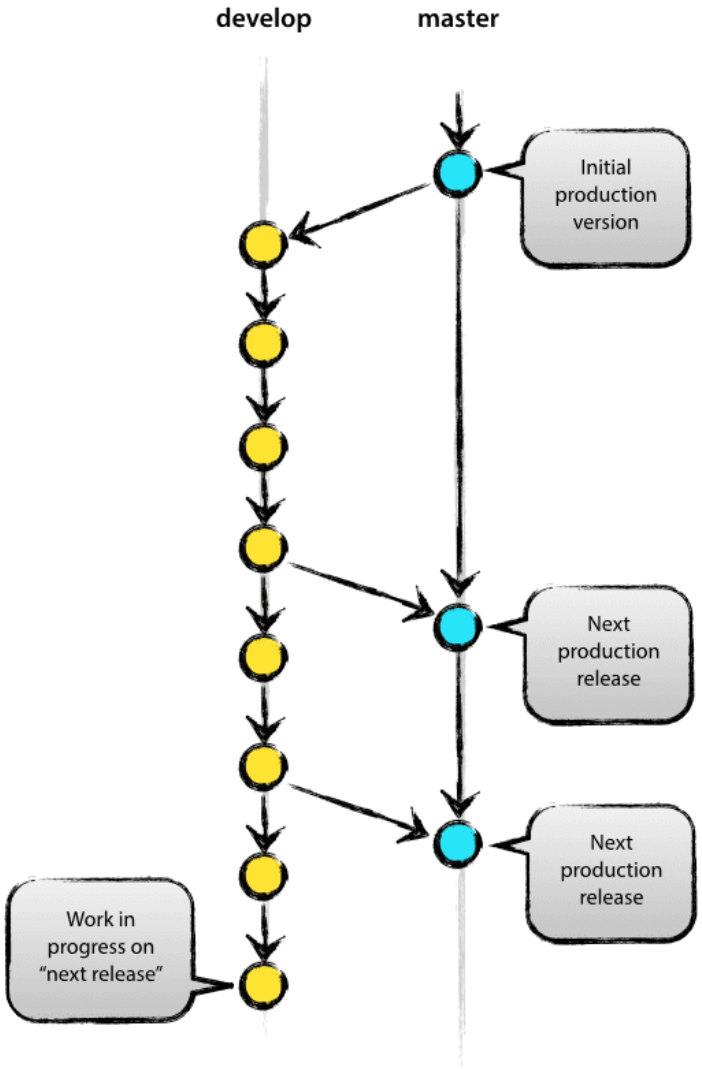
GitFlow



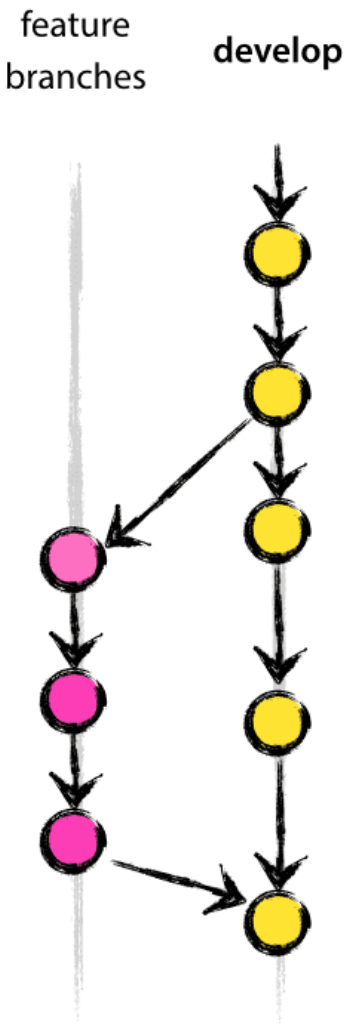
Git Flow es una estrategia de gestión de las ramas definida en 2010 por Vincent Driessen que se ha convertido en un estándar.

Se puede ver aquí el blog original: http://nvie.com/posts/a-successful-git-branching-model/

GitFlow – Ramas principales



GitFlow – Creando una feature



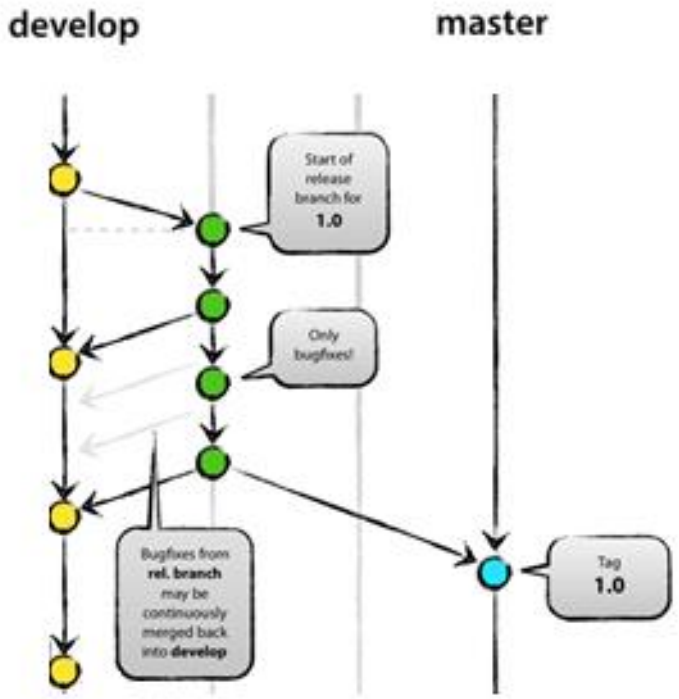
Creación

```
git checkout -b myfeature develop
```

Finalización

```
git checkout develop
git merge --no-ff myfeature
git branch -d myfeature
git push origin develop
```

GitFlow – Creando una release



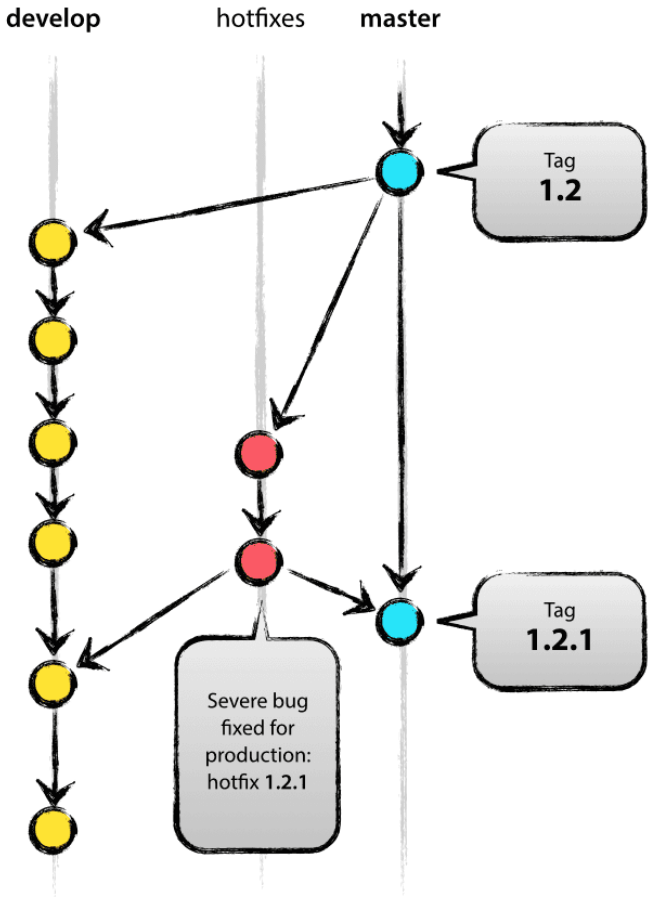
Creación

```
git checkout -b release-1.2 develop
```

Finalización

```
git checkout develop
git checkout master
git merge --no-ff release-1.2
git tag -a 1.2

git checkout develop
git merge --no-ff release-1.2

git branch -d release-1.2
```

GitFlow – Creando un hotfix



Creación

```
git checkout -b hotfix-1.2.1 master
./bump-version.sh 1.2.1
git commit -a -m "Bumped version number to 1.2.1"
```

Finalización

```
git checkout master
git merge --no-ff hotfix-1.2.1
git tag -a 1.2.1

git checkout develop
git merge --no-ff hotfix-1.2.1
```