

# Git

## Control de versiones distribuido



## Agenda



- Tipos de sistemas de control de versiones
- Conceptos básicos
- Comandos
- Branching en Git
- Herramientas

The background of the slide is a dense, repeating pattern of white, three-dimensional spheres. Each sphere has a subtle hexagonal or faceted texture, giving it a crystalline appearance. The spheres are arranged in a staggered grid, creating a sense of depth and perspective. A solid green horizontal banner is positioned in the lower third of the image, containing the title text in white. The banner has a small triangular tab extending downwards from its left edge.

## Tipos de sistemas de control de versiones

# ¿Qué es el control de versiones?



**Gestión de cambios sobre un archivo o conjunto de archivos a lo largo del tiempo.**

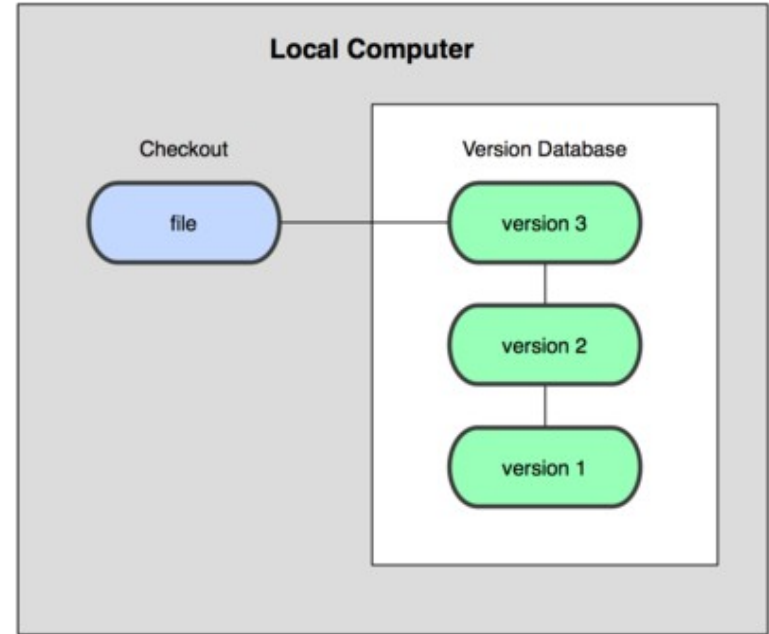
## Permite

- Recuperar versiones de un archivo.
- Consultar el histórico de cambios.
- Comparación entre versiones.
- ...

# Tipos de sistemas de control de versiones

## Sistemas de control local

- Base de datos local (rcs)

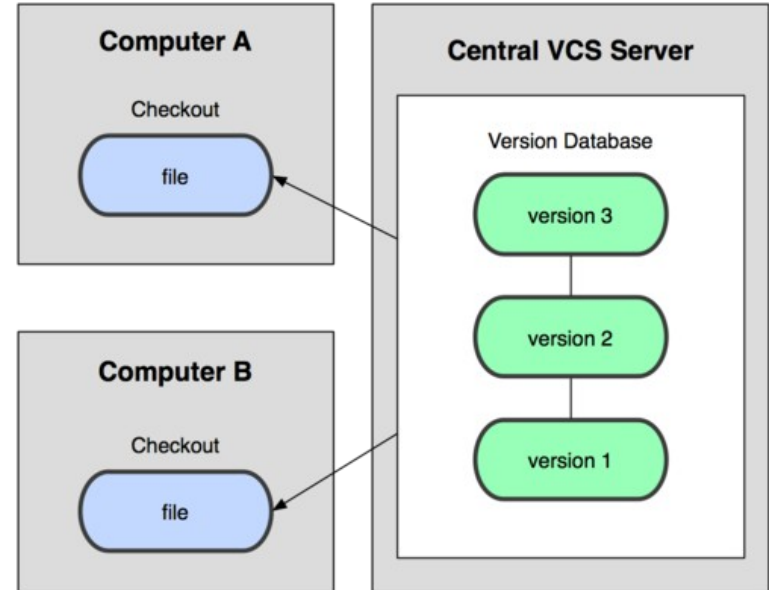


# Tipos de sistemas de control de versiones

## Sistemas de control de versiones centralizados

- Servidor central + n clientes
- Fomentan la colaboración entre desarrolladores.
- Mejora en la comunicación de cambios.
- Dependencia con un único nodo central.
- Caída = imposibilidad de subir cambios

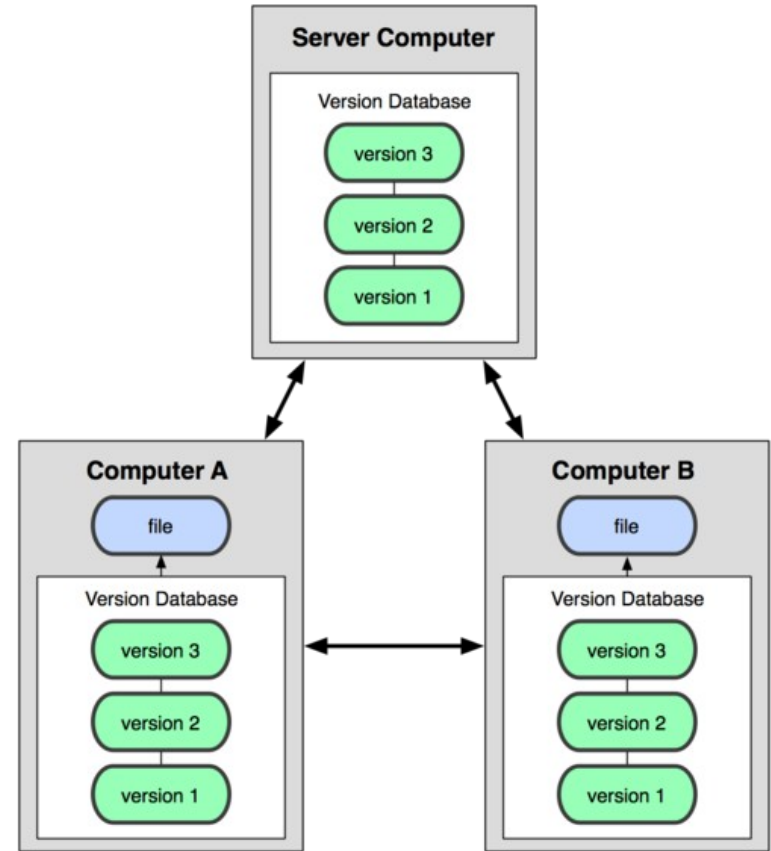
CVS



# Tipos de sistemas de control de versiones

## Sistemas de control de versiones distribuidos

- Copia completa del servidor.
- Muerte de servidor -> n copias distribuidas (backup).
- Colaboración entre desarrolladores sin pasar por servidor central (modelos jerárquicos)



# Conceptos básicos





# Git : un poco de historia

- Kernel de linux (1991-2002) : Cambios = parches + archivos
- A partir de 2002 : DVCS BitKeeper
- 2005 : Bitkeeper deja de ser gratuito
  - Comunidad diseña su propia herramienta de control de versiones : Git

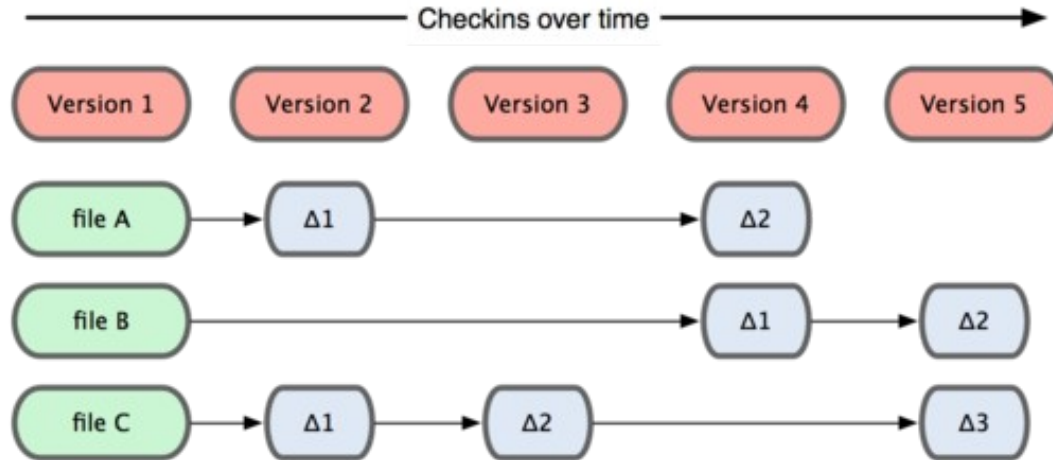
## Requisitos

- Velocidad
- Diseño sencillo
- Fuerte apoyo al desarrollo no lineal (miles de ramas paralelas)
- Completamente distribuido
- Capaz de manejar grandes proyectos como el núcleo de Linux de manera eficiente (velocidad y tamaño de los datos)



# Fundamentos de Git

- Instantáneas, no diferencias
- Otros sistemas : información = archivo + modificaciones

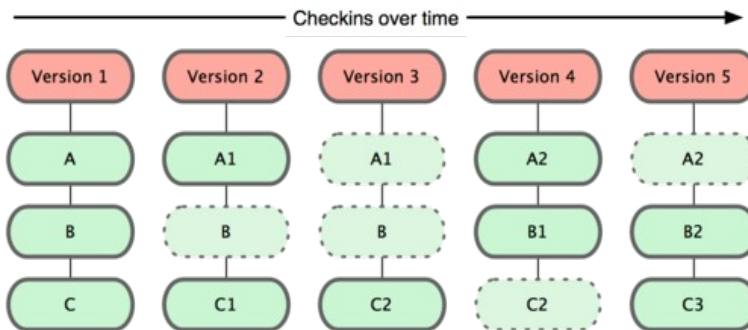


# Fundamentos de Git

- Git : foto en cada momento.
- Si no hay cambio en un fichero se guarda un enlace a la versión anterior (> eficiencia)
- Operaciones locales (ej: histórico de cambios)
- Trabajo en modo avión
- Tiene integridad = hash SHA-1

Ej. : objeto --> 24b9da6552252987aa493b52f8696cd6d3b00373

No se almacena por nombre, se hace por hash de su contenido



# Git : los tres estados

## Confirmado (committed)

**Almacenado** en base de datos

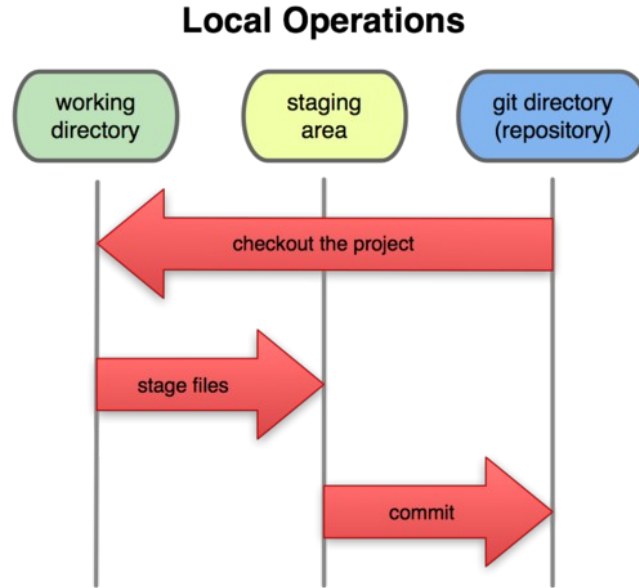
## Modificado (modified)

Actualizado pero **no confirmado**  
en base de datos

## Preparado (staged)

**Preparado** para ser añadido en la  
**siguiente confirmación.**

# Git : los tres estados



- Directorio de git (.git) = metadatos + base de datos de objetos
- Directorio de trabajo = copia de una version del proyecto
- Área de preparación = fichero con los cambios a incluir en el siguiente commit



Comandos

# Configuración de tu identidad

```
$ git config --global user.name "Joselin Cognito Face"
$ git config --global user.email joselin.cognito@iecisa.com

$ git config --list
user.name=Joselin Cognito Face
user.email=joselin.cognito@iecisa.com
core.repositoryformatversion=0
core.filemode=true
core.logallrefupdates=true
core.autocrlf=false
```

# Ayuda

```
$ git help <comando>  
$ git <comando> --help  
$ man git-<comando>
```



# Obtener un repositorio

- Inicializar un repositorio en un directorio existente

```
$ git init
```

*Crea un directorio .git con todos los metadatos necesarios pero sin ningún fichero en seguimiento*

- Clonar un repositorio existente

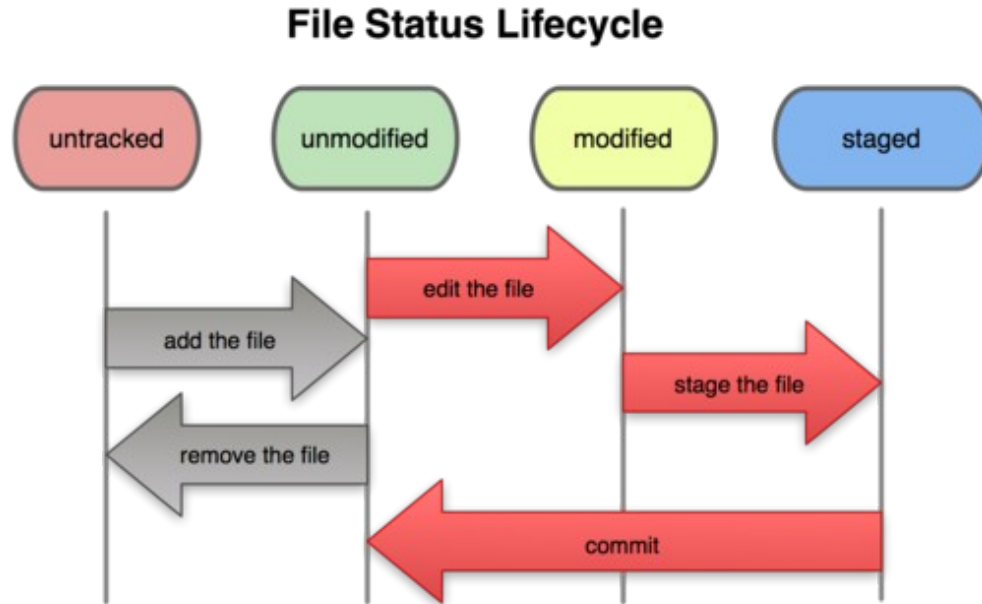
```
$ git clone https://scm.iecisa.com/foo.git
```

*Copia en directorio foo y extrae de la base de datos una copia de la ultima versión.*

*clone <> checkout (se recibe una copia completa)*

# Guardando cambios

- Un archivo puede estar bajo seguimiento (**tracked**) o sin seguimiento (**untracked**)
- Un archivo en seguimiento puede estar sin modificar, modificado o en preparación



## Comprobando el estado de los archivos

- Supongamos un repositorio recién clonado

```
$ git status
# On branch master
nothing to commit (working directory clean)
```

- Se crea un nuevo fichero

```
$ touch README
$ git status
# On branch master
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#   README
nothing added to commit but untracked files present (use "git add" to track)
```

# Seguimiento de nuevos ficheros

```
$ git add README

$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   README
#
```

## Preparando archivos modificados

- Se modifica un archivo que estaba en seguimiento (benchmarks.rb)

```
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   README
#
# Changed but not updated:
#   (use "git add <file>..." to update what will be committed)
#
#       modified:   benchmarks.rb
#
```

*El archivo se ha modificado pero no esta preparado  
(área de preparación)*

# Preparando archivos modificados

```
$ git add benchmarks.rb
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   README
#       modified:   benchmarks.rb
#
```

# Visualizando qué ha cambiado en un archivo

```
$ git diff
```

*modificado pero no  
preparado*

```
$ git diff --cached | git diff --staged
```

*modificado y que irá en  
la proxima confirmacion*

## Confirmando cambios

```
$ git commit      (solicita mensaje de confirmacion.  
                  Por defecto, salida de git status)  
  
$ git commit -m "Story 182: Fix benchmarks for speed")  
[master]: created 463dc4f: "Fix benchmarks for speed"  
2 files changed, 3 insertions(+), 0 deletions(-)  
create mode 100644 README
```

*git commit --> foto del sistema*



## Saltándose el área de preparación

```
$ git status
# On branch master
#
# Changed but not updated:
#
#   modified:   benchmarks.rb
#

$ git commit -a -m 'added new benchmarks'
[master 83e38c7] added new benchmarks
1 files changed, 5 insertions(+), 0 deletions(-)
```

# Eliminando archivos

- Eliminar = eliminar de seguimiento + confirmar

```
$ git rm <fichero>
```

- Si se elimina de disco directamente....

```
$ rm grit.gemspec
$ git status
# On branch master
#
# Changed but not updated:
#   (use "git add/rm <file>..." to update what will be committed)
#
#       deleted:    grit.gemspec
#
```

# Eliminando archivos

- Ejecutando **git rm** se prepara la eliminación

```
$ git rm grit.gemspec
rm 'grit.gemspec'

$ git status
# On branch master
#
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       deleted:    grit.gemspec
#
```

# Moviendo archivos

## Sintaxis

```
$ git mv file_from file_to
```

```
$ git mv README.txt README
$ git status
# On branch master
# Your branch is ahead of 'origin/master' by 1 commit.
#
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       renamed:    README.txt -> README
#
```

Esto es equivalente a :

```
$ mv README.txt README
$ git rm README.txt
$ git add README
```

# Viendo el histórico de confirmaciones

```
$ git log
    commit ca82a6dff817ec66f44342007202690a93763949
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Mon Mar 17 21:52:11 2008 -0700

    changed the version number

commit 085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Sat Mar 15 16:40:33 2008 -0700

    removed unnecessary test code

commit allbef06a3f659402fe7563abf99ad00de2209e6
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Sat Mar 15 10:31:28 2008 -0700

    first commit
```

# Viendo el histórico de confirmaciones

```
$ git log -p (modificaciones en cada commit)
$ git log -p -2
$ git log --stat (numero de ficheros modificados, añadidos y eliminados)
$ git log --pretty=<formato>
```

<formato>=oneline, short, full, fuller, format

```
$ git log --format
```

Opción	Descripción de la salida
%H	Hash de la confirmación
%h	Hash de la confirmación abreviado
%T	Hash del árbol
%t	Hash del árbol abreviado
%P	Hashes de las confirmaciones padre
%p	Hashes de las confirmaciones padre abreviados
%an	Nombre del autor
%ae	Dirección de correo del autor
%ad	Fecha de autoría (el formato respeta la opción `--date`)
%ar	Fecha de autoría, relativa
%cn	Nombre del confirmador
%ce	Dirección de correo del confirmador
%cd	Fecha de confirmación
%cr	Fecha de confirmación, relativa
%s	Asunto

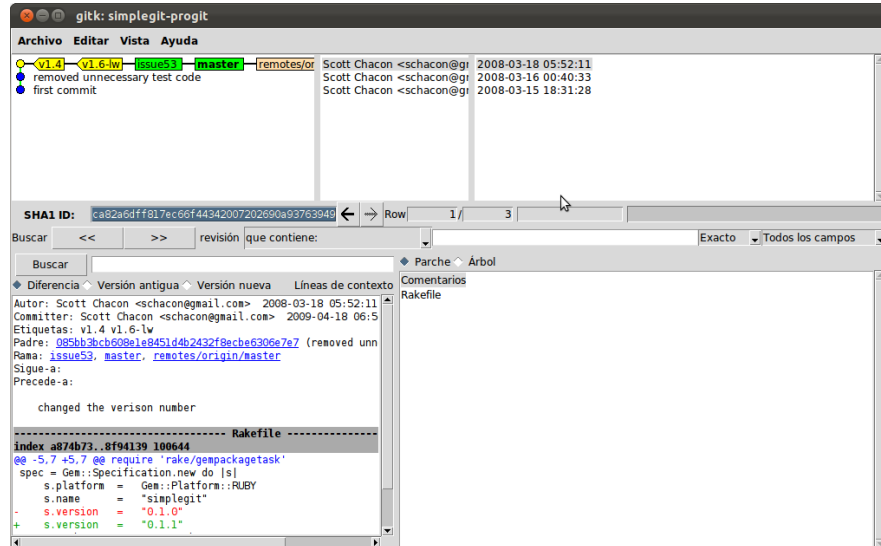
# Viendo el histórico de confirmaciones

```
$ git log --graph
```

Más opciones

```
$ git help log
```

Histórico en formato gráfico (gitk)



# Deshaciendo cambios : último commit

## Sintaxis

```
$ git commit --amend
```

## Ejemplo

```
$ git commit -m 'initial commit'  
$ git add forgotten_file  
$ git commit --amend
```

*se fusiona en un único commit*



# Deshaciendo cambios : volviendo a un commit

## Sintaxis

```
$ git reset [options] [<commit>]
```

## Ejemplo

```
$ git log --pretty=oneline
15027957951b64cf874c3557a0f3547bd83b3ff6 Merge branch 'experiment'
a6b4c97498bd301d84096da251c98a07c7723e65 beginning write support
0d52aaab4479697da7686c15f77a3d64d9165190 one more thing

$ git reset --hard 0d52aaab4479697da7686c15f77a3d64d9165190
```

## Eliminando del área de preparación

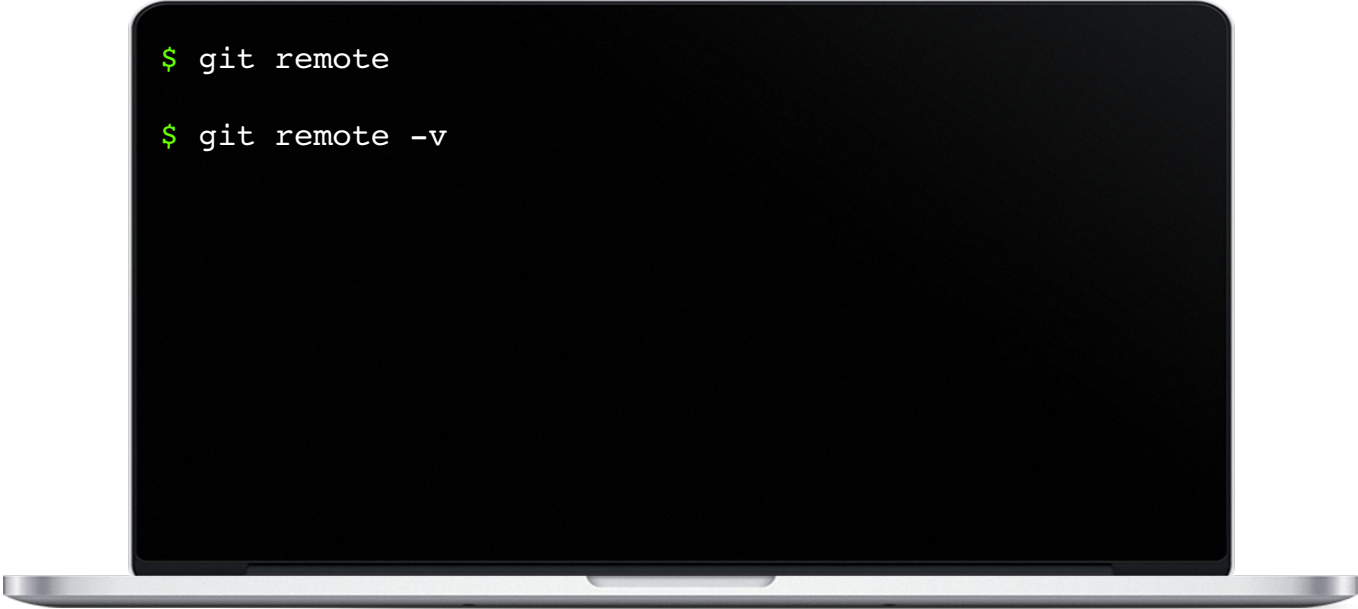
```
$ git add .
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   README.txt
#       modified:   benchmarks.rb
```

- Deshaciendo la modificación de un archivo

```
$ git checkout -- benchmarks.rb
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   README.txt
#
$ git reset HEAD benchmarks.rb
```

# Trabajo con repositorios remotos

- Colaboración entre desarrolladores, con repositorio central...
- enviar (**push**) o recibir (**pull**)



```
$ git remote  
  
$ git remote -v
```

# Añadiendo nuevos repositorios

## Sintaxis

```
$ git remote add <alias> <url>
```

## Ejemplo

```
$ git remote add pb git://github.com/paulboone/ticgit.git  
$ git remote -v  
origin    git://github.com/schacon/ticgit.git  
pb        git://github.com/paulboone/ticgit.git
```

# Renombrando repositorios remotos

## Sintaxis

```
$ git remote rename <old> <new>
```

## Ejemplo

```
$ git remote rename pb paul  
$ git remote  
origin  
paul
```

# Eliminando repositorios remotos

## Sintaxis

```
$ git remote remove <alias>
```

## Ejemplo

```
$ git remote remove paul  
$ git remote  
origin
```

# Enviando datos a repositorios remotos

## Sintaxis

```
$ git push [nombre-remoto][nombre-rama]
```

## Ejemplo

```
$ git push origin master
```

# Recuperando datos de repositorios remotos : fetch

## Sintaxis

```
$ git fetch [nombre-repository]
```

## Ejemplo

```
$ git fetch origin
```



# Recuperando datos de repositorios remotos : pull

## Sintaxis

```
$ git pull [nombre-remoto][nombre-rama]
```

## Ejemplo

```
$ git pull origin master
```

# Creando etiquetas

- Marcado de una versión

```
$ git tag
v0.1
v1.3

$ git tag -l 'v1.4.2.*'
v1.4.2.1
v1.4.2.2
v1.4.2.3
v1.4.2.4
```

# Creando etiquetas

- Tipos : ligeras y anotadas
  - ligera = rama que no cambia (un puntero)
  - anotada = objeto completo en la base de datos de git
    - Tienen
      - suma de comprobación
      - nombre del etiquetador
      - mail
      - fecha
      - firma GPG (opcional)



```
$ git tag
v0.1
v1.3

$ git tag -l 'v1.4.2.*'
v1.4.2.1
v1.4.2.2
v1.4.2.3
v1.4.2.4
```

# Creando etiquetas

- Creación etiqueta anotada:

```
$ git tag -a v1.4 -m 'my version 1.4'
```

```
$ git tag  
v0.1  
v1.3  
v1.4
```

- Creación etiqueta ligera:

```
$ git tag v1.6
```

*sin -a, s o -m*

# Creando etiquetas

- Etiquetando a posteriori

```
$ git log --pretty=oneline
15027957951b64cf874c3557a0f3547bd83b3ff6 Merge branch 'experiment'
a6b4c97498bd301d84096da251c98a07c7723e65 beginning write support
0d52aaab4479697da7686c15f77a3d64d9165190 one more thing
6d52a271eda8725415634dd79daabbc4d9b6008e Merge branch 'experiment'
0b7434d86859cc7b8c3d5e1dddfed66ff742fcabc added a commit function
4682c3261057305bdd616e23b64b0857d832627b added a todo file
166ae0c4d3f420721acbb115cc33848dfcc2121a started write support
9fceb02d0ae598e95dc970b74767f19372d61af8 updated rakefile
964f16d36dfccde844893cac5b347e7b3d44abbc commit the todo
8a5cbc430f1a9c3d00faaeffd07798508422908a updated readme
```

```
$ git tag -a v1.2 9fceb02
```

Completo o una parte  
de la suma

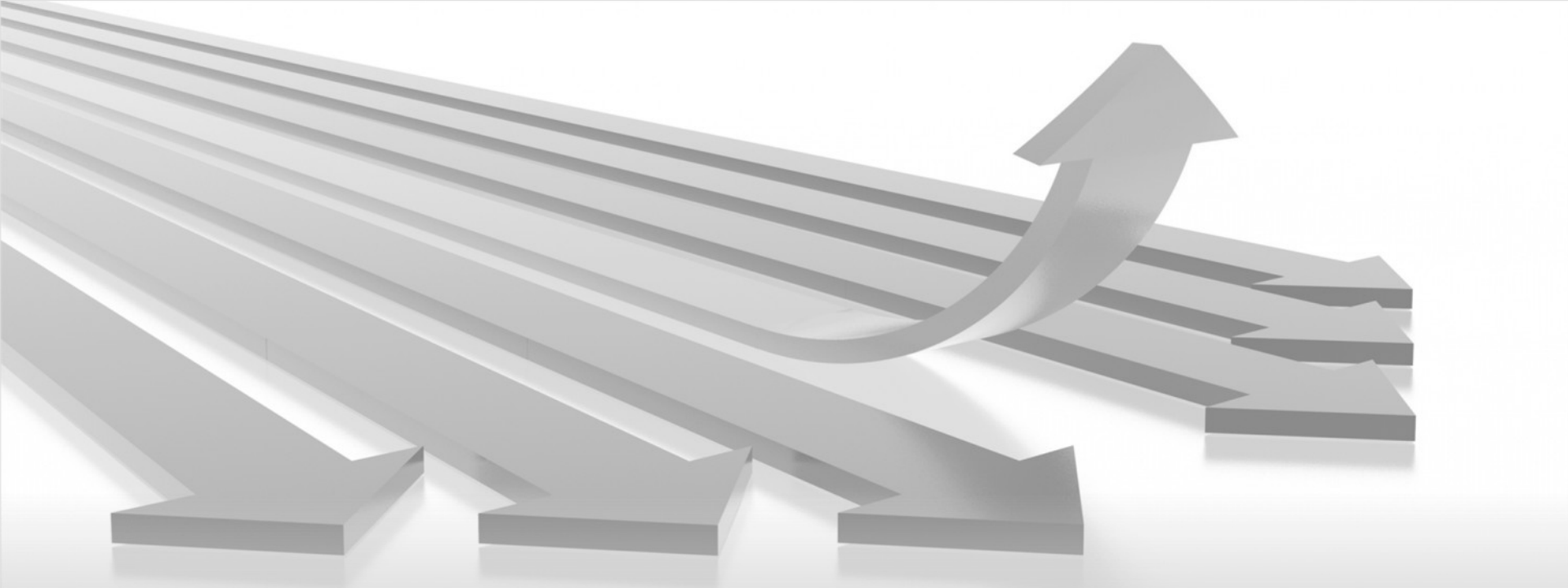
## Creando etiquetas

- Las etiquetas no se comparten al hacer un push. Hay que compartirlas de manera explícita.

```
$ git push origin [tag-name]
```

- si se tienen muchas....

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



## Branching en Git

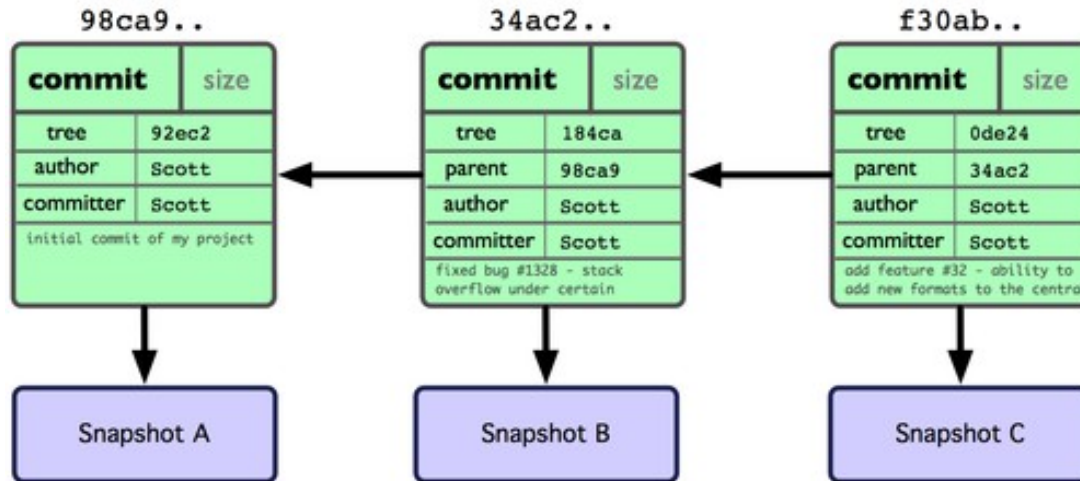
# Branching en Git

- Punto fuerte de Git.
- Destaca por la velocidad y facilidad de uso.
- Mejora el proceso de branch y merge de Subversion.
- Git promueve el uso intensivo de ramas para el desarrollo.



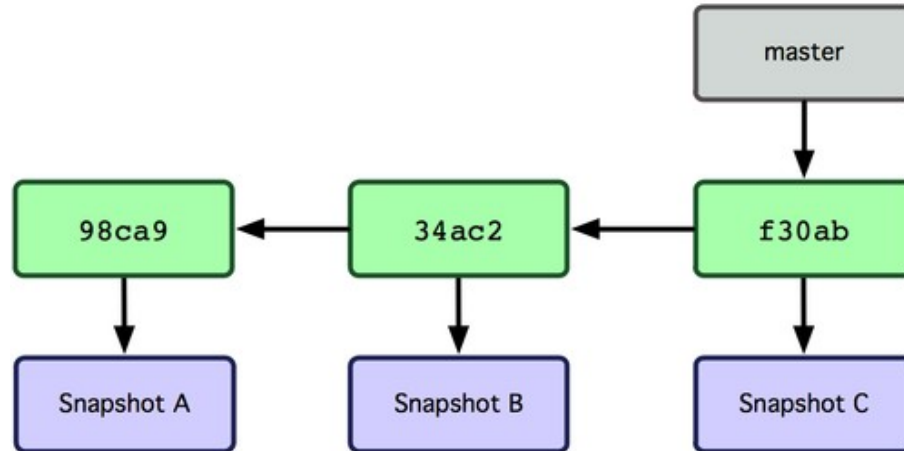
# Branching en Git

- Git no almacena los datos de forma incremental. Almacena una instantanea de los datos en cada confirmación.
- Por cada commit se genera un punto de control que se conserva.



# Branching en Git

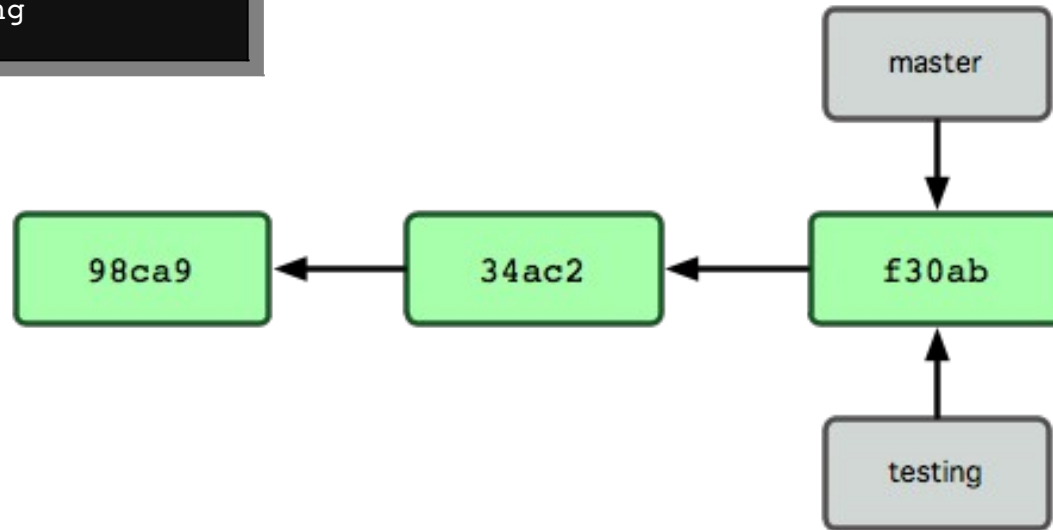
- Una rama es un puntero móvil a una de las confirmaciones (**commit**).
- En Git todo son ramas.
- La rama por defecto se llama **master**.



# Branching en Git

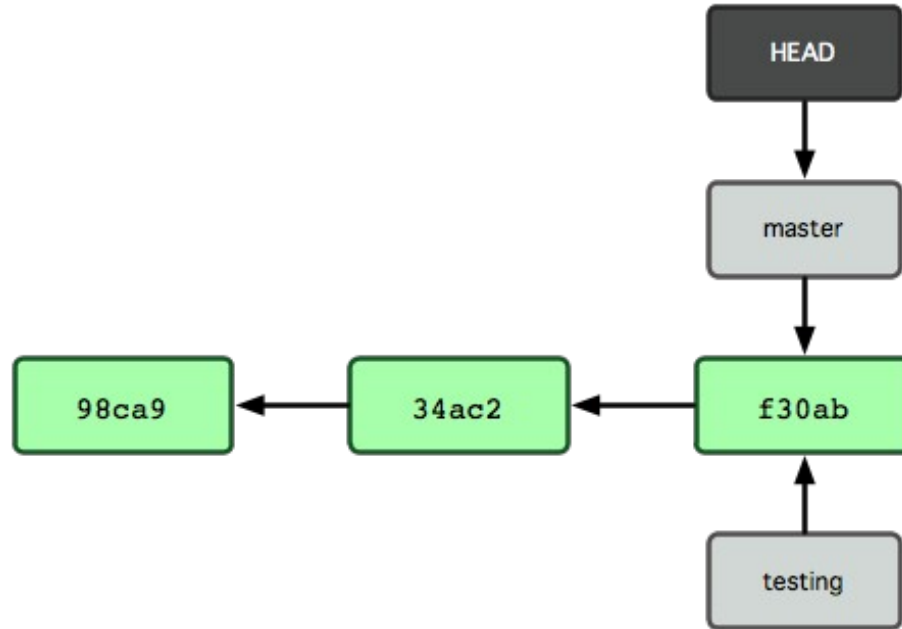
- Con la creación de una rama se crea un nuevo puntero.

```
$ git branch testing
```



# Branching en Git

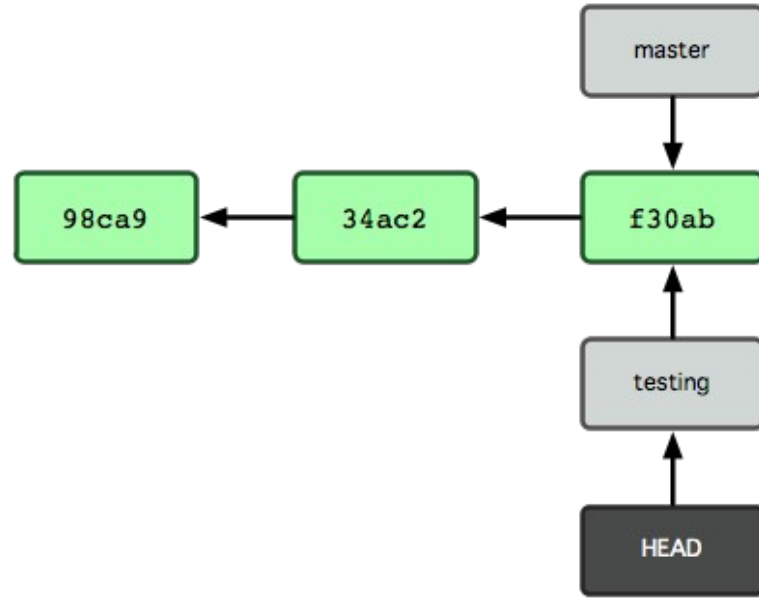
- ¿Y con tanta rama en cuál estoy trabajando? **HEAD**.



# Branching en Git

- Para poder trabajar en una rama hay que saltar a ella. *La creación no lo hace.*

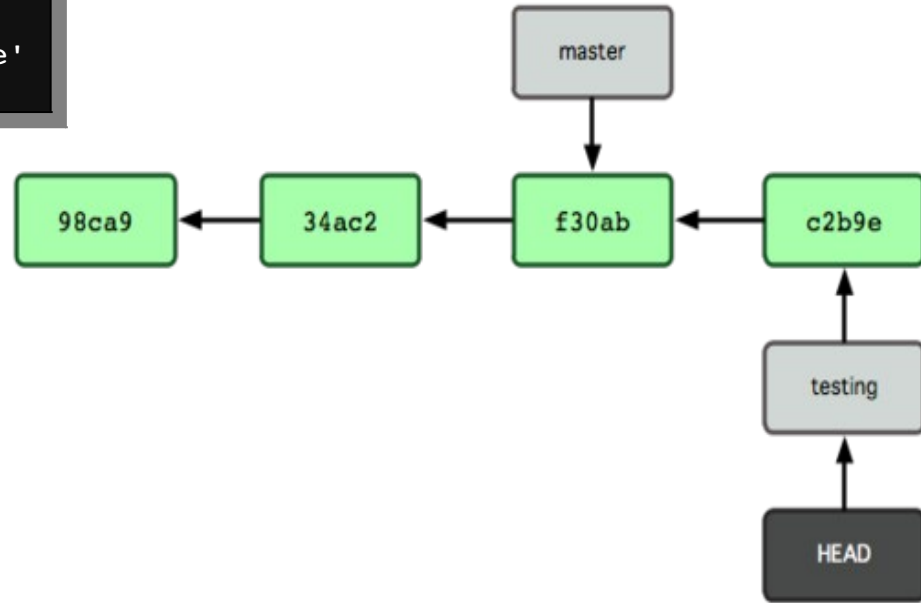
```
$ git checkout testing
```



# Branching en Git

- Las modificaciones hechas a partir de este momento se confirman sobre la nueva rama.

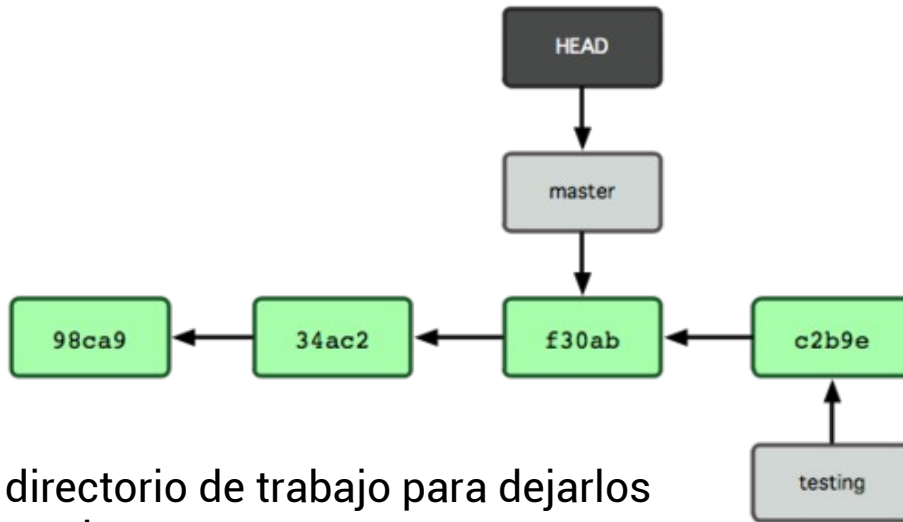
```
$ vim test.rb  
$ git commit -a -m 'made a change'
```



# Branching en Git

- Si volvemos a trabajar sobre la rama master cambiamos el puntero HEAD.

```
$ git checkout master
```

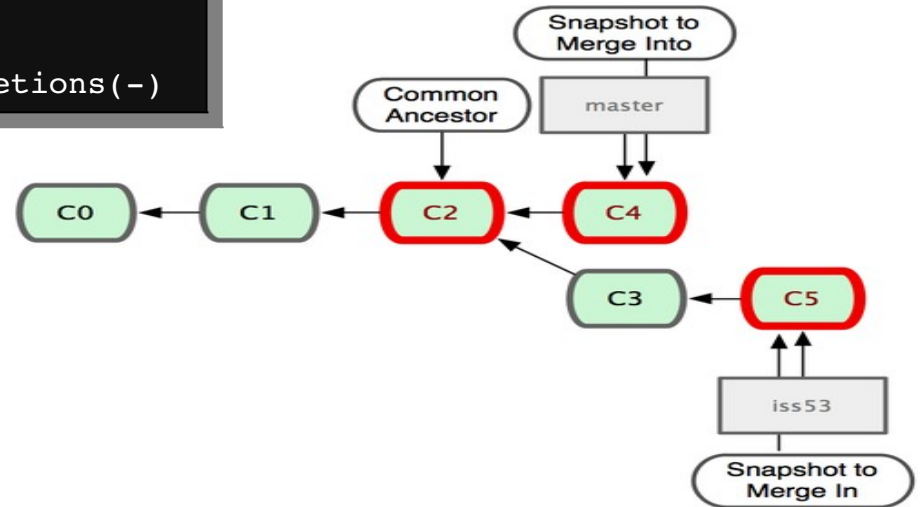


- También se revierten los archivos del directorio de trabajo para dejarlos como estaban en la confirmación apuntada por master.

# Branching en Git

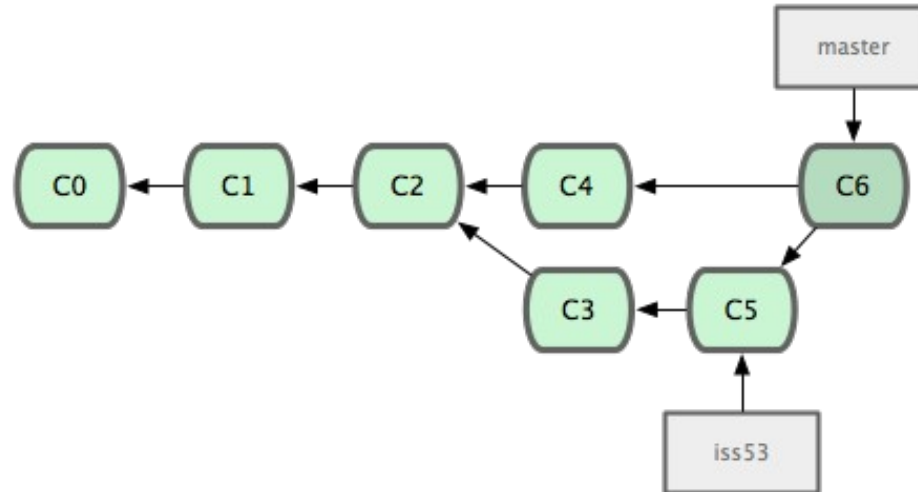
- Una vez que el trabajo sobre una rama ha finalizado ya podemos fusionar los cambios.

```
$ git checkout master
$ git merge iss53
Merge made by recursive.
 README |      1 +
 1 files changed, 1 insertions(+), 0 deletions(-)
```





# Branching en Git



- Ahora ya se puede eliminar la rama

```
$ git branch -d iss53
```

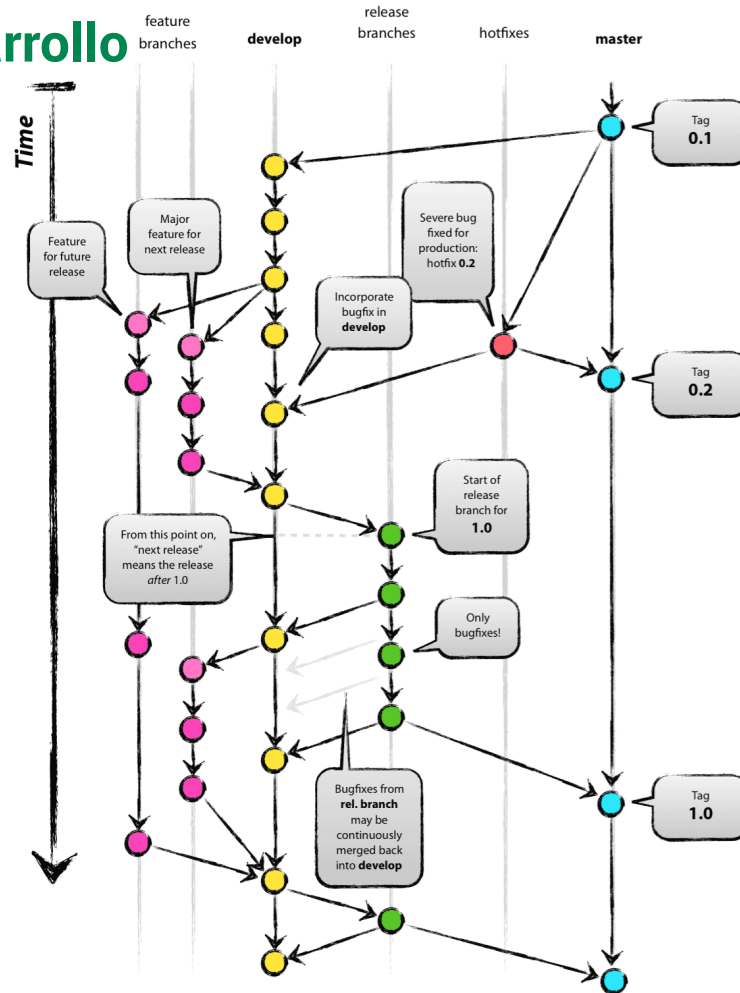
# Branching en Git

- ¿Por qué es tan rápido crear una rama?
  - Rama = fichero de 40 caracteres con el SHA-1 de la confirmación a la que se apunta.
  - Subversion → rama = copia de todos los archivos del proyecto.
- Procedimientos básicos de branching
  - Rama por funcionalidad o user story (feature/issue branching).
  - Rama para bugfixing.
  - Rama para pruebas locales.
  - ...

# Git : modelo de desarrollo

## Ramas principales

- master  
(producción)
- develop (ci)



## Tipos de ramas

- issue/feature  
(Jira id)
- release  
(release-<version>)
- hotfix  
(hotfix-<version>)

# Issue/feature branches

- Ramas locales
- Creadas a partir de *develop*
- Finalización → merge a *develop*
- Convenio de nombrado (*Jira issue id*)
  - IDOC-435
  - VIPGDP-17

```
$ git checkout -b idoc-435 develop
Switched to a new branch "idoc-435"
```

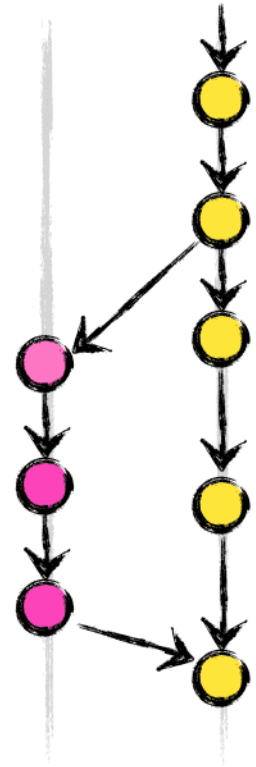
```
$ git checkout develop
Switched to branch 'develop'

$ git merge --no-ff idoc-435
Updating ealb82a..05e9557
(Summary of changes)

$ git branch -d idoc-435
Deleted branch idoc-435 (was 05e9557).

$ git push origin develop
```

feature  
branches      develop



## Release branches

- Creadas a partir de *develop*
- Finalización → merge a *develop* y *master*
- Convenio de nombrado (*release-<version>*)
  - release-1.2
  - release-2.1.3
  - ...

```
$ git checkout -b release-1.2 develop
Switched to a new branch "release-1.2"
$ ./update-version.sh 1.2
Files modified successfully, version updated to 1.2.
$ git commit -a -m "Updated version number to 1.2"
[release-1.2 74d9424] Updated version number to 1.2
1 files changed, 1 insertions(+), 1 deletions(-)
```

```
$ git checkout master
Switched to branch 'master'
$ git merge --no-ff release-1.2
Merge made by recursive.
(Summary of changes)
$ git tag -a 1.2
```

```
$ git checkout develop
Switched to branch 'develop'
$ git merge --no-ff release-1.2
Merge made by recursive.
(Summary of changes)
```

```
$ git branch -d release-1.2
Deleted branch release-1.2 (was ff452fe).
```

# Hotfix branches

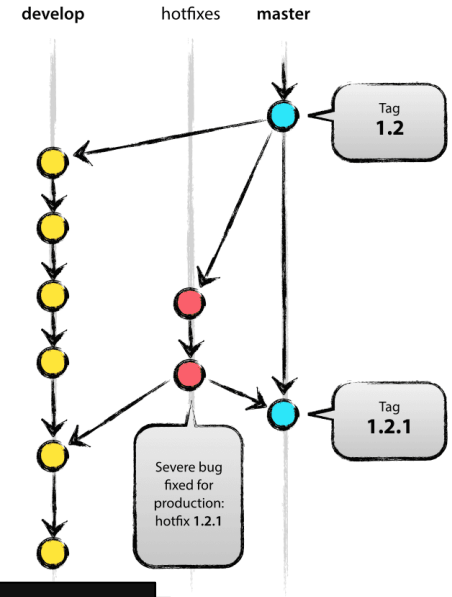
- Creadas a partir de *master*
- Finalización → merge a *develop* y *master*
- Convenio de nombrado (*hotfix-<version>*)

```
$ git checkout -b hotfix-1.2.1 master
Switched to a new branch "hotfix-1.2.1"
$ ./update-version.sh 1.2.1
Files modified successfully, version updated to 1.2.1.
$ git commit -a -m "Updated version number to 1.2.1"
[hotfix-1.2.1 41e61bb] Updated version number to 1.2.1
1 files changed, 1 insertions(+), 1 deletions(-)
```

```
$ git checkout master
Switched to branch 'master'
$ git merge --no-ff hotfix-1.2.1
Merge made by recursive.
(Summary of changes)
$ git tag -a 1.2.1
```

```
$ git checkout develop
Switched to branch 'develop'
$ git merge --no-ff hotfix-1.2.1
Merge made by recursive.
(Summary of changes)
```

```
$ git branch -d hotfix-1.2.1
Deleted branch hotfix-1.2.1 (was abbe5d6).
```



# Herramientas



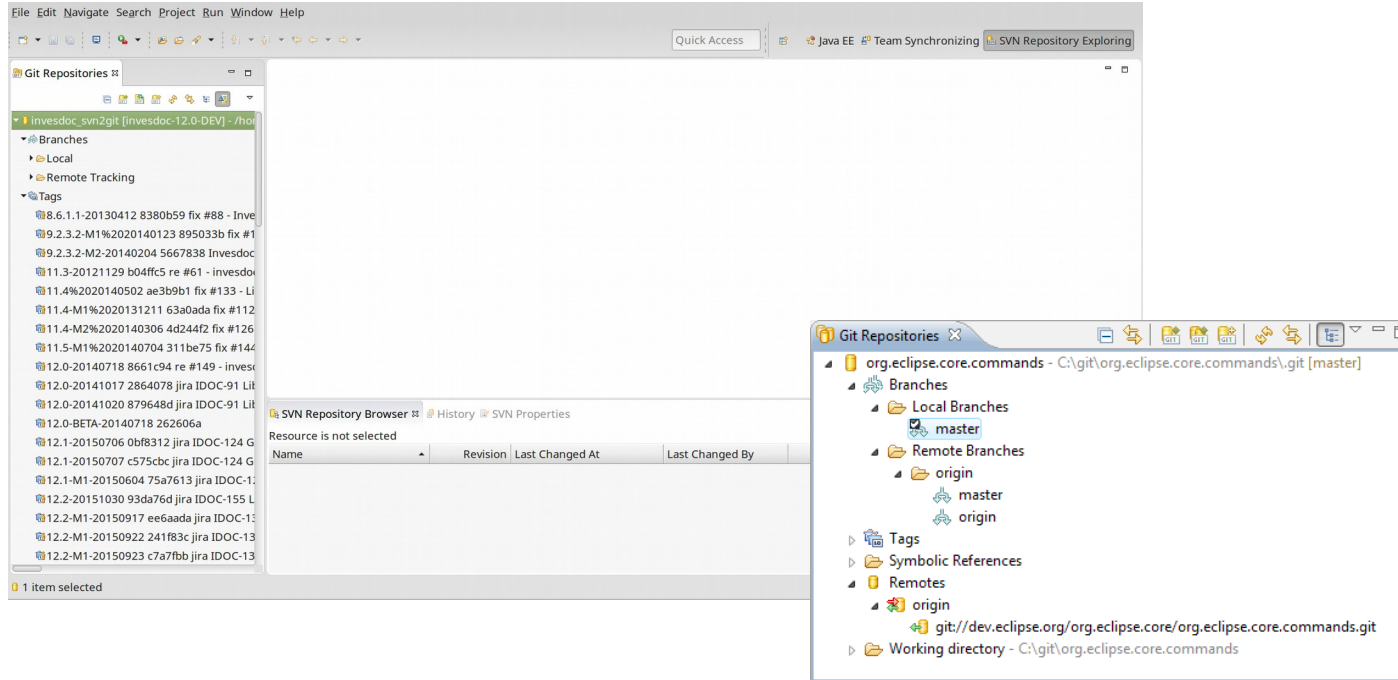
# Git : requisitos para poder empezar a trabajar

- Instalación de cliente Git:
  - Linux : Git
  - Windows : msysGit  
<http://msysgit.github.com>
- Integración con Eclipse : Egit
- Gogs





# Git : Integración con Eclipse



<http://www.eclipse.org/egit/documentation/>



INFORMÁTICA

El Corte Inglés