ES
Español

Arquitectura del Software

Seminario: GIT

2023

Ing. Liborio Castañeda Valencia

Sistemas de control de versiones

- Centralizados
  - Un repositorio centralizado de todo el código
  - Ejemplos: CVS, Subversión,...
- Distribuidos
  - Cada usuario tiene su propio repositorio
  - Ejemplos: mercurial, git, ...

Git

Diseñado por Linus Torvalds (Linux), 2005

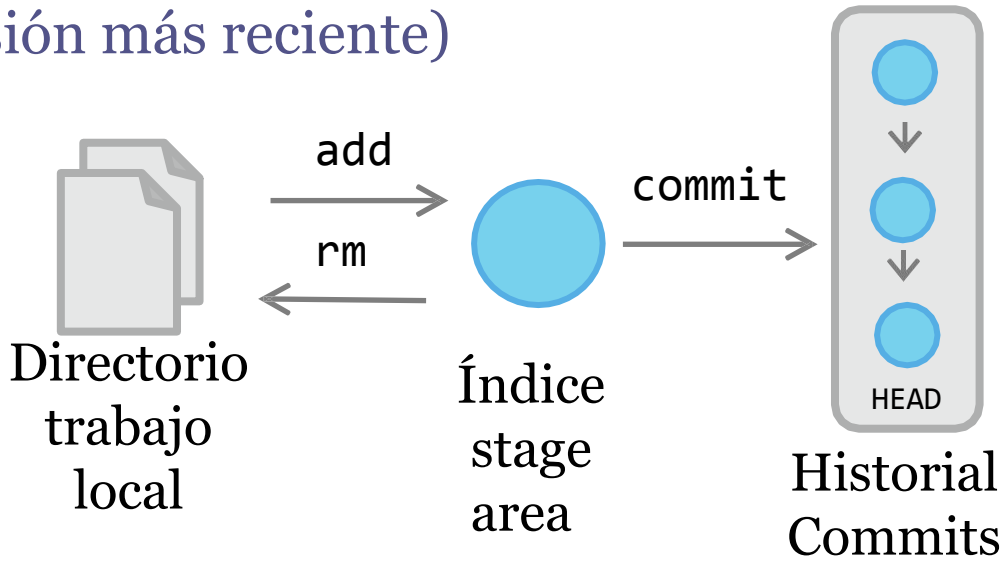
Objetivos:

- Aplicaciones con gran nº de archivos de código
  - Eficiencia
- Trabajo distribuido
  - Cada desarrollador tiene su propio repositorio
    - Copia local de todo el historial de cambios
    - Es posible realizar commit's incluso sin conexión
- Desarrollo no lineal (ramificaciones)

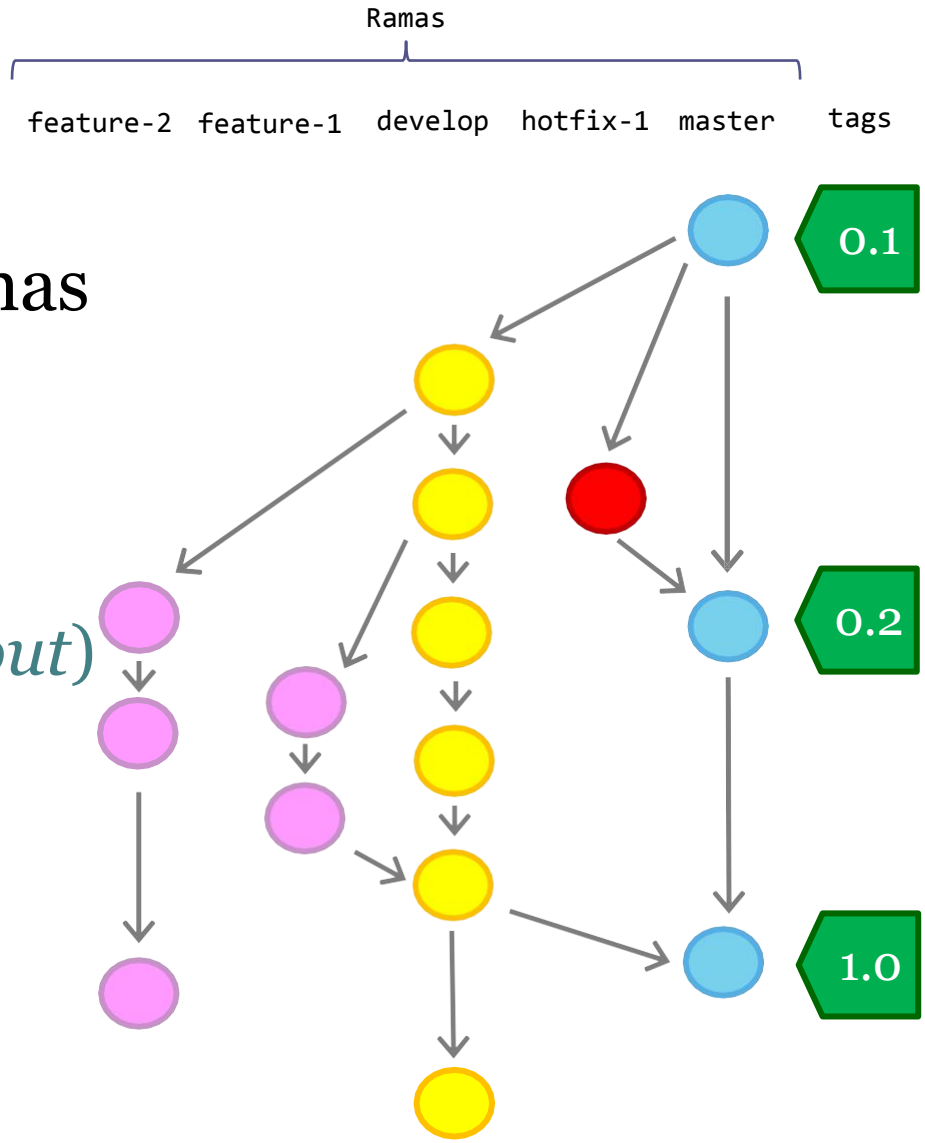
Componentes locales

3 componentes locales:

- Directorio de trabajo local
- Índice: área de ensayo (stage). A veces también caché.
- Historial: Almacena versiones ó commits
  - HEAD (versión más reciente)



Ramas



Git facilita gestión de ramas

- master = rama inicial

Operaciones:

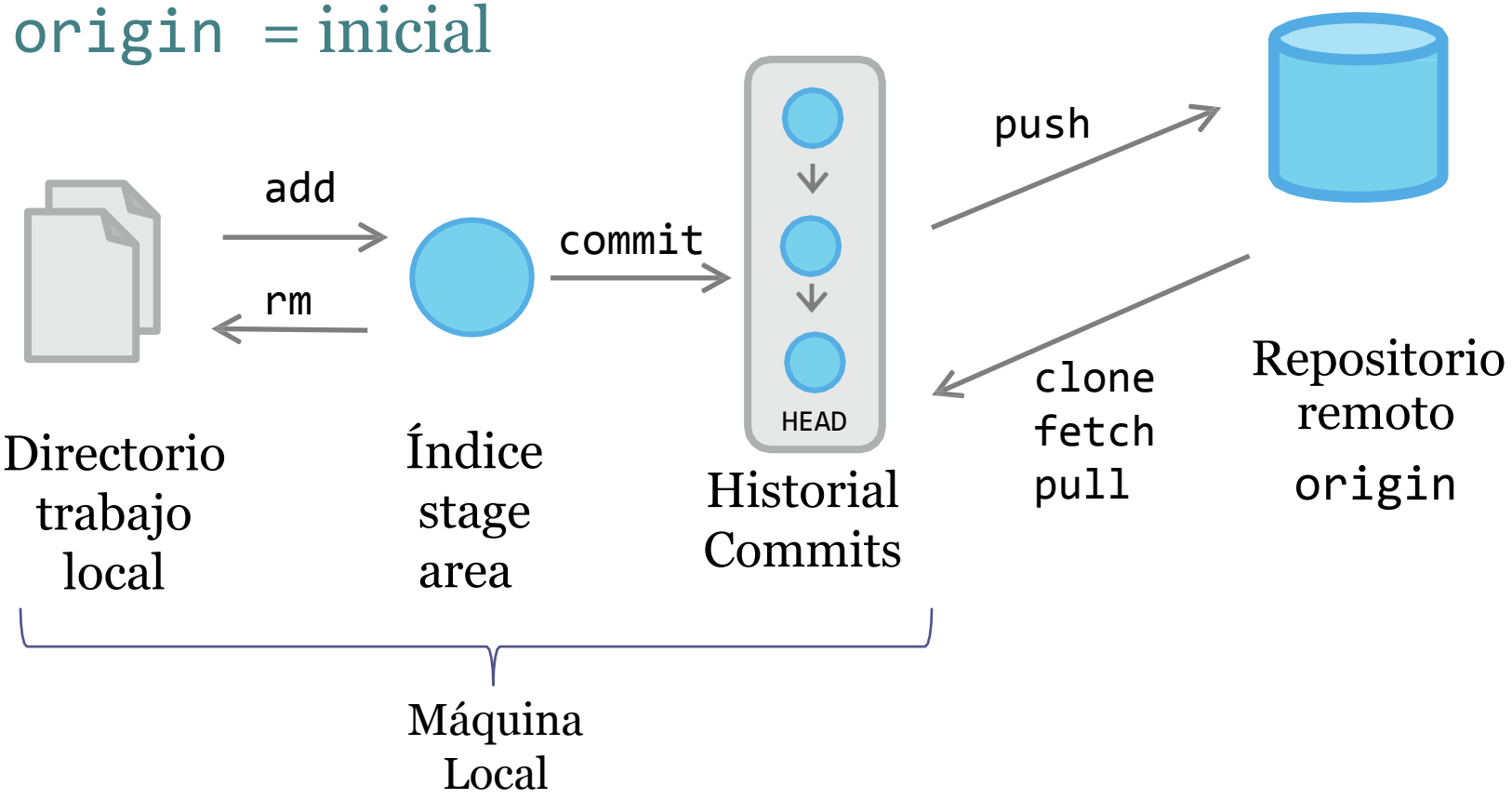
- Crear ramas (*branch*)
- Cambiar a ramas (*checkout*)
- Combinar (*merge*)
- Etiquetar (*tag*)

Múltiples estilos de ramificación

Repositorios remotos

Se pueden conectar con repositorios remotos

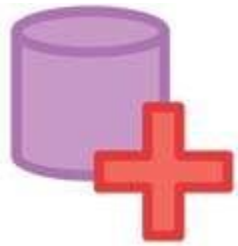
`origin` = inicial



Funcionamiento básico

```
init
clone
config
add
commit
status
log
diff
```

init - Crear repositorios



`git init`

Transforma el directorio actual en repositorio Git

Se crea directorio .git

Variantes:

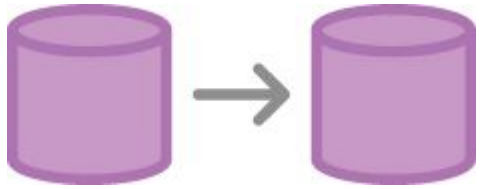
`git init <directorio>`

Crea un repositorio vacío en el directorio especificado

`git init --bare <directorio>`

Inicializa repositorio Git pero omite directorio de trabajo

NOTA: Normalmente, esta instrucción sólo se realiza una vez



clone - Clonar repositorios

```
git clone <repo>
```

Clonar el repositorio <repo> en la máquina local

<repo> puede estar en una máquina remota

Ejemplo:

```
git clone
https://github.com/Arquisoft/ObservaTerra0.git
```

NOTA: Al igual que init, esta instrucción sólo se realiza una vez

config - Configurar git

```
git config --global user.name <name>
```

Declara el nombre de usuario

Otras opciones de configuración:

user.email, merge.tool, core.editor, ...

Ficheros de configuración:

<repo>/.git/config -- Específicos de repositorio

~/.git/config -- Globales

add - Añadir al índice

```
git add <fichero>
git add <dir>
```

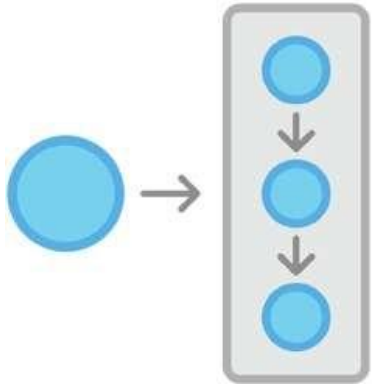
Añade fichero o directorio al índice

Variantes

`git add --all` = Añade/borra ficheros

El índice ó área de ensayo almacena copias de los ficheros antes de ser incluidos en el historial

commit - Añadir al historial



```
git commit
git commit -m "mensaje"
```

- Añade los ficheros del índice al historial
  - Crea una nueva instantánea "snapshot" del proyecto
  - Cada instantánea tiene un identificador SHA1
    - Puede recuperarse posteriormente
    - Pueden asignarse etiquetas para facilitar su gestión

NOTA: Conviene excluir de control de versiones algunos ficheros
Ejemplos: binarios (*.class), temporales, configuración (.settings), privados (claves de Bases de datos...), etc.
Se incluyen en fichero: `.gitignore`

status - Observar índice

```
git status
```

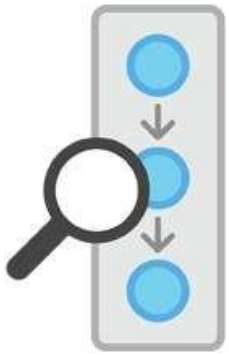
Muestra ficheros *staged*, *unstaged* y *untracked*

staged = en índice pero no en historial

unstaged = modificados pero no añadidos a índice

untracked = en directorio de trabajo

log - Observar historial



`git log`

Muestra historial de cambios

Variantes

`git log --oneline`	Resumen en 1 línea
`git log --stat`	Estadísticas
`git log -p`	Camino completo con diff
`git log --autor="expr"`	Commits de un autor
`git log --grep="expr"`	Busca commits

`git log --graph --decorate --online`

Muestra grafo de cambios

diff - Mostrar diferencias

`git diff`

Dir. trabajo vs índice

`git diff --cached`

Índice vs commit

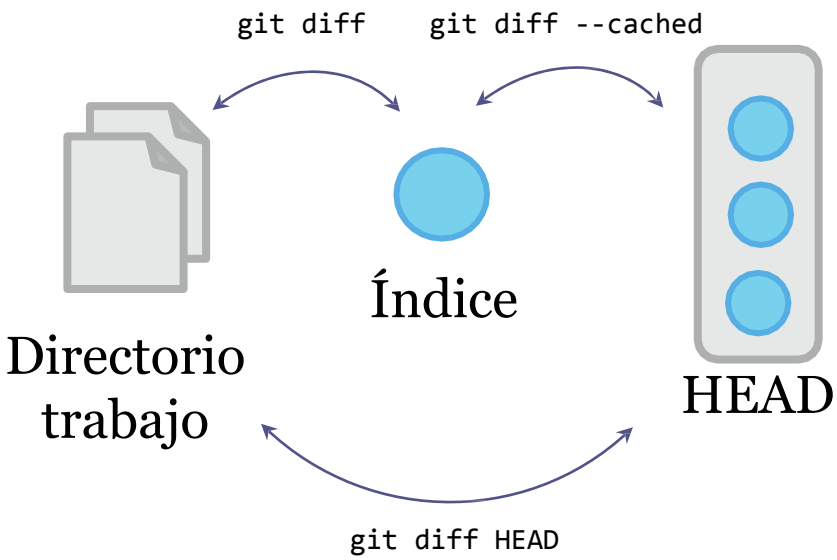
`git diff HEAD`

Dir. trabajo vs commit

Algunas pciones:

--color-words

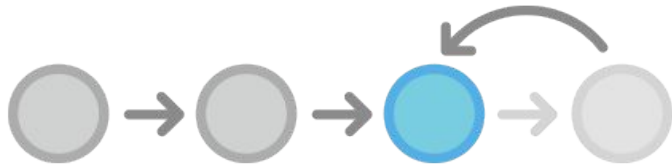
--stat



Deshaciendo cambios

Comandos para deshacer cambios

- checkout
- revert
- reset
- clean

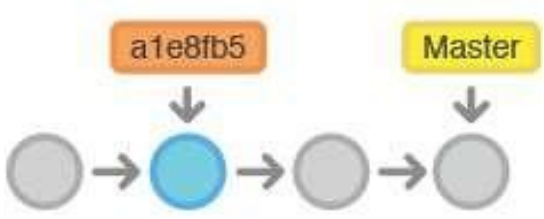


checkout - Cambiar

Cambia directorio de trabajo

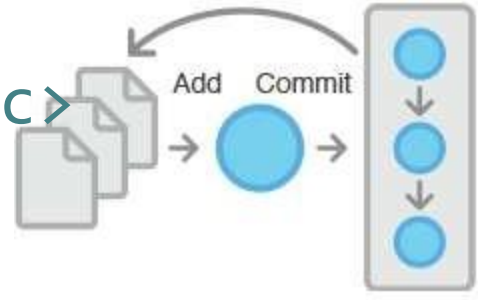
`git checkout <c>` Cambiar a commit `<c>`

Se pasa a estado "*detached HEAD*"



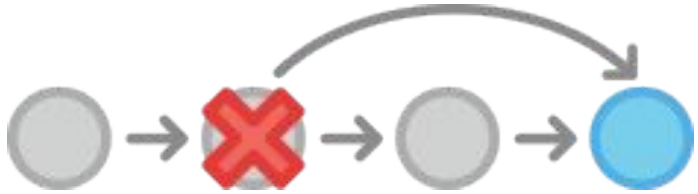
`git checkout <c> <f>`

Recupera fichero `<f>` de commit `<c>`



NOTA:
`checkout` también se utiliza para cambiar a diferentes ramas

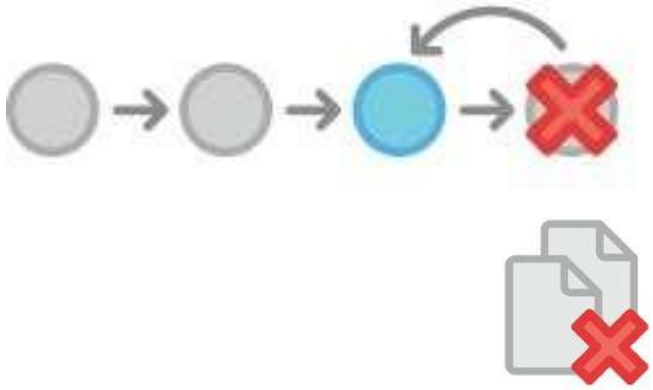
revert - Recuperar



`git revert <c>` Recupera commit `<c>`

- Añade la versión recuperada al historial
- Operación "segura"
  - permite rastrear cambios en historial

reset - Deshacer



Deshacer cambios

Operación no segura

`git reset`	Deshace cambios en índice
`git reset --hard`	Deshace cambios en índice y directorio trabajo
`git reset <c>`	Deshacer cambios y recuperar commit `<c>`

NOTA: Es peligroso hacer reset en repositorios ya publicados
Es mejor utilizar `revert`

clean - Limpiar

Borrar ficheros locales

`git clean -f` Borra ficheros *untracked*

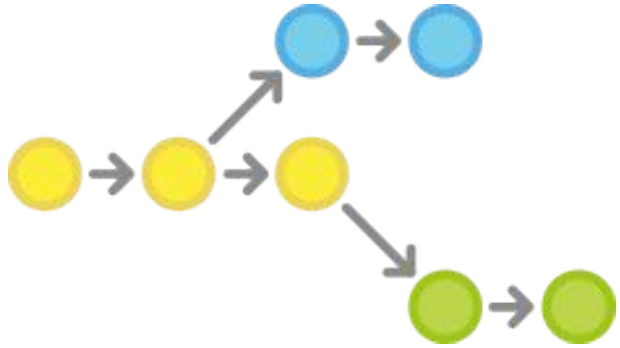
NOTA: Peligroso (se pueden perder cambios locales)

`git clean -n` Muestra qué ficheros se borrarían

Ramas

```
branch
checkout
merge
```

branch - Ramas



Gestión de ramas

`git branch` Muestra las ramas existentes

`git branch <r>` Crear la rama `<r>`

`git branch -d <r>` Borrar rama `<r>`

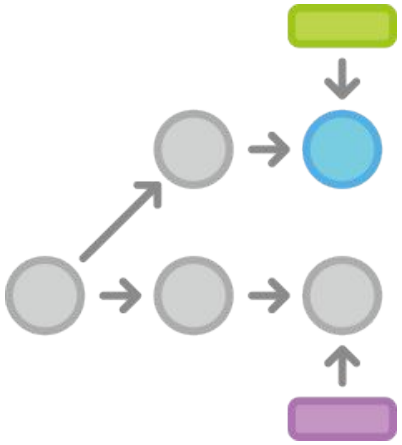
Segura (no borra si no hay mezclas pendientes)

`git branch -D <r>` Borrar rama `<r>`

Insegura (borra una rama y sus commits)

`git branch -m <r>` Renombrar rama actual a `<r>`

checkout - Cambiar



Cambiar a una rama

`git checkout <r>`

Cambia a la rama existente `<r>`

`git checkout master` Cambia a rama master

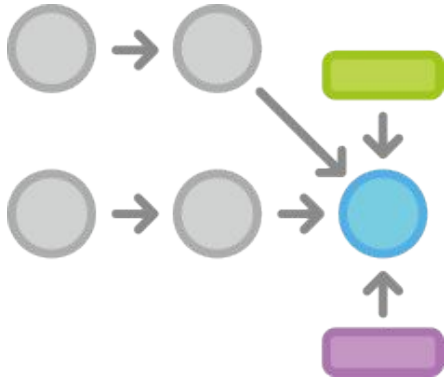
`git checkout -b <r>`

Crear rama `<r>` y cambia a ella

Equivalente a

```
git branch <r>
git checkout <r>
```

merge - Combinar



Combinar dos ramas

`git merge <r>`	Mezclar rama actual con `<r>`
`git merge --no-ff <r>`	Mezclar generando commit de mezcla (más seguro)

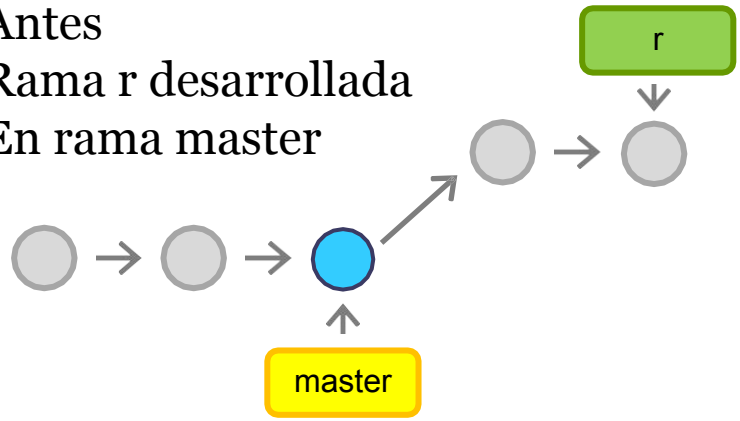
2 tipos de combinación

- Merge fast-forward
- 3-way merge

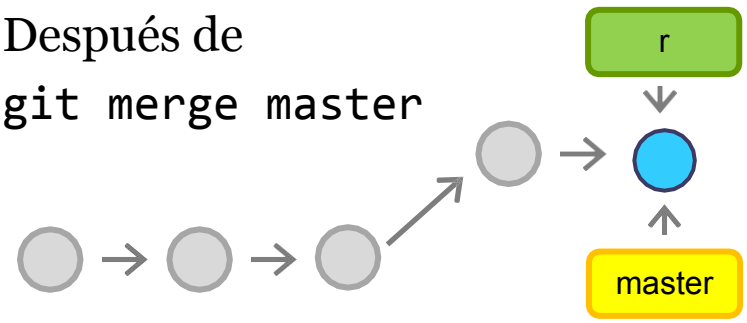
merge fast-forward

Cuando hay camino lineal entre rama actual y rama a mezclar

Antes
Rama r desarrollada
En rama master



Después de
`git merge master`



Modificar historial

```
commit --amend
rebase
reflog
```

reflog - movimientos en historial

`git reflog`

- Muestra movimientos del historial
  - git almacena información de todos los movimientos que se producen en las diferentes ramas
    - git reflog muestra los cambios aunque ya no estén en ninguna rama

Repositorios remotos

```
remote
fetch
pull
push
```

remote - Conectar repositorios

`git remote`

Ver repositorios externos

`git remote add <nombre> <uri>`

Crear conexión de nombre `<nombre>` a `<uri>`

`git remote rm <nombre>`

Borrar conexión `<nombre>`

`git rename <anterior> <nuevo>`

Renombrar conexión `<anterior>` a `<nuevo>`

NOTAS: `git clone` crea automáticamente una conexión llamada `origin`
Es posible tener conexiones a más de un repositorio externo

fetch - traer

Traer elementos de repositorio remoto

- Permite descargar ramas externas
- Operación segura: no mezcla con ficheros locales

`git fetch <remote>`

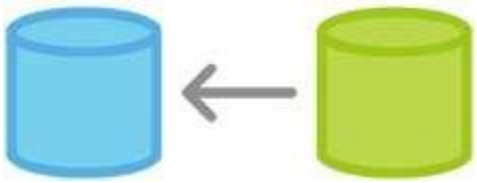
- Descargar todas las ramas del repositorio <remote>

`git fetch <remote> <rama>`

- Descargar la rama `<rama>` de repositorio `<remote>`

NOTA: Asigna FETCH_HEAD a cabeza de rama traída
Convenio para nombrar ramas: `<remoto>/<rama>`
Ejemplo: `origin/master`

pull - traer y mezclar



```
git pull <remoto>
```

Trae un repositorio remoto y lo mezcla

Equivale a:

```
git fetch
git merge FETCH_HEAD
```

push - enviar

```
git push <remoto> <rama>
```

Enviar commits de repositorio local a remoto

Variantes

```
git push <remoto> --all
```

Enviar todas las ramas

Si hay cambios en repositorio remoto, muestra un error (non-fast-forward).
Solución:

1. Traer (pull) cambios y mezclar con repositorio local
2. Volver a enviar (push)

NOTA: También puede usarse opción: `--force` (no recomendado)

Github

- Herramienta de codificación social
  - Compañía Github Inc. creada en 2008
    - 2013: >3 millones de usuarios, >5millones proyectos
- Gestión gratuita de proyectos de código abierto
- Proyectos públicos por defecto y gratis
  - Es posible tener proyectos privados pagando

Github

- Facilidades como repositorio de proyectos
  - Gestión de issues/milestones
  - Wiki
  - Seguimiento de repositorios, usuarios, etc.
- Pull Requests
  - Solicitar combinaciones y mezclas
  - Revisiones de código
    - Permite incluir comentarios, ver diferencias, etc.
  - Gestión de listas de *pull requests*

Referencias

Tutoriales

- http://rogerdudler.github.com/git-guide/
- https://www.atlassian.com/git
- http://training.github.com/materials/slides/
- http://nvie.com/posts/a-successful-git-branching-model/

Vídeos

- http://vimeo.com/49444883