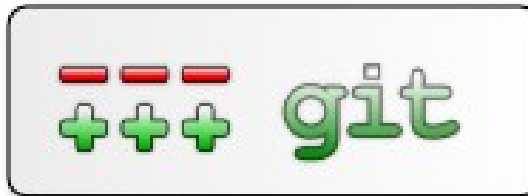


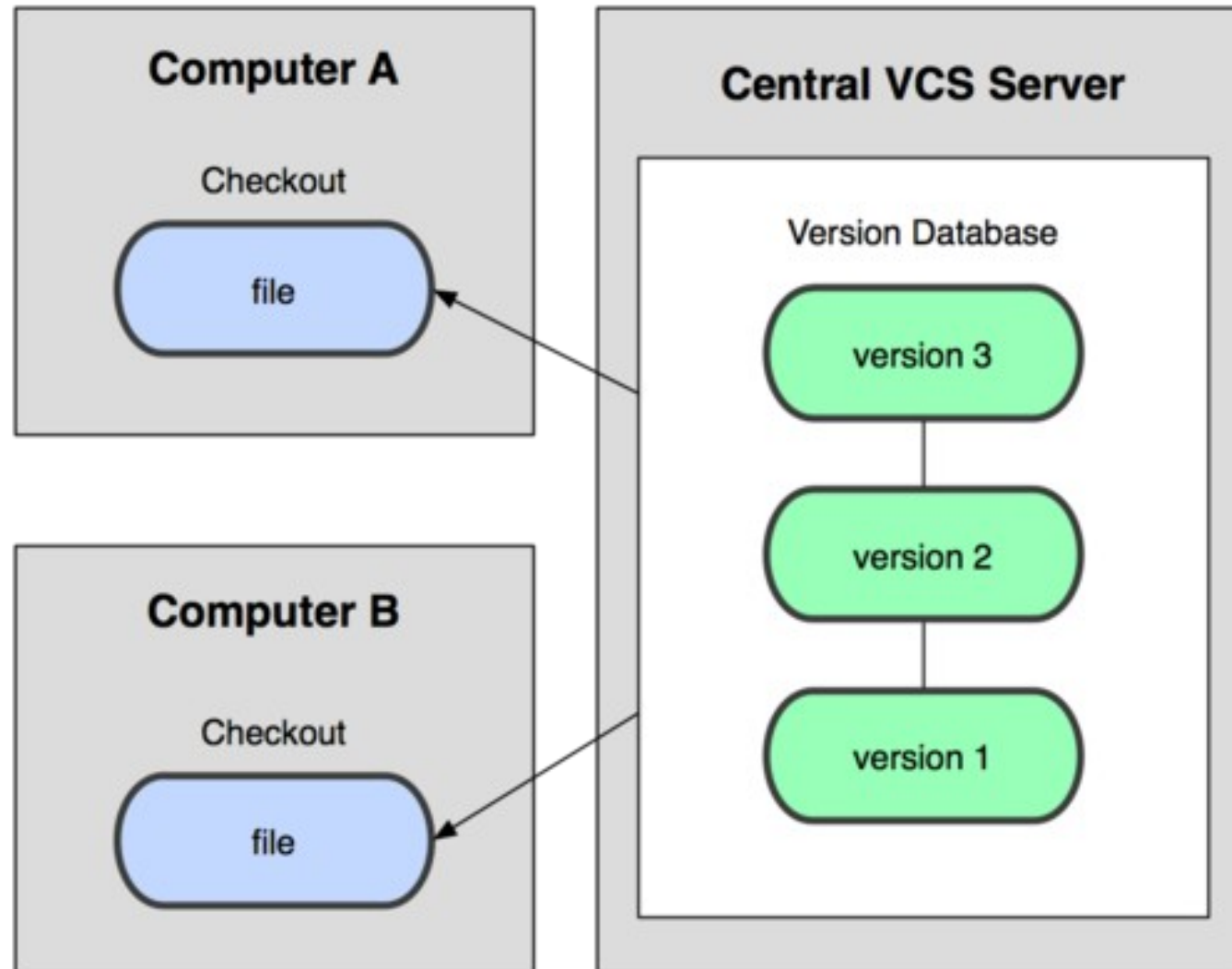
Introducción al uso de



Iñaki Arenaza
iarenaza@mondragon.edu
@iarenaza

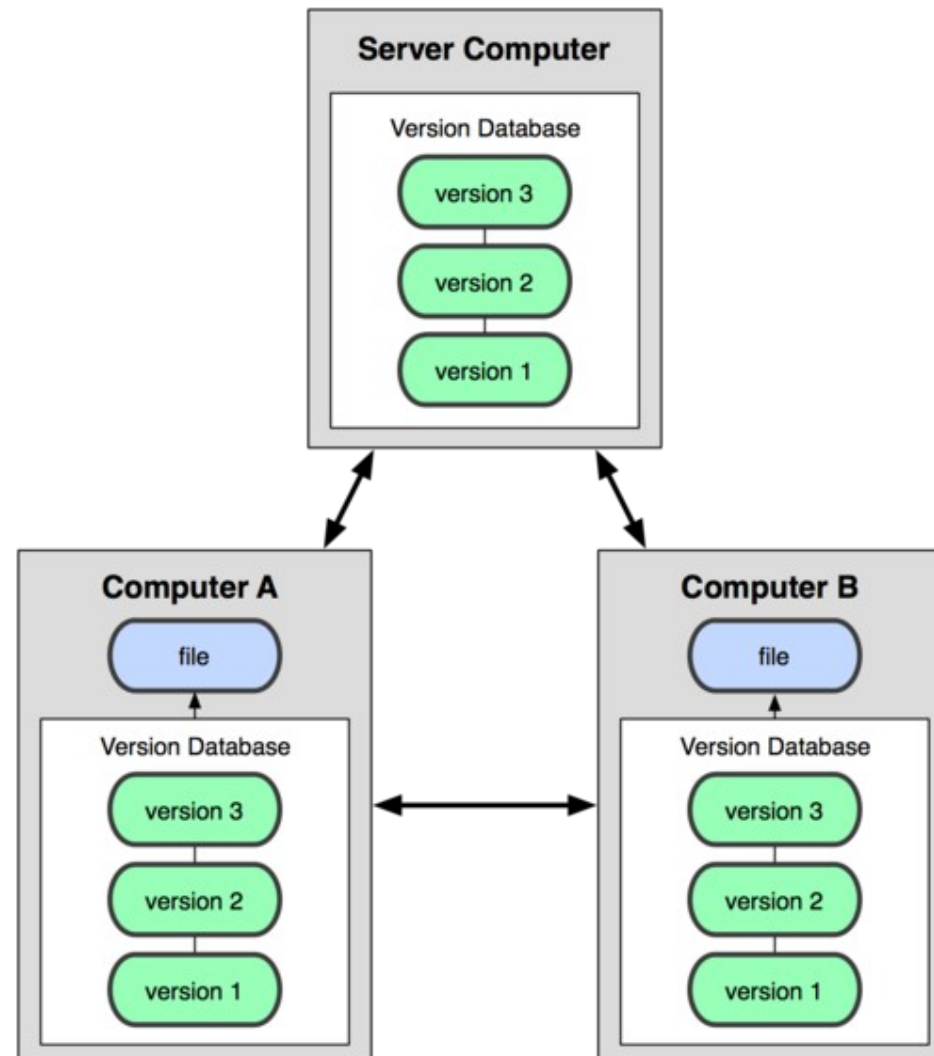
Sistemas de Control de Versiones Centralizados (CVCS)

Ejemplos: CVS, Subversion, Perforce, SourceSafe, ...

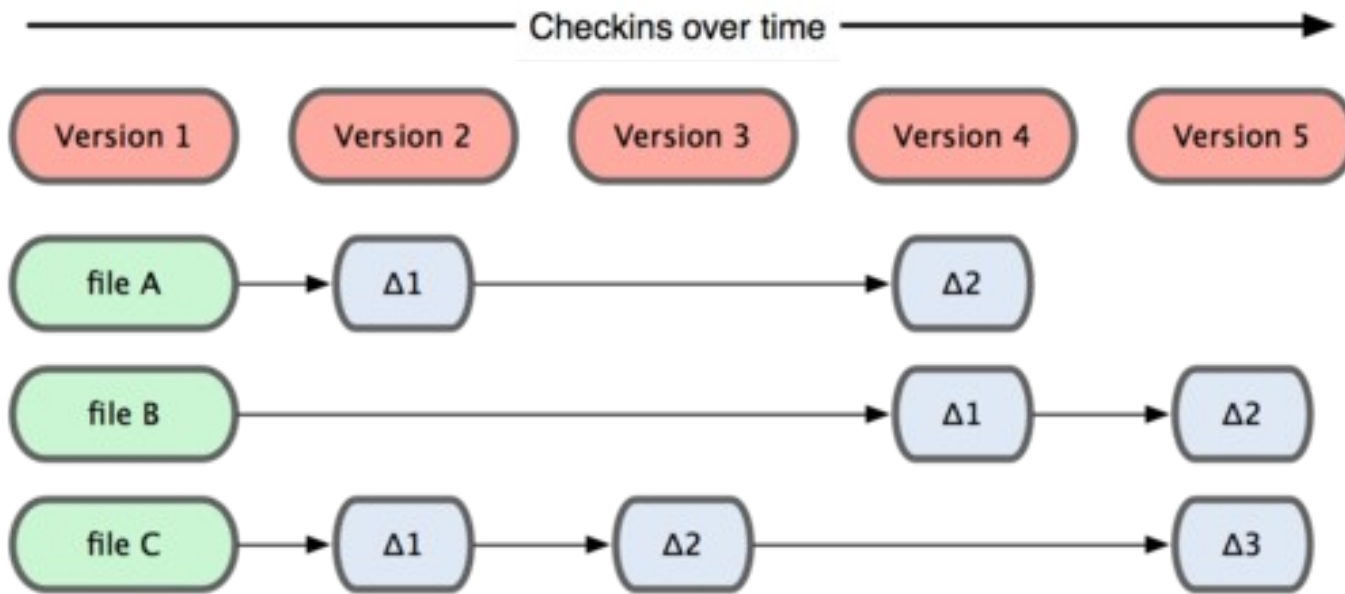


Sistemas de Control de Versiones Distribuidos (DVCS)

Ejemplos: git, Mercurial, Bazaar, BitKeeper,...

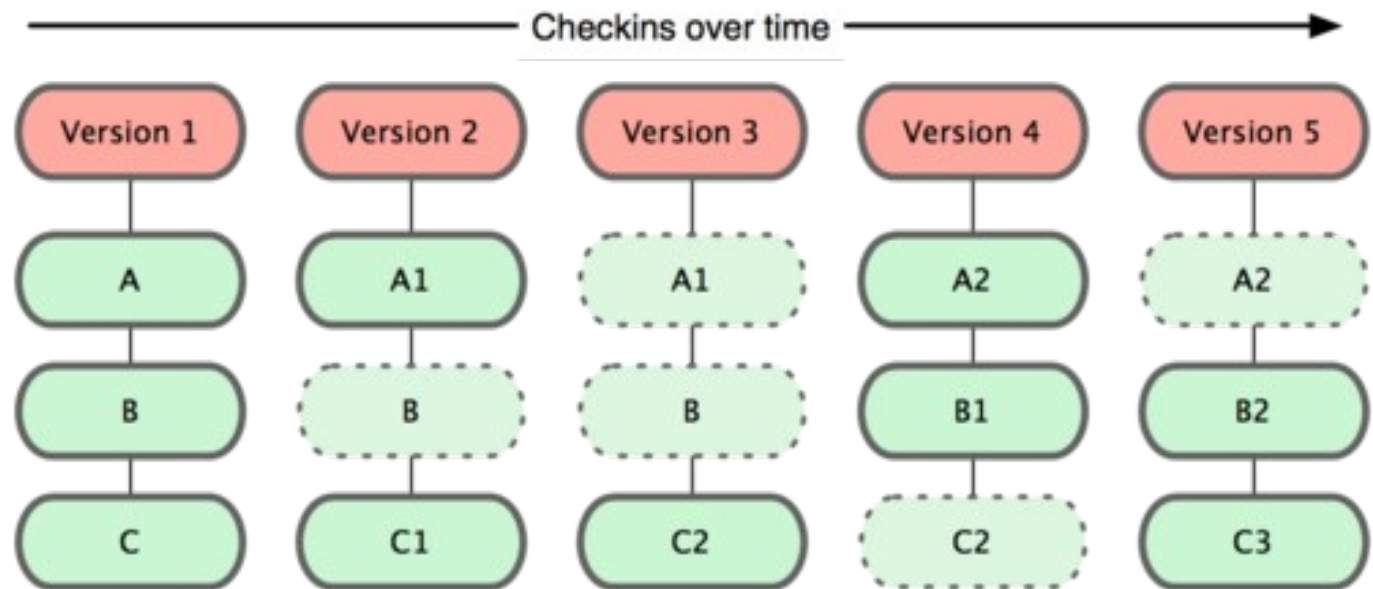


Diferencias versus instantáneas



Bazaar,
Mercurial*

Fuente: <http://progit.org/book/ch1-3.html> (CC-BY-NC-SA 3.0)



git,
Mercurial*

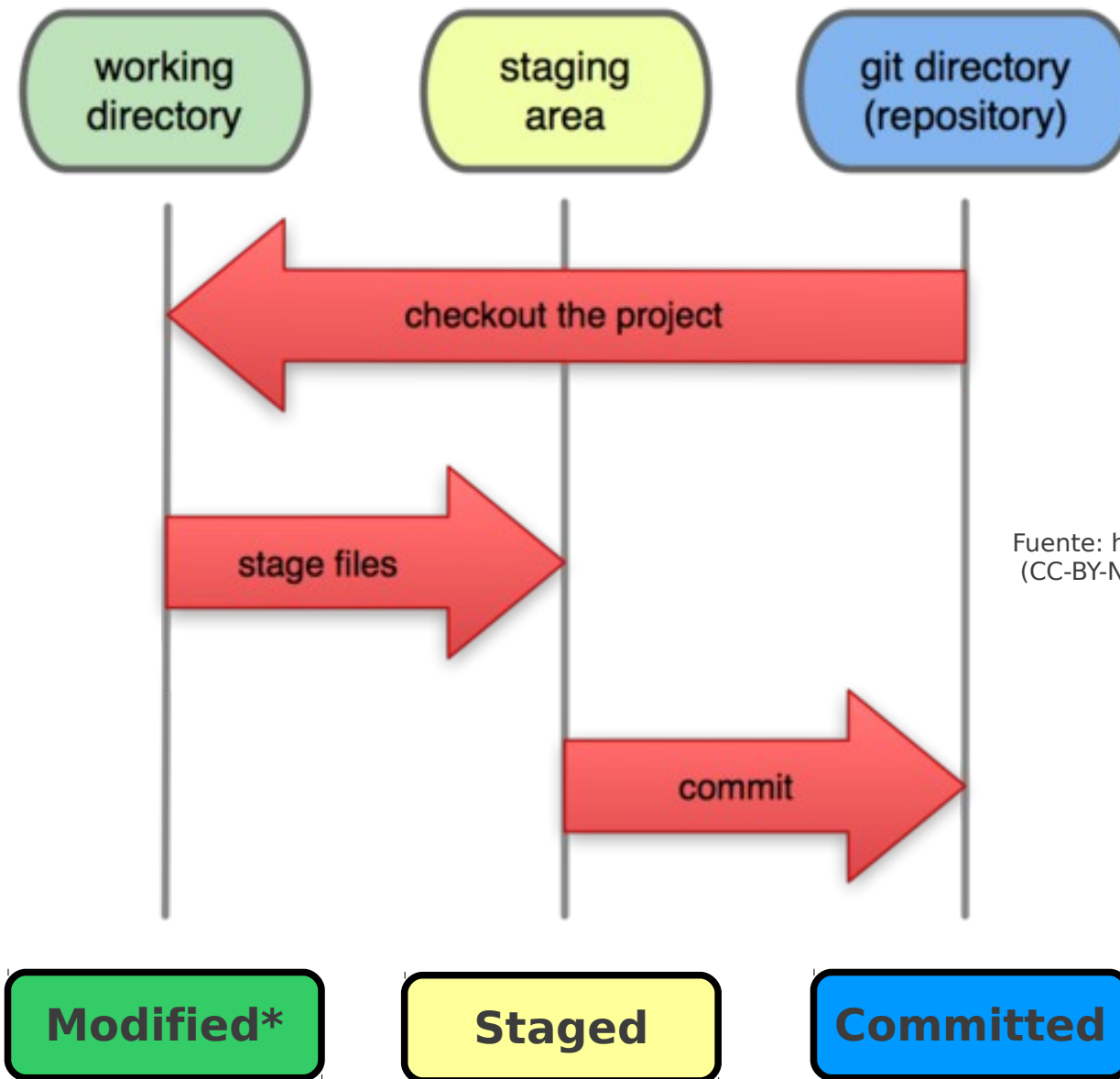
Fuente: <http://progit.org/book/ch1-3.html> (CC-BY-NC-SA 3.0)

(Algunas) características de git

- (Casi) todas las operaciones son locales
- Git tiene integridad fuerte (sha1)
- Git (generalmente) sólo añade datos

Los tres espacios y tres estados

Local Operations



Fuente: <http://progit.org/book/ch1-3.html>
(CC-BY-NC-SA 3.0)

Los tres espacios

- El directorio (repositorio) es donde git almacena los metadatos y la base de datos de objetos para tu proyecto.
- El directorio de trabajo es una copia de trabajo de una versión del proyecto.
- El área de preparación (staging area) es un archivo que almacena información sobre lo que irá en el próximo commit. Antes se le llamaba “el índice”.

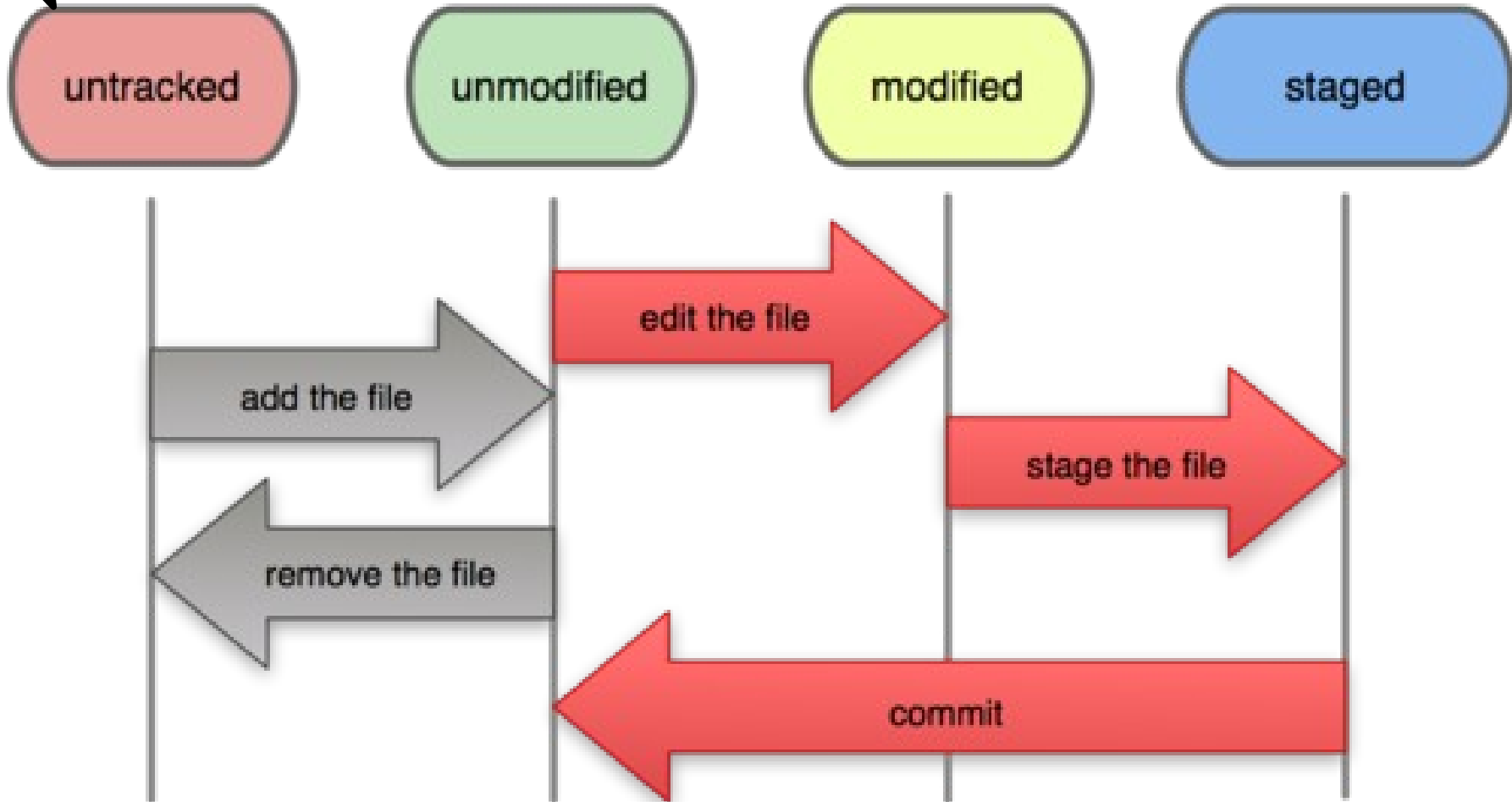
Los tres estados

- Confirmado/no modificado (committed): los datos están almacenados de manera segura en el directorio.
- Modificado (modified): se ha modificado el archivo pero todavía no se ha confirmado.
- Preparado (staged): se ha marcado para confirmación un archivo modificado en su versión actual.

Los “tres + 1” estados

El estado
3+1

File Status Lifecycle



Terminología de git

- En los ejemplos siguientes `$GIT_DIR` contiene la ruta de un repositorio git dado
- objeto: unidad de almacenamiento en git.
 - Se identifica de forma unívoca por el SHA1 de su contenido.
 - Por tanto, un objeto es inmutable.

Tipos de objetos de git

- blob: objeto sin tipo, para guardar el contenido de un fichero. “Un fichero”.
- tree: lista de nombres y permisos, junto con las referencias de objetos blob o tree asociados. “Un directorio”.

Tipos de objetos de git

- commit: información de una revisión dada. Incluye:
 - los padres del objeto,
 - la persona que ha realizado el commit de la revisión,
 - el autor de la revisión,
 - la fecha de la misma,
 - un mensaje asociado,
 - el objeto tree que corresponde al directorio raíz de la revisión.

Tipos de objetos de git

- tag: identifica de forma simbólica a otros objetos y puede ser usado para firmar éstos. Contiene:
 - el nombre y tipo de otro objeto,
 - un nombre simbólico (el de la propia tag)
 - puede contener un mensaje asociado.
 - opcionalmente puede incluir una firma (PGP). En este último caso se denomina un "objeto de etiqueta firmada".

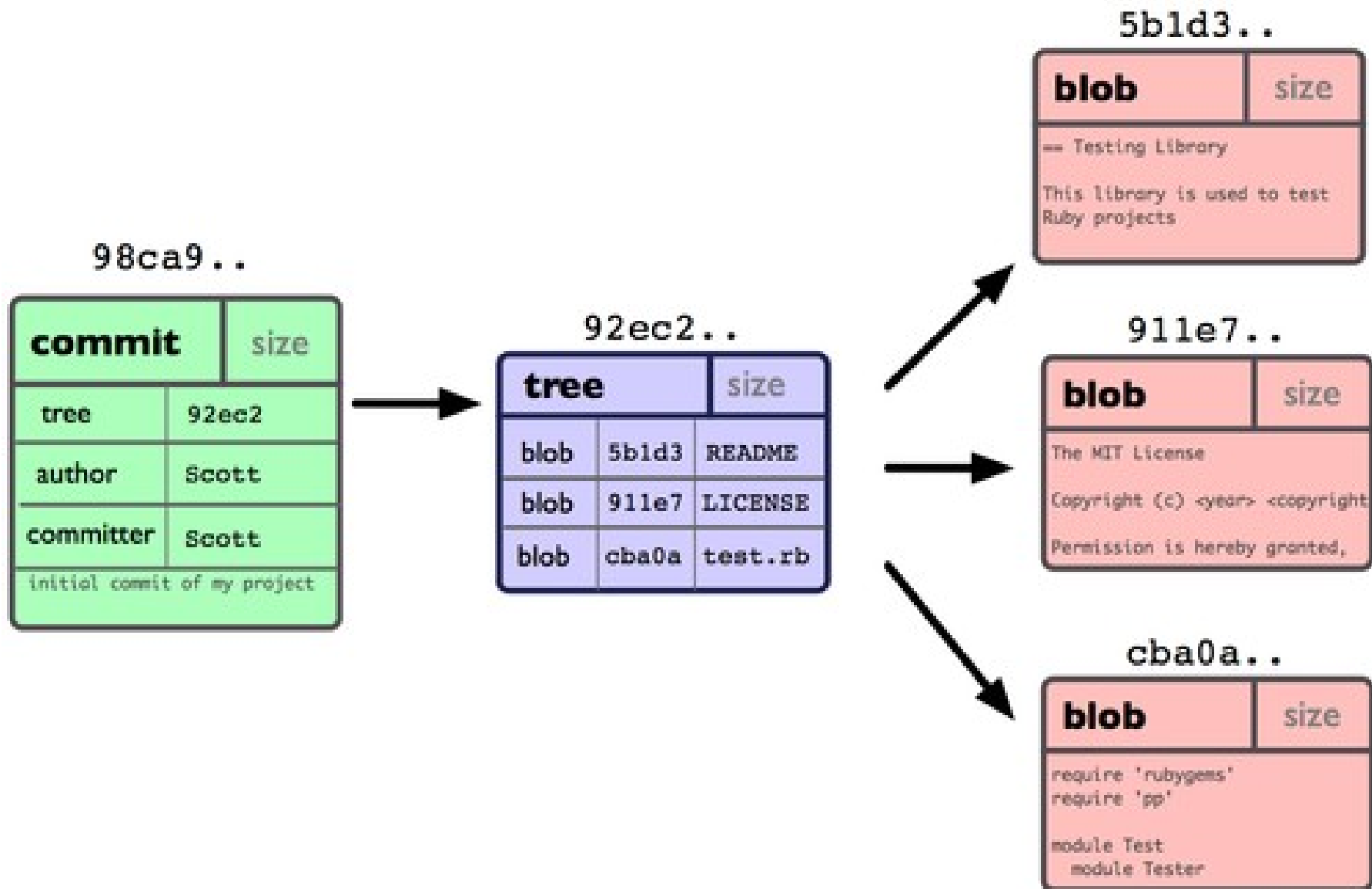
Terminología de git (cont.)

- nombre de objeto o identificador de objeto: identificador único del objeto (de 40 bytes con la representación hexadecimal del SHA1 de su contenido)
- base de datos de objetos: almacena un conjunto de objetos (habitualmente en `$GIT_DIR/objects/`).
- ref o referencia: cadena de 40 bytes con la representación hexadecimal de un SHA1, o un nombre simbólico (que se almacena en `$GIT_DIR/refs/`) que denota un objeto particular.

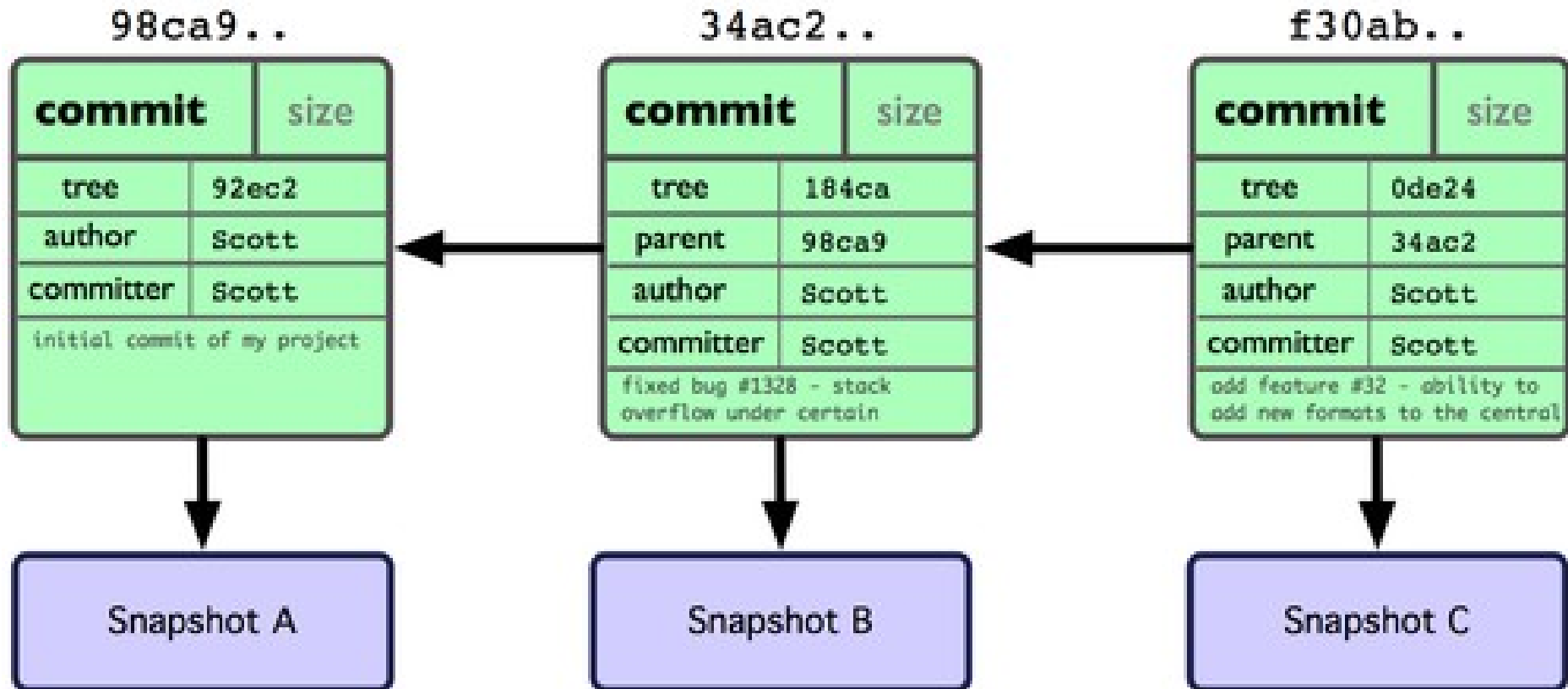
Terminología de git (cont.)

- **revisión:** estado concreto de una serie de ficheros y directorios que ha sido almacenado en la base de datos de objetos. Se hace referencia a él por medio de un objeto commit.
- **padre:** un objeto commit contiene una lista (potencialmente vacía) de objetos commit que representan a sus predecesores lógicos en la línea de desarrollo, esto es, sus ancestros.

Tipos de objetos de git



Tipos de objetos de git



Terminología de git (cont.)

- repositorio: colección de referencias junto con una base de datos de objetos tiene todos los objetos que son alcanzables desde dichas referencias.:
 - Puede contener además algunos meta datos adicionales usados por determinadas órdenes de git.
 - Puede contener una copia de trabajo de una revisión.
- repositorio desnudo (bare): repositorio que no tiene una copia de trabajo.
 - Los de control de git que normalmente estarían presentes en el subdirectorio oculto `.git` están presentes en el propio directorio del repositorio.

Terminología de git (cont.)

- árbol de trabajo o copia de trabajo: Una revisión extraída del repositorio, para poder trabajar con ella.
- índice: una colección de ficheros con información de `stat(2)`, cuyos contenidos están almacenados como objetos.
 - El índice es una versión almacenada del árbol de trabajo.

Terminología de git (cont.)

- rama: línea activa de desarrollo.
 - El commit más reciente de una rama se denomina la punta de dicha rama. La punta de la rama se referencia por medio de una cabeza.
 - La copia de trabajo está siempre asociada a una rama (la rama "actual" o "checked out") y la cabeza especial "HEAD" apunta a esa rama.
- cabeza: una referencia con nombre, que apunta al objeto commit de la punta de una rama.
 - Las cabezas se almacenan en `$GIT_DIR/refs/heads/`, (salvo que se usen referencias empaquetadas).

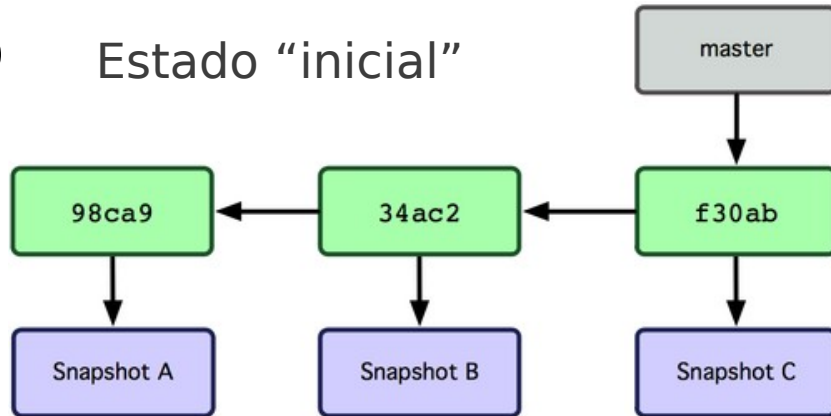
Terminología de git (cont.)

- checkout: acción de actualizar parte o todo el árbol de trabajo con un objeto árbol o blob desde la base de datos de objeto
 - Además actualiza el índice y la referencia HEAD si se ha cambiado de rama.

Terminología de git (cont.)

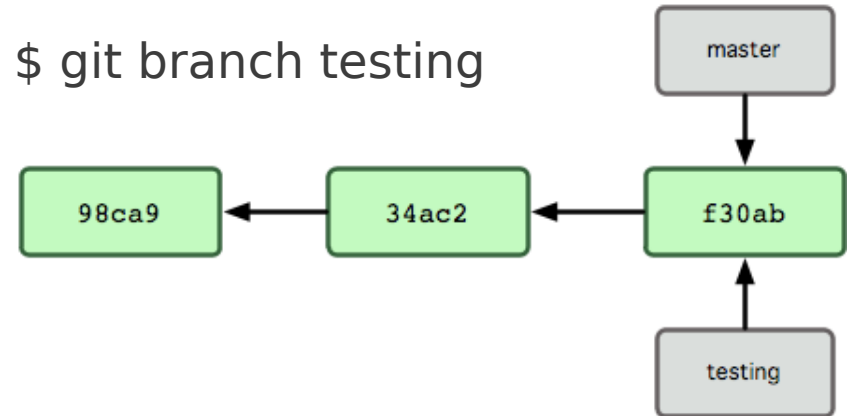
1

Estado "inicial"



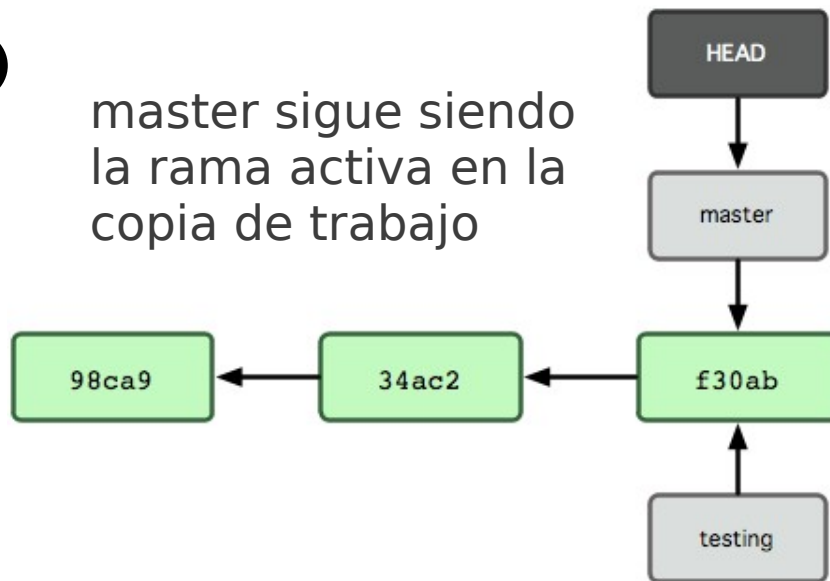
2

\$ git branch testing



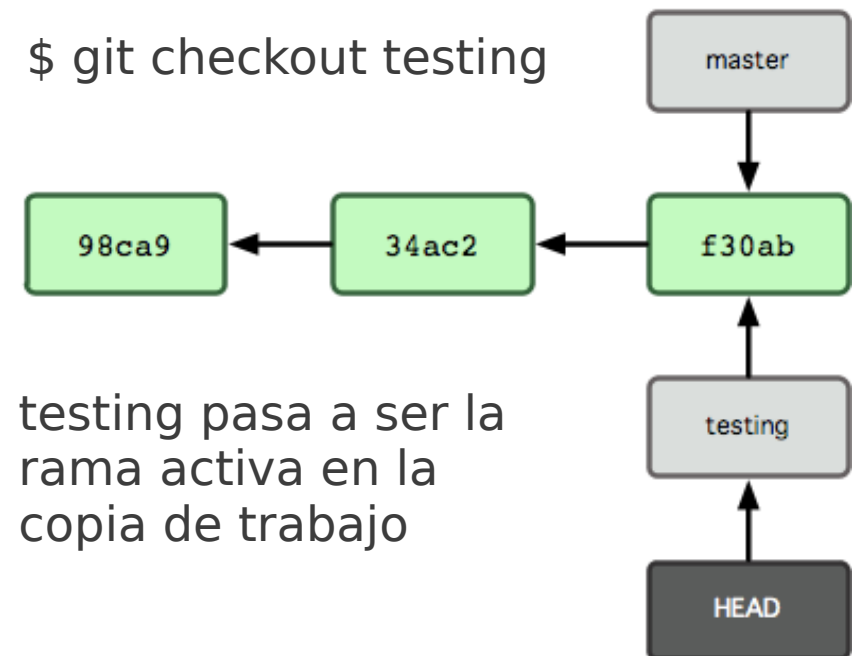
3

master sigue siendo la rama activa en la copia de trabajo



4

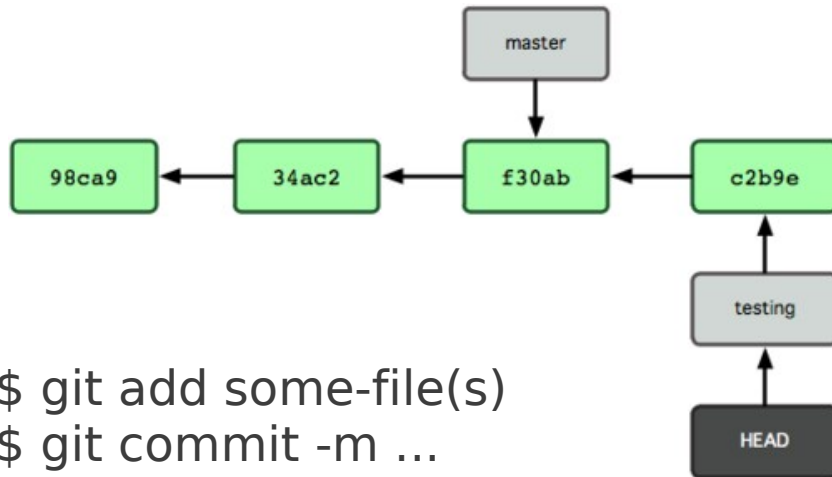
\$ git checkout testing



testing pasa a ser la rama activa en la copia de trabajo

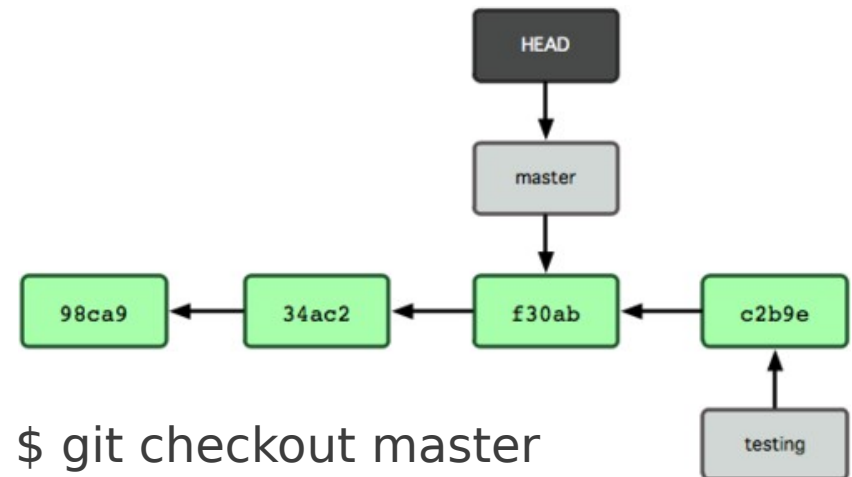
Terminología de git (cont.)

1



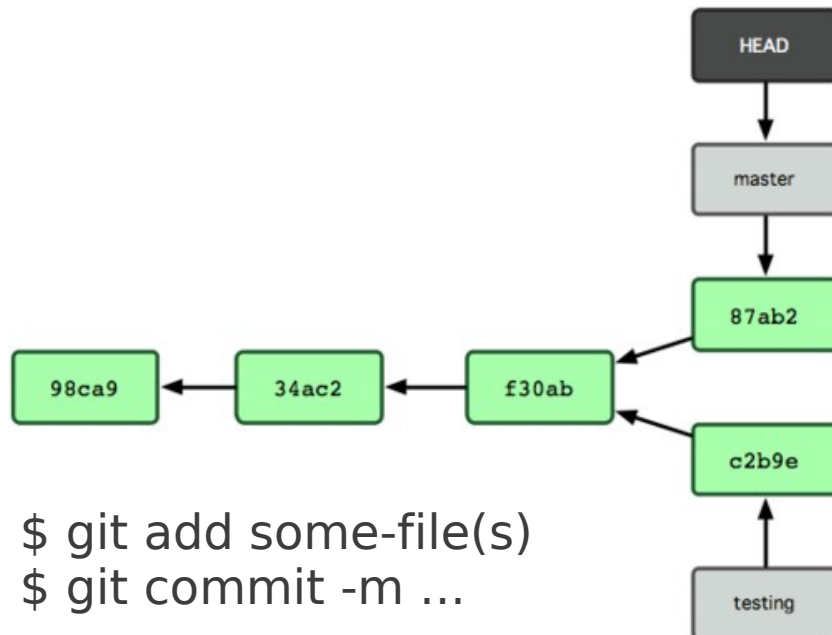
\$ git add some-file(s)
\$ git commit -m ...

2



\$ git checkout master

3



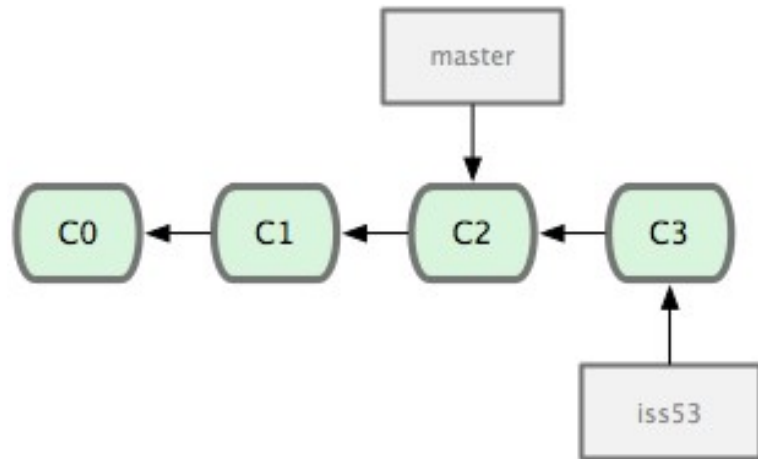
\$ git add some-file(s)
\$ git commit -m ...

Terminología de git (cont.)

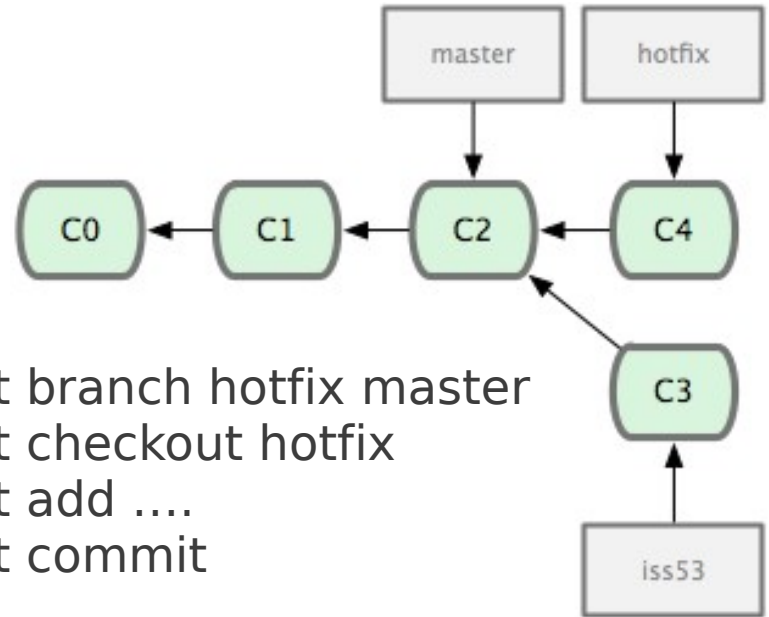
- merge: fusionar los contenidos de otra rama (potencialmente desde un repositorio externo) en la rama actual.
 - Si la rama es de otro repositorio, primero se hace un fetch* de la rama y después se fusiona en la rama actual.
 - La fusión puede crear un nuevo objeto commit si una de las ramas no es un ancestro de la otra.
 - Si una es ancestro de la otra, simplemente se mueve la referencia de la cabeza de la rama fusionada (*fast-forward merge*).

merge: escenario 1

1

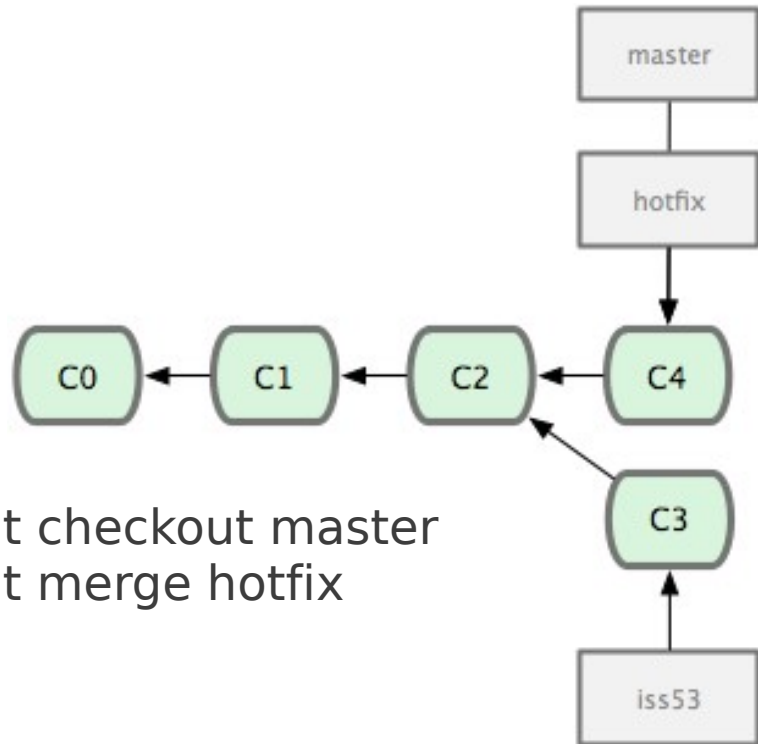


2



```
$ git branch hotfix master  
$ git checkout hotfix  
$ git add ....  
$ git commit
```

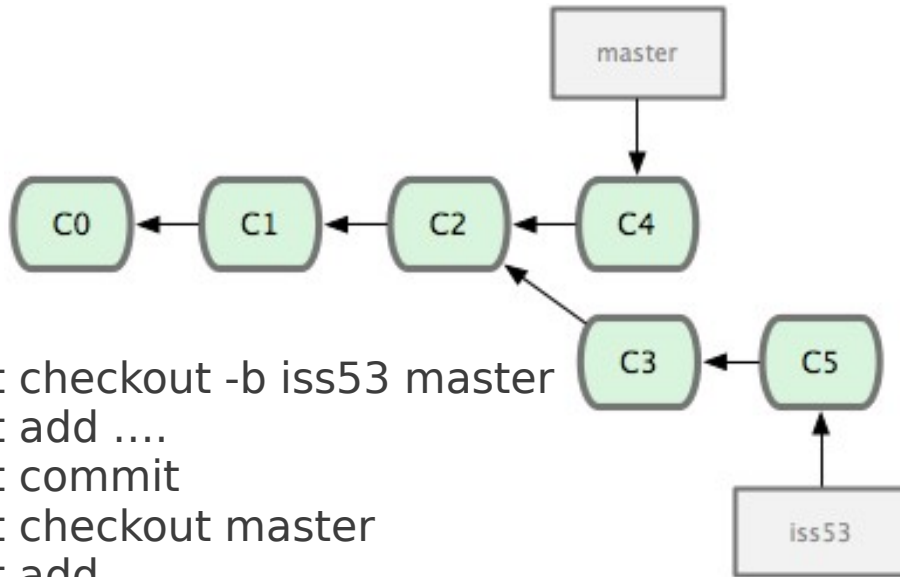
3



```
$ git checkout master  
$ git merge hotfix
```

merge: escenario 2

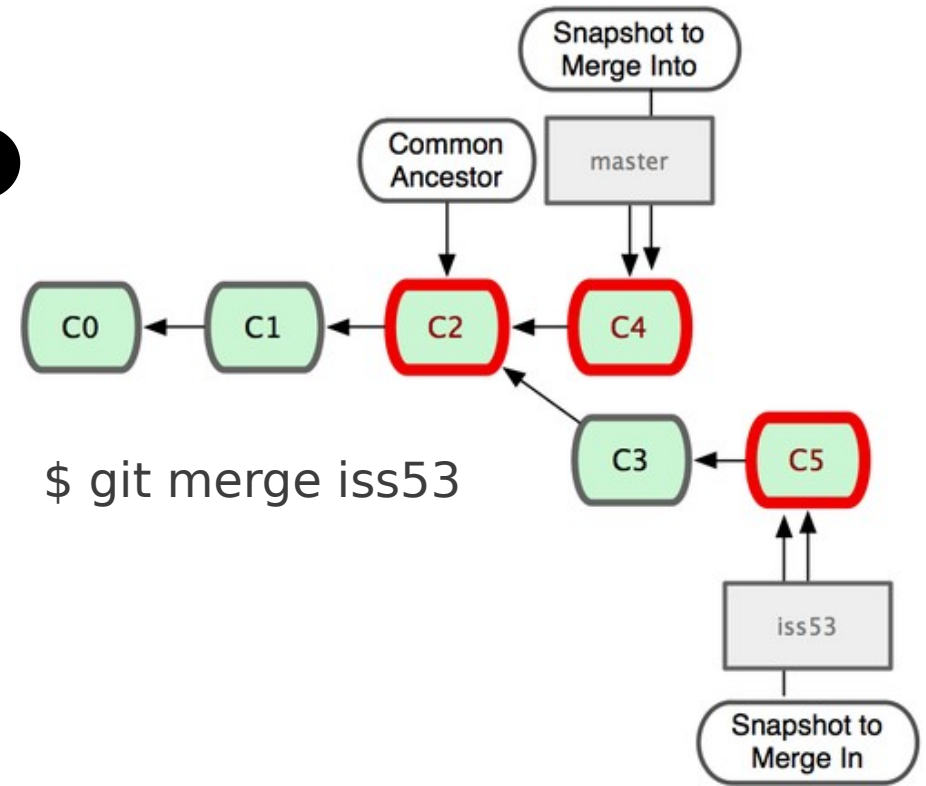
1



```

$ git checkout -b iss53 master
$ git add ....
$ git commit
$ git checkout master
$ git add ....
$ git commit
  
```

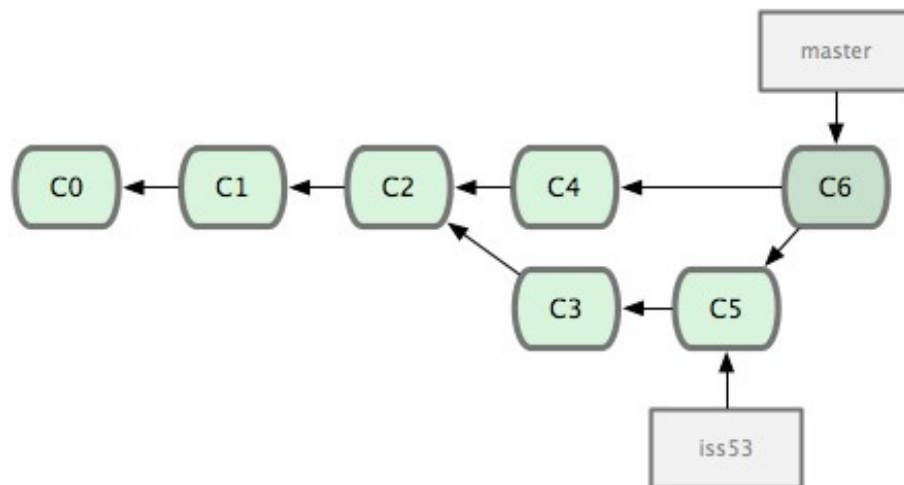
2



```

$ git merge iss53
  
```

3



Operaciones en el área de trabajo

- Crear nuevas ramas locales (ultra rápido y baratísimo en disco):

```
$ git branch mdl21-nested-groups mdl21-ldap-refactor
```

- Extraer una rama al área de trabajo:

```
$ git checkout mdl21-nested-groups
```

- Consultar la rama activa:

```
$ git branch
```

- Mostrar el estado del área de trabajo:

```
$ git status
```

- Marcar cambios para commit:

```
$ git add fichero1 fichero2 ...
```

```
$ git rm fichero3 fichero4 ...
```

Operaciones en el área de trabajo

- Mostrar diferencias con el índice o con HEAD:

```
$ git diff  
$ git diff HEAD
```

- Mostrar diferencias con otras ramas:

```
$ git diff MOODLE_21_STABLE  
$ git diff MOODLE_21_STABLE..mdl21-ldap-refactor
```

- Hacer commit de los cambios (marcados):

```
$ git commit
```

- Usar gitk para visualizar el historial de una rama:

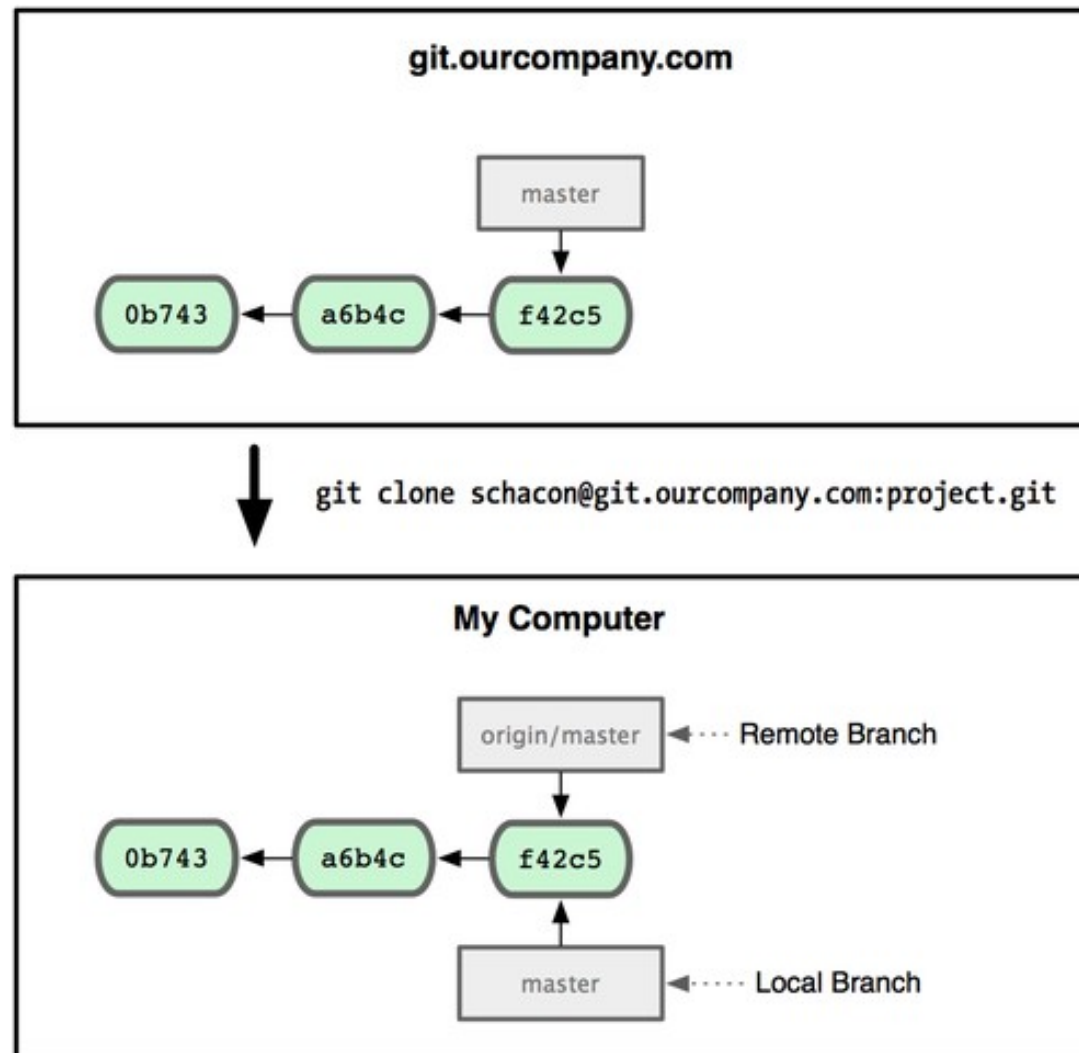
```
$ gitk mdl21-nested-group
```

- Usar gitk para visualizar el historial de todas las ramas:

```
$ gitk --all
```

Terminología de git (cont.)

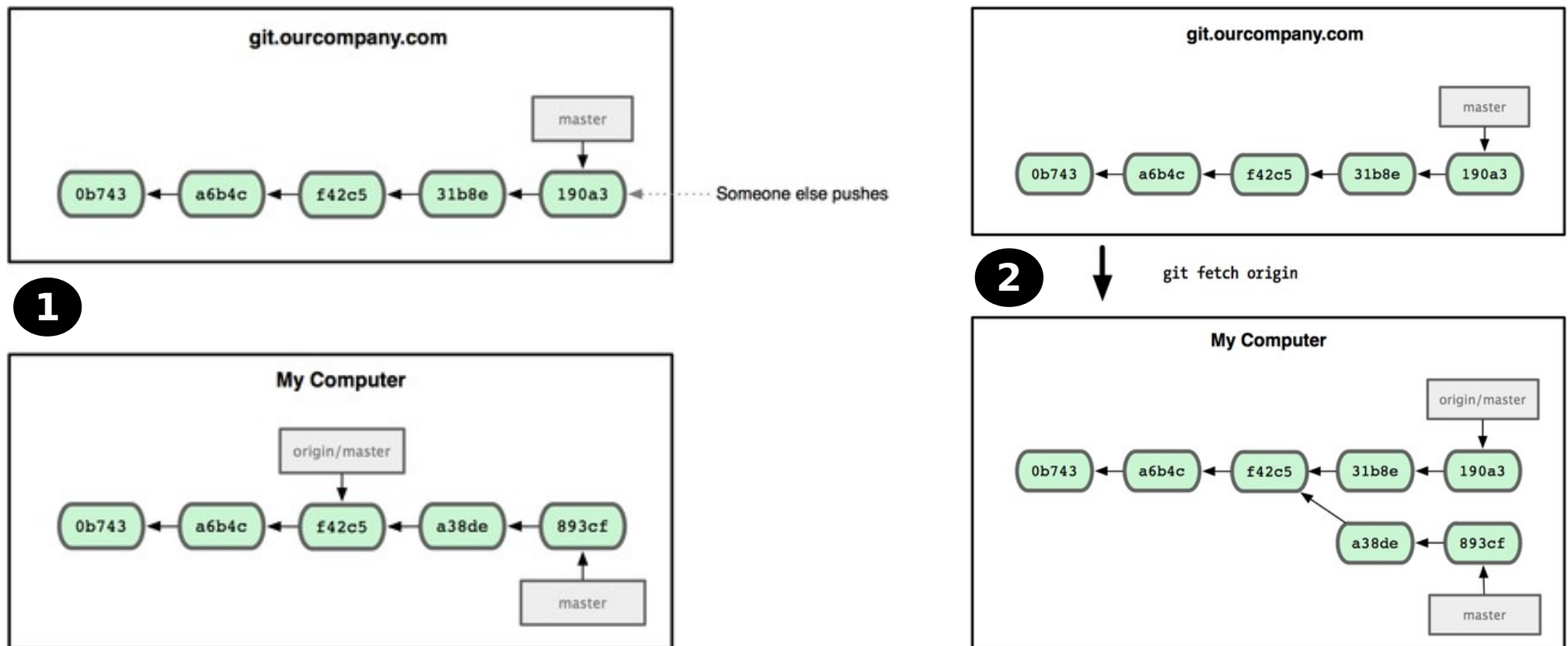
- clone: obtener una copia local completa* de un repositorio git remoto.



* las cabezas de las ramas remotas son inamovibles**

Terminología de git (cont.)

- fetch: obtener la cabeza de una rama (o varias) desde un repositorio remoto, copiando los objetos faltan y moviendo la(s) cabeza(s) remota(s).



Terminología de git (cont.)

- pull: hacer un fetch seguido de un merge, con una rama remota dada.
- push: enviar los objetos de la rama local que no están en la rama remota a la que hace referencia el pull, y actualizar la cabeza de la rama remota.
 - Es la acción complementaria de pull.
 - Si la cabeza de la rama remota no es un ancestro de la cabeza de la rama local, el push falla*.

Operaciones en el área de trabajo

■ Clonar un repositorio “remoto”:

```
$ git clone git://git.moodle.org/moodle.git
$ git clone ssh://iarenaza@git.moodle.org/moodle.git
$ git clone http://git.moodle.org/moodle.git
$ git clone git@github.com:iarenaza/moodle.git
$ git clone /ruta/a/moodle.git /ruta/a/otro-moodle.git
$ git clone -o moodle.git git@github.com:iarenaza/moodle.git
```


Operaciones en el área de trabajo

- Incorporar nueva rama del repositorio remoto al repositorio local:

```
$ git fetch moodle.git
$ git branch mdl21-enrol-database-refactor \
  moodle.git/mdl21-enrol-database-refactor
```

- Enviar ramas locales al repositorio remoto:

```
$ git push moodle.git mdl21-nested-groups
$ git push moodle.git +mdl21-nested-groups
$ git push moodle.git mdl21-nested-groups:mdl21-nestgrp
```

- Configurar rama para poder hacer pull desde repositorio remoto:

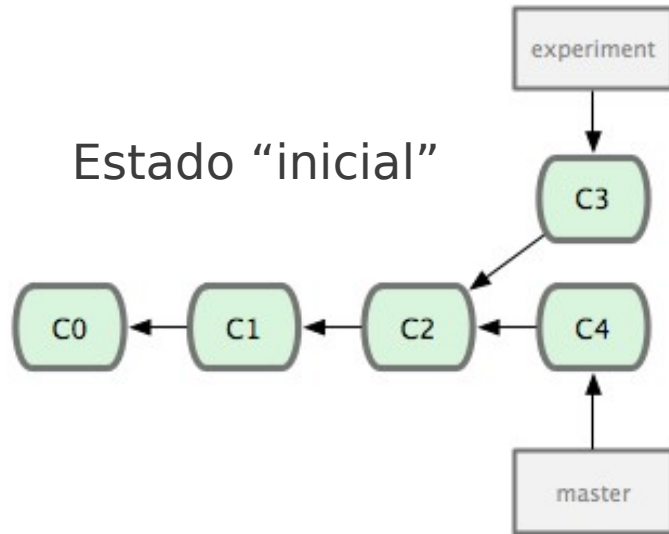
```
$ git config branch.mdl21-nested-groups.remote \
  moodle.git
$ git config branch.mdl21-nested-groups.merge \
  refs/heads/mdl21-nested-groups
```

Terminología de git (cont.)

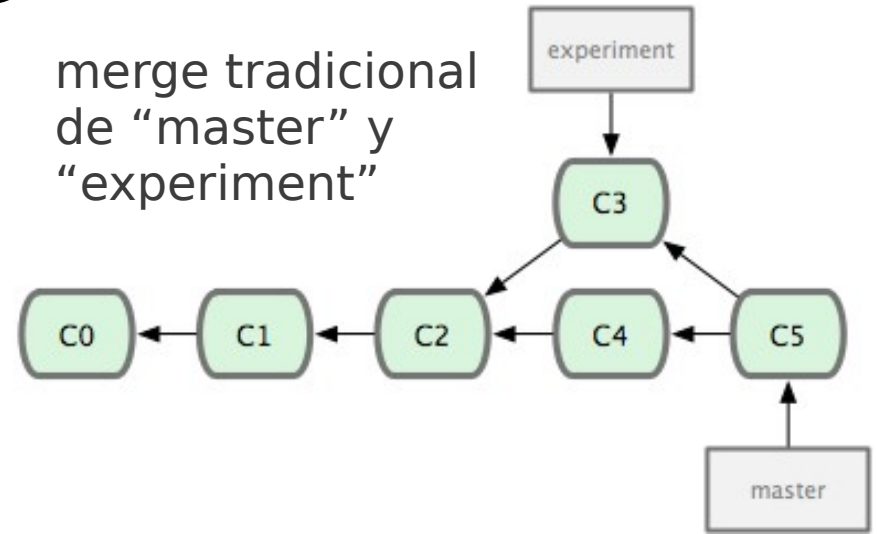
- rebase: re-aplicar una serie de cambios desde una rama en la cabeza de otra rama diferente, y hacer que la cabeza de esa otra rama apunte al resultado.
 - ¡OJO! Rescribe el historial de la rama.
 - Puede ser problemático en ramas publicadas en repositorios remotos.

rebase: "limpieza" del historial

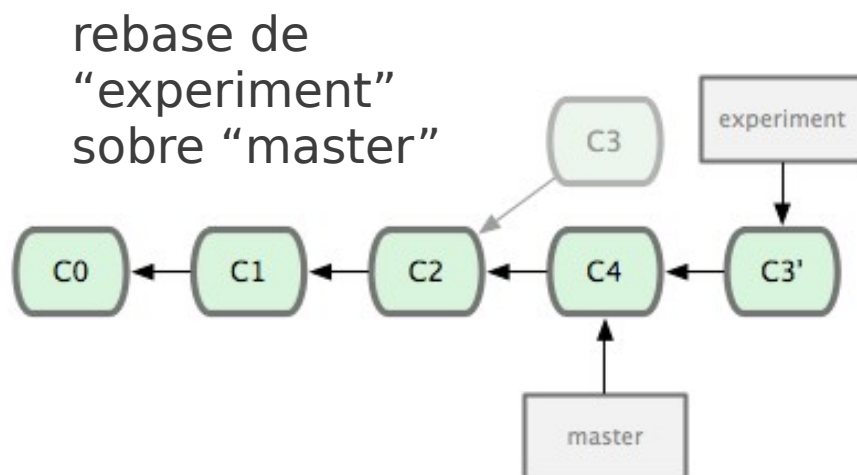
1



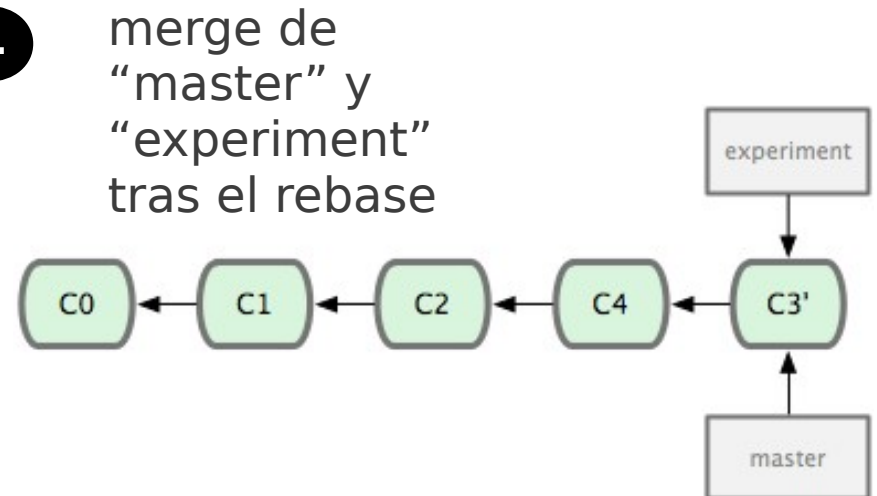
2



3



4



Limpieza del historial

IMPORTANTE: No hace rebase de los cambios de una rama si ésta ha sido publicada en otro repositorio.

Limpieza del historial

- Sólo se puede aplicar a la rama activa:

```
$ git checkout wip-mdl21-enrol-db-refactor
$ git rebase MOODLE_21_STABLE
```

- Si hay conflictos, solucionar a mano, y decirle a git qué hemos arreglado:

```
$ editar enrol/database/config.html
$ git add enrol/database/config.html enrol/database/enrol.php
$ editar xxxx
$ git add xxxxx
$ git rebase --continue
```

- Podemos abortar en todo momento:

```
$ git rebase --abort
```

Creación de parches y series

■ Creación de parches monolíticos:

```
$ git checkout mdl21-enrol-db-refactor  
$ git diff MOODLE_21_STABLE > mdl21-enrol-db-refactor.diff
```

■ Creación de series de parches:

```
$ git checkout mdl21-enrol-database-refactor  
$ git format-patch -o serie-enrol-db MOODLE_21_STABLE  
$ git format-patch -o -s -10 mdl21-enrol-db
```

Ejemplo de modelo de trabajo

Para mantener
modificaciones locales de
Moodle en Mondragon
Unibertsitatea

Premisas del modelo de trabajo

Repositorio 'compartido'
de referencia

Premisas del modelo de trabajo

Aprovechar el repositorio
de git.moodle.org

No usar 'git clone' para la
importación desde
git.moodle.org

Premisas del modelo de trabajo

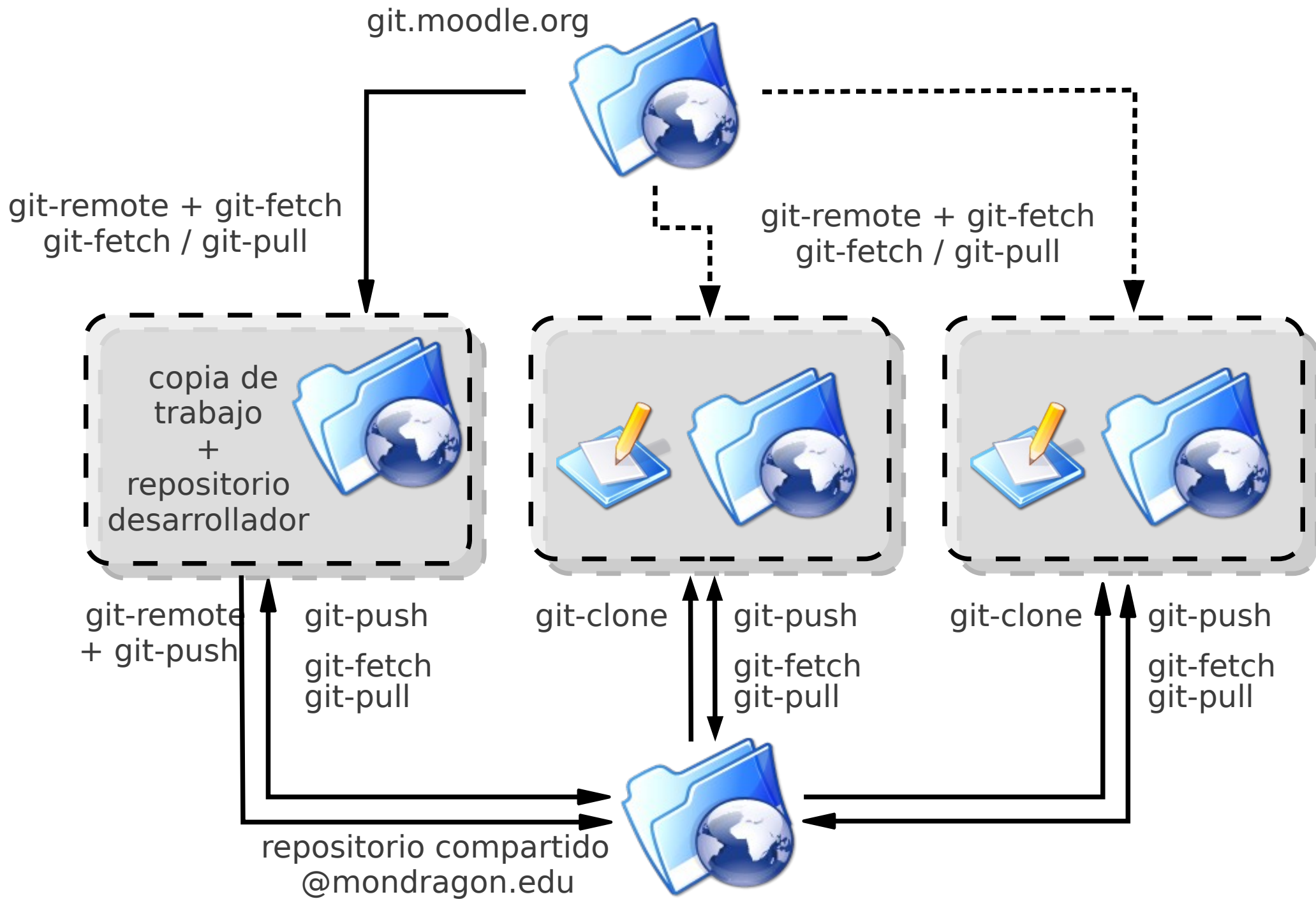
Repositorio compartido
sólo con ramas locales

Premisas del modelo de trabajo

Desarrollo siempre en las
ramas locales

Premisas del modelo de trabajo

Ramas estándar sólo en
repositorios de los
desarrolladores



Creación repositorio compartido

Crear repositorio primer desarrollador

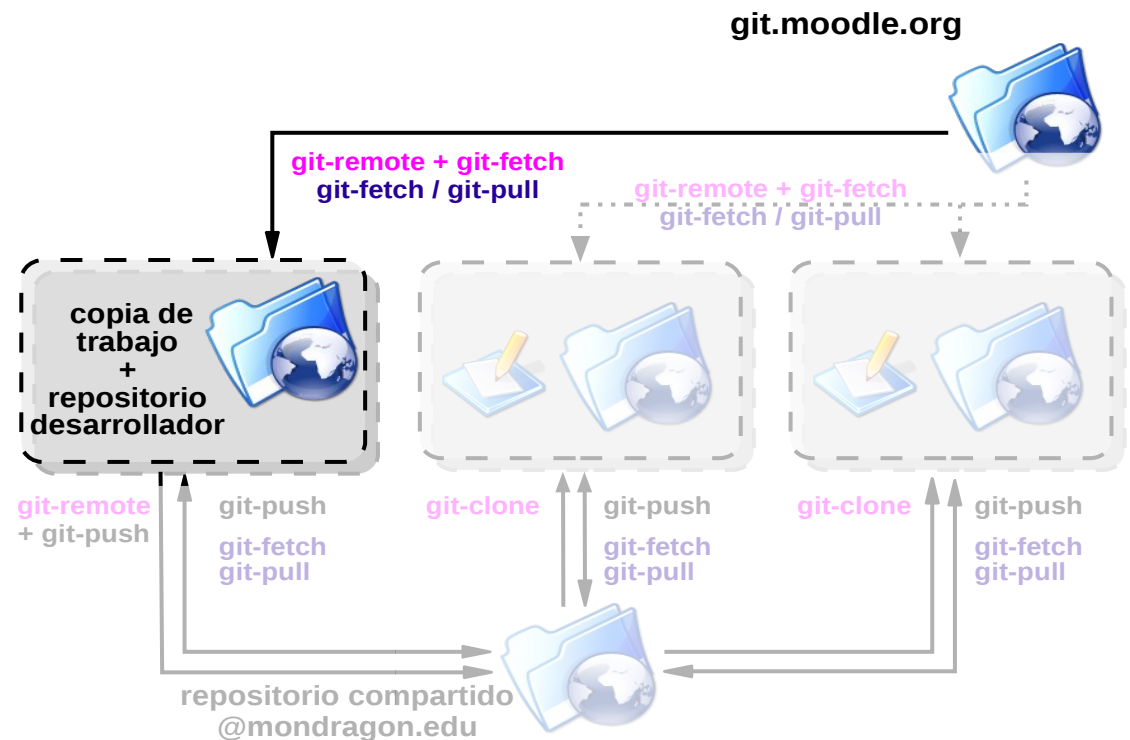
```
$ git config --global user.name 'Desarrollador-1'  
$ git config --global user.email 'desa-1@mondragon.edu'  
$ cd /ruta/repositorio/desarrollador  
$ mkdir desarrollador-1.git  
$ cd desarrollador-1.git  
$ git init
```



Creación repositorio compartido

Importar repositorio de moodle.org

```
$ git remote add -t master -t MOODLE_21_STABLE \  
  -m master moodle-org \  
  git://git.moodle.org/moodle.git  
$ git fetch moodle-org
```



Creación repositorio compartido

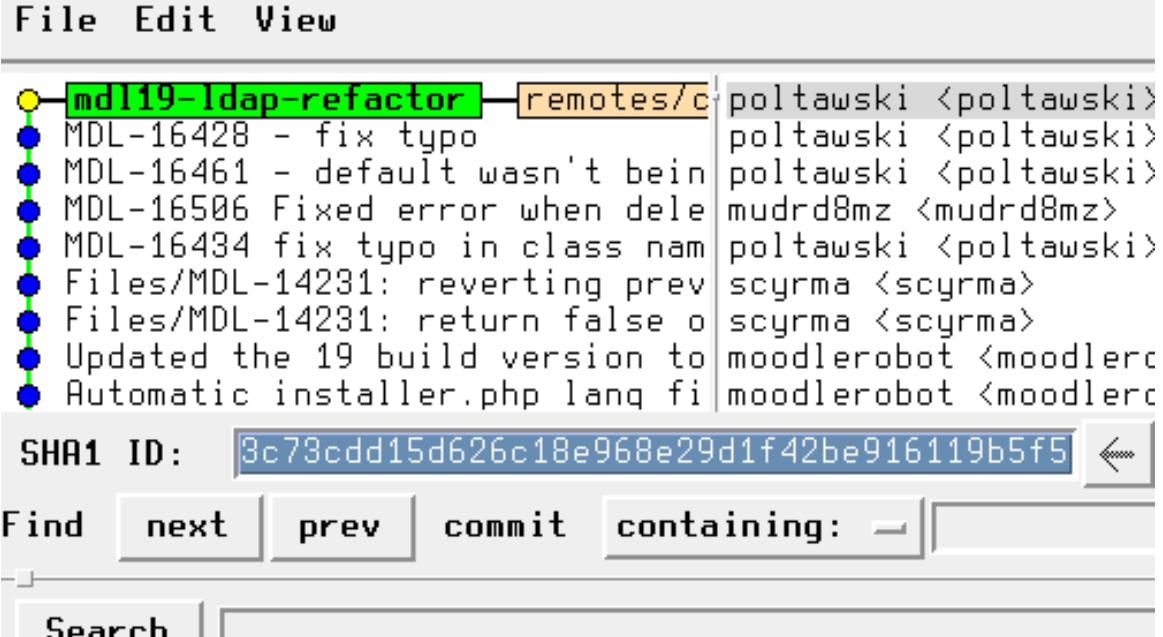
Ramas locales de seguimiento (opcional)

```
$ git branch --track master moodle-org/master
$ git branch --track MOODLE_21_STABLE \
  moodle-org/MOODLE_21_STABLE
```


Creación repositorio compartido

Crear ramas locales de trabajo

```
$ git branch mdl21-ldap-refactor moodle-org/MOODLE_21_STABLE
```



File Edit View

Branch	Author
● mdl19-ldap-refactor — remotes/c	poltawski <poltawski>
● MDL-16428 - fix typo	poltawski <poltawski>
● MDL-16461 - default wasn't bein	poltawski <poltawski>
● MDL-16506 Fixed error when dele	mudrd8mz <mudrd8mz>
● MDL-16434 fix typo in class nam	poltawski <poltawski>
● Files/MDL-14231: reverting prev	scyrma <scyrma>
● Files/MDL-14231: return false o	scyrma <scyrma>
● Updated the 19 build version to	moodlerobot <moodlerc>
● Automatic installer.php lang fi	moodlerobot <moodlerc>

SHA1 ID:

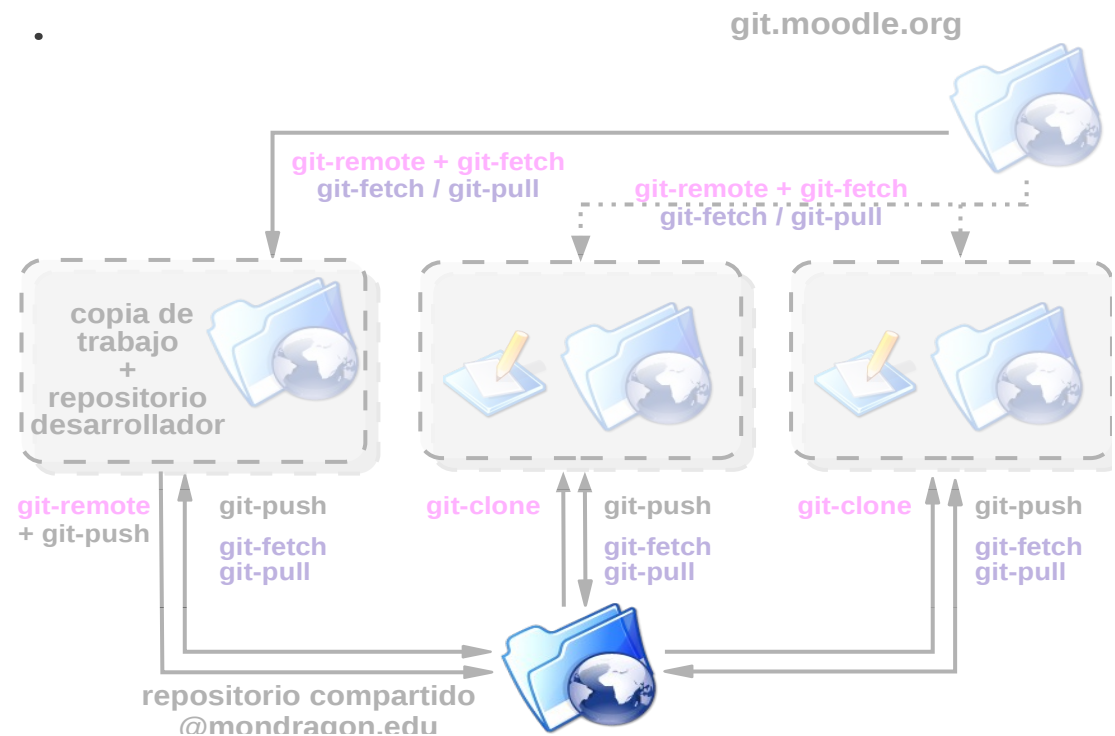
Find

Search

Creación repositorio compartido

Crear repositorio compartido

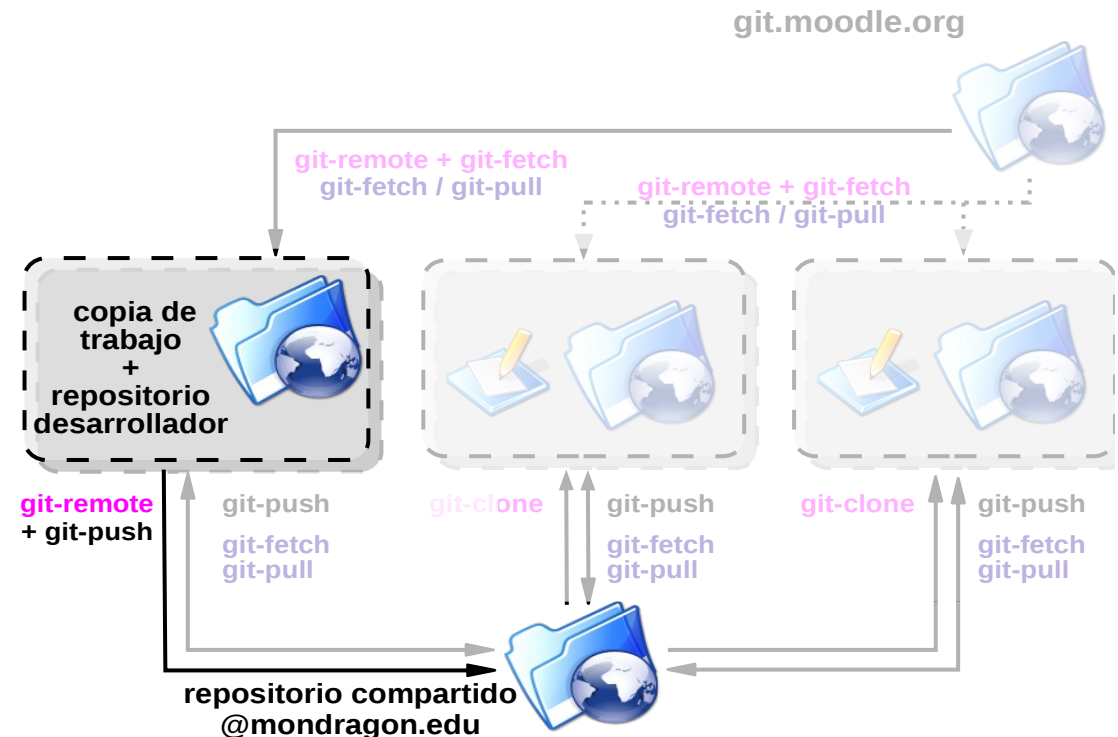
```
$ cd /ruta/repositorio/compartido
$ mkdir compartido.git
$ cd compartido.git
$ git --bare init --shared=all
$ chmod g=rwxs,o=rx .
$ sudo chgrp -R git-moodle .
```



Creación repositorio compartido

Enviar rama local al repositorio compartido

```
$ cd /ruta/repositorio/desarrollador/desarrollador-1.git  
$ git remote add compartido \  
  /ruta/repositorio/compartido/compartido.git  
$ git push compartido mdl21-ldap-refactor
```



Creación repositorio compartido

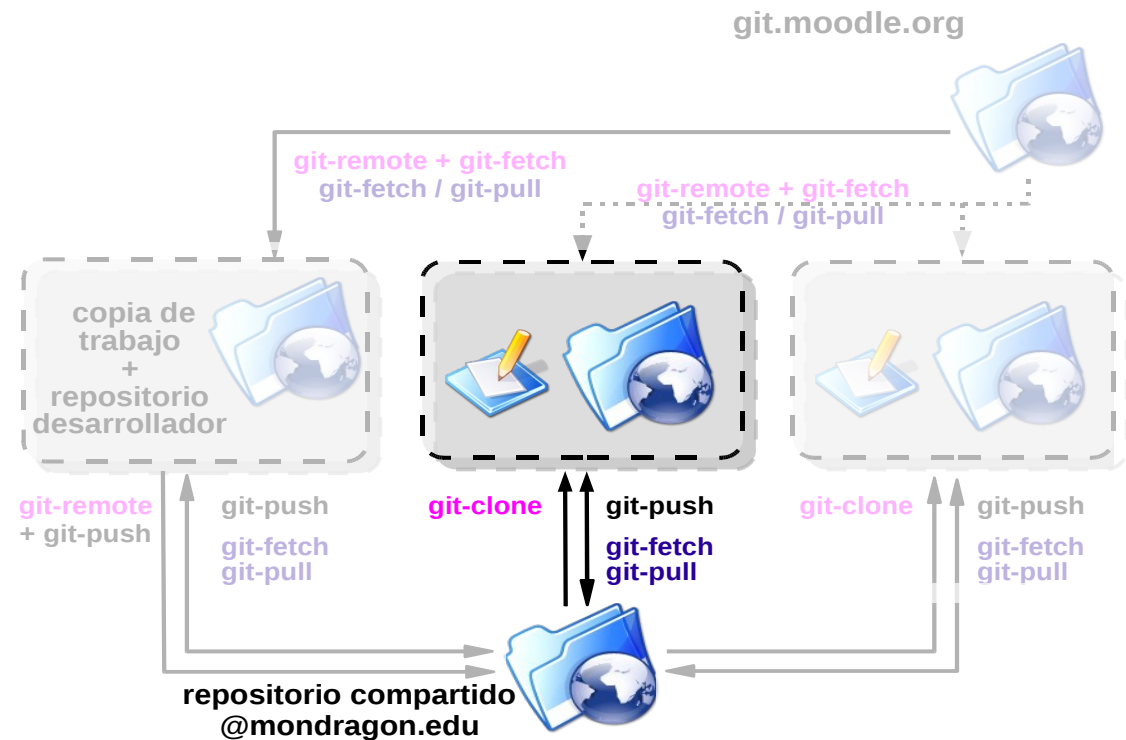
Configurar rama local para hacer pull desde repositorio compartido

```
$ git config branch.md121-ldap-refactor.remote \  
compartido  
$ git config branch.md121-ldap-refactor.merge \  
refs/heads/md121-ldap-refactor
```

Creación repo nuevo desarrollador

Clonar repositorio compartido

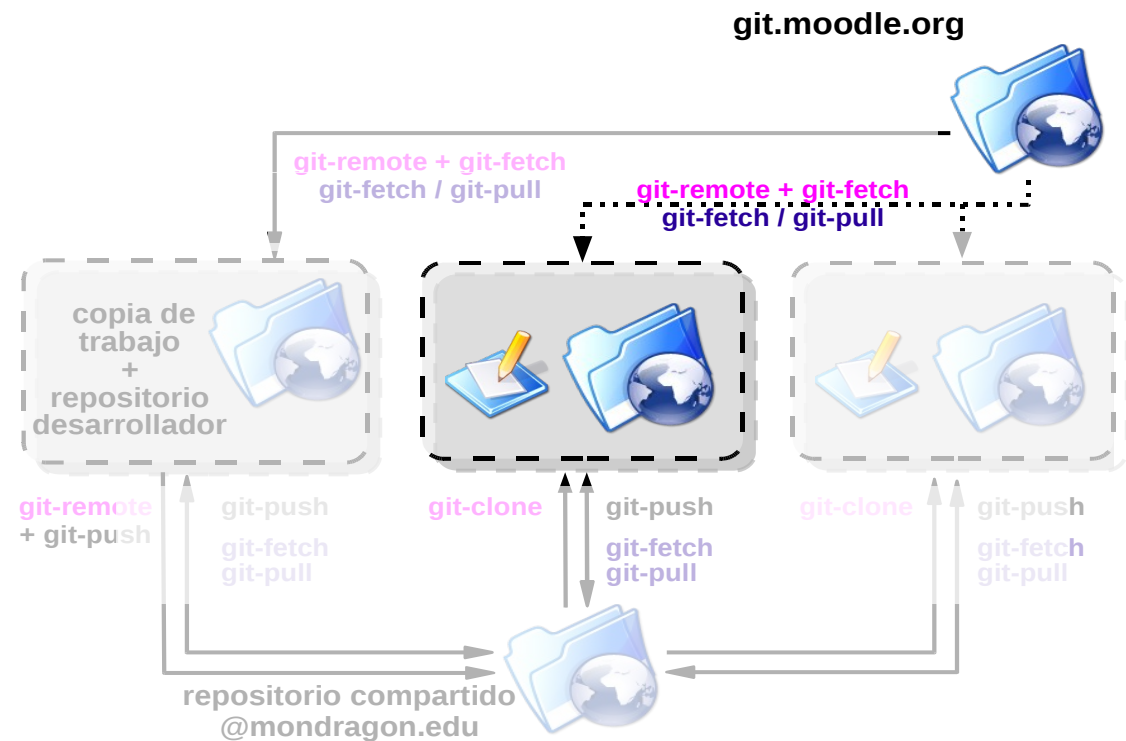
```
$ git config --global user.name 'Desarrollador-2'  
$ git config --global user.email 'desa-2@mondragon.edu'  
$ cd /ruta/repositorio/desarrollador  
$ git clone -o compartido \  
  /ruta/repositorio/compartido/compartido.git \  
  desarrollador-2.git
```



Creación repo nuevo desarrollador

Importar ramas estándar (opcional)

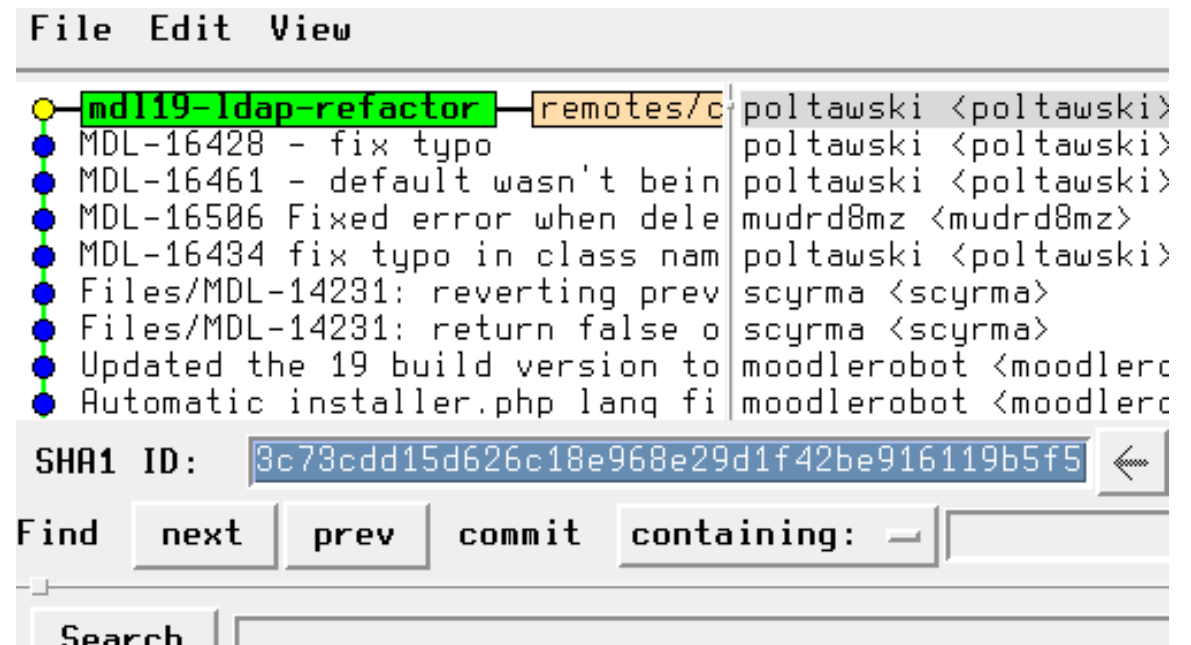
```
$ cd desarrollador-2.git
$ git remote add -t master -t MOODLE_21_STABLE \
  -m master moodle-org \
  git://git.moodle.org/moodle.git
$ git fetch moodle-org
```



Creación repo nuevo desarrollador

Crear ramas locales de trabajo

```
$ git branch mdl21-ldap-refactor \  
    compartido/mdl21-ldap-refactor  
$ git checkout mdl21-ldap-refactor
```



Algunos trucos bajo la manga (bonus track 😊)

- `git add --interactive`
- `git cherry-pick`
- `git stash [save | list | show | apply | remove | clear]`
- `git bisect`
- `git blame`
- `git gc, git prune, git fsck`
- `.git/hooks/*`

Algunas direcciones de interés

■ git Cheat Sheets:

- <http://devcheatsheet.com/tag/git/>

■ github/GITORIOUS:

- <http://github.com/>
- <http://gitorious.org/>

■ gitosis:

- <https://secure.wikimedia.org/wikibooks/en/wiki/Git/Gitosis>
- <http://thinkshout.com/blog/2011/03/lev/redmine-and-gitosis-project-management-nirvana>

■ Gerrit:

- <https://code.google.com/p/gerrit/>

■ Jenkins:

- <http://jenkins-ci.org/>