Control de versiones con git

Más allá del commit

Mario Sánchez Prada
msanchez@igalia.com

A Coruña, 28 de Marzo de 2011

"I think Git overshadows any other achievement that Linus guy ever had. He should stop working on that hobby kernel, and put more time into Git, I think it has potential."

Eitan Isaacson
Open Source developer
October 21st, 2009

Acerca de mí

- Ingeniero en Informática por la *UDC*
- Miembro de la empresa *Igalia*
- Desarrollador de Software Libre (*GNOME* & *Maemo*)

Acerca de mí

- Ingeniero en Informática por la *UDC*
- Miembro de la empresa *Igalia*
- Desarrollador de Software Libre (*GNOME* & *Maemo*)
- **Usuario de *git***

Acerca de *git*

- Sistema de Control de Versiones **Distribuido** (*DVCS*)
- Inicialmente escrito en C por Linus Torvalds
- Sustituto de *BitKeeper* en el desarrollo del kernel de Linux
- Ampliamente usado hoy en día

`http://git-scm.com`

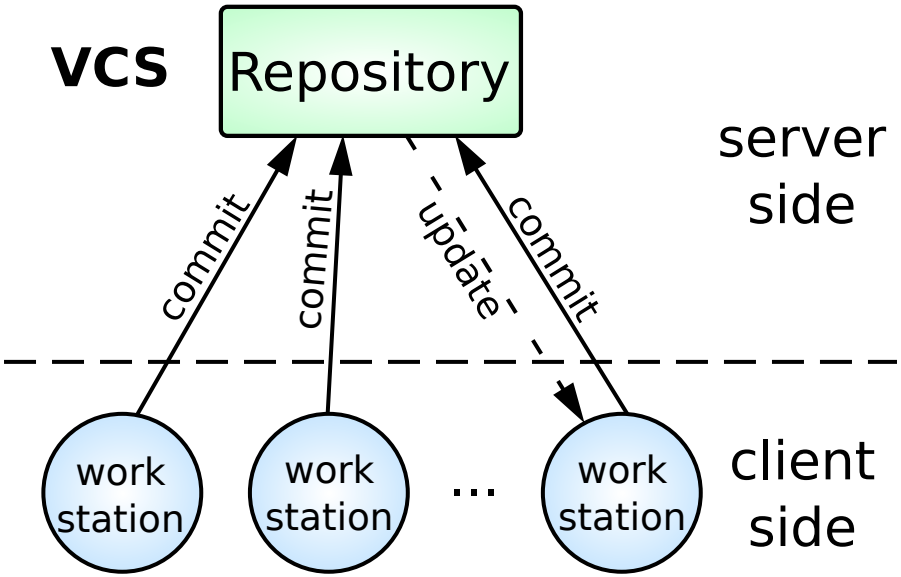
Conceptos básicos

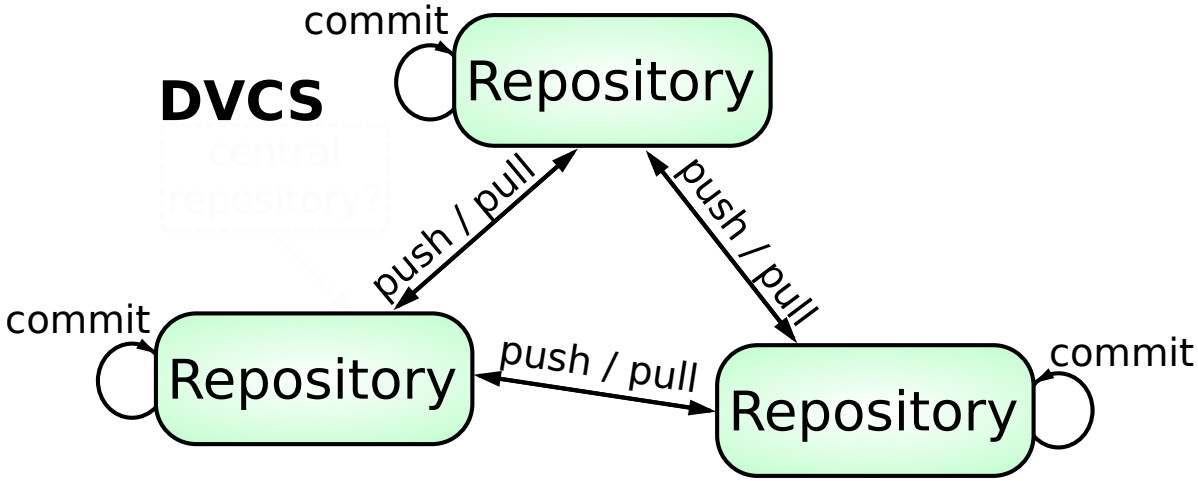
El problema

Gestión de cambios en un fichero, proyecto...

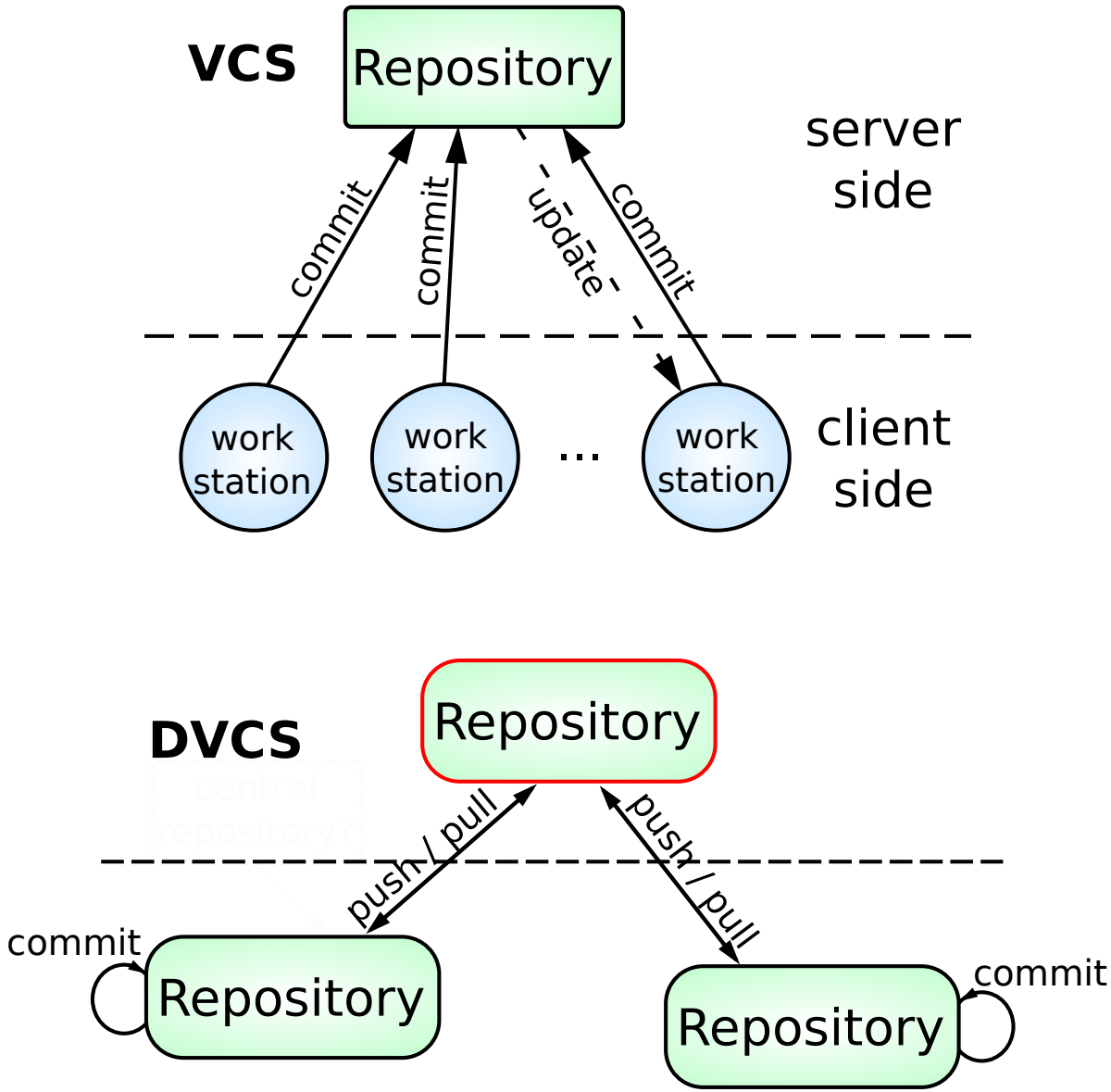
- Opción 0: no hacer nada (versión única)
- Opción 1: Sistemas rudimentarios (copias manuales)
  - `Proyecto`
  - `Proyecto.bckp`
  - `Proyecto-20100514.bckp`
  - `Proyecto-20100514.bckp2`
  - `Proyecto-20100514.bckp2-final`
  - `Proyecto-20100514.bckp2-definitivo`
  - `Proyecto-20100514.bckp2-definitivo-final`
  - `Proyecto-????????`
- Opción 2: *VCS's* "tradicionales": *CVS, Subversion*...
  - Requieren normalmente conexión a un repositorio central
- Opción 3: *DVCS's*: *Bazaar*, *Mercurial*, *Monotone*, ***git*** ...
  - Autonomía y flexibilidad **total** (i.e. *commits* locales)

VCS vs DVCS

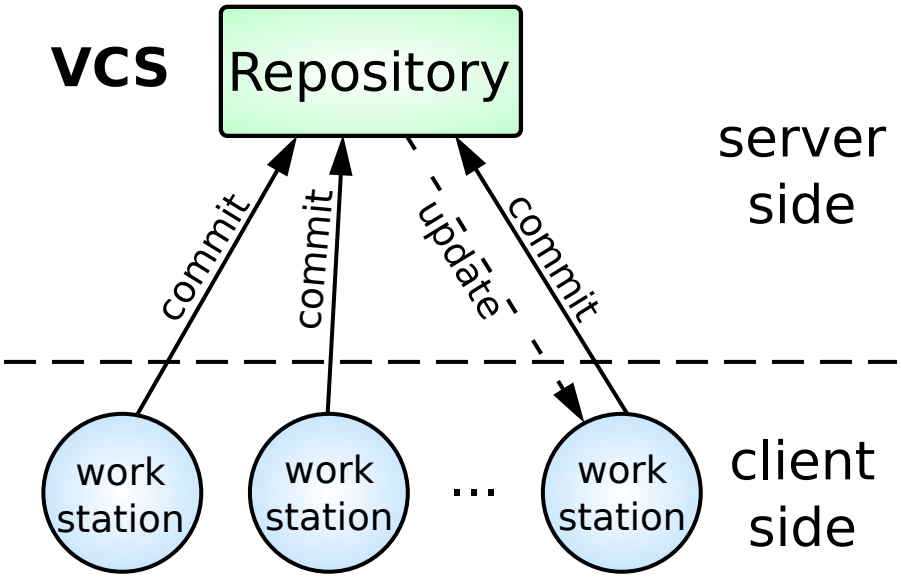

VCS
Repository
server side
commit
commit
update
commit
work station
work station
...
work station
client side

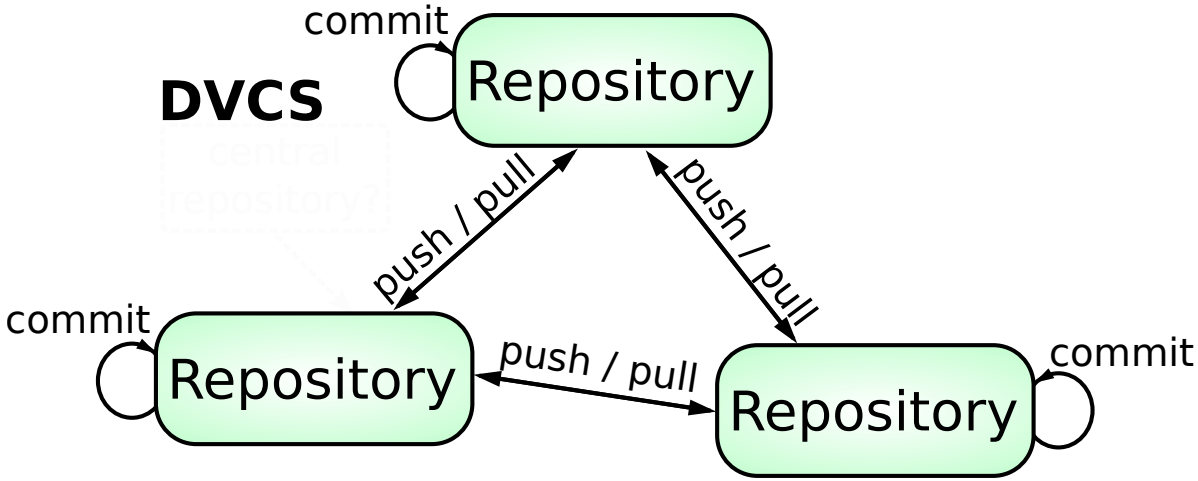

commit
DVCS
Repository
central repository?
push / pull
push / pull
commit
Repository
push / pull
Repository
commit

VCS vs DVCS


VCS
Repository
server side
commit
commit
update
commit
work station
work station
...
work station
client side
DVCS
Repository
push / pull
push / pull
commit
Repository
Repository
commit

VCS vs DVCS


VCS
Repository
server side
commit
commit
update
commit
work station
work station
...
work station
client side


commit
DVCS
Repository
central repository?
push / pull
push / pull
commit
Repository
push / pull
Repository
commit

Ventajas (y desventajas) de los DVCS

Ventajas

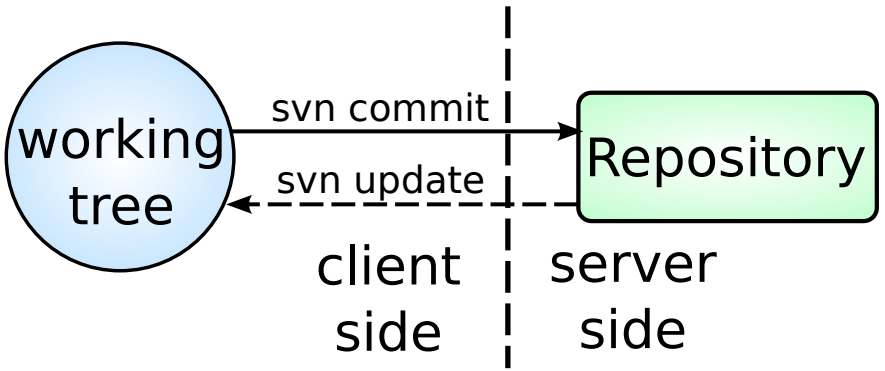
- Repositorio local completo (versatilidad, rapidez...)
- Mejora el *workflow* para trabajos colaborativos
- Integración de cambios desde fuentes externas
- Gestión de versiones, "forks", experimentos...
- Separación clara entre trabajo privado y público
- *Backups* implícitos
- No necesitan conexión a internet permanente

Desventajas

- Curva de aprendizaje mayor que en los *VCS*
- Importación inicial más lenta (repo completo)
- Requieren disciplina mayor para su uso (muchas posibilidades)

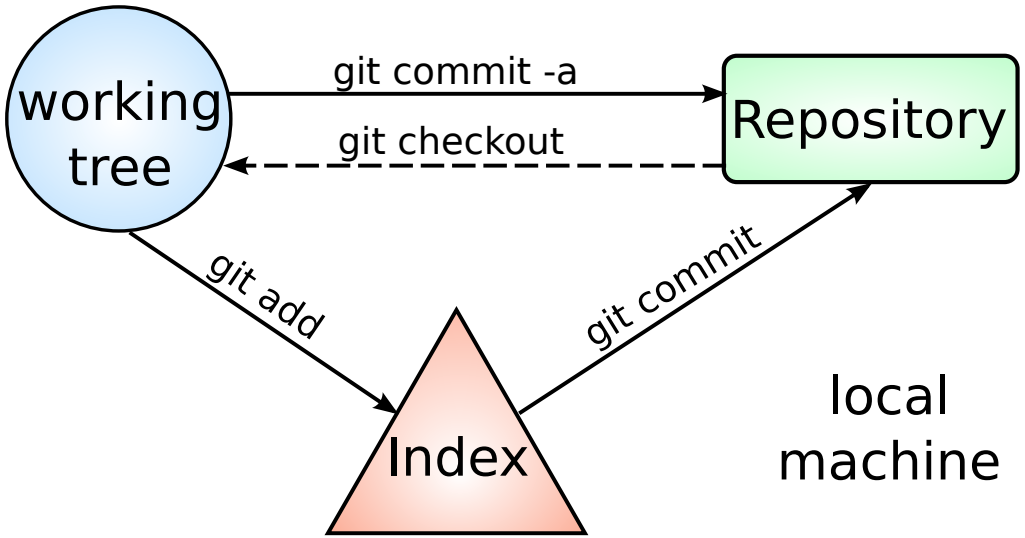
Subversion (VCS) vs Git (DVCS)

Subversion



Requires the two client / server parts to work

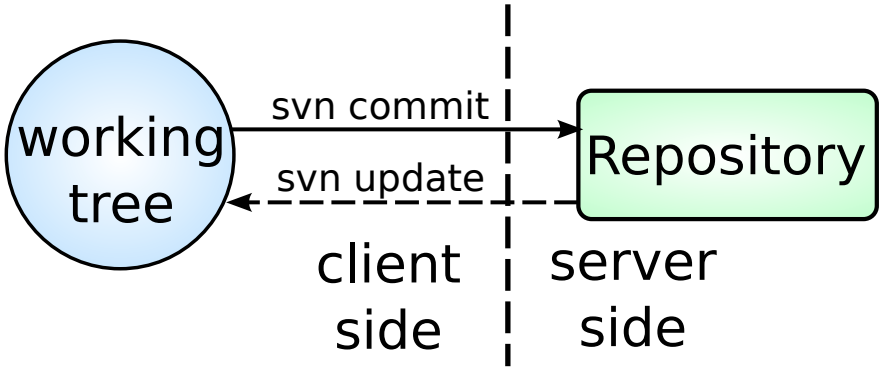
Git



Doesn't require anything else but the local machine itself

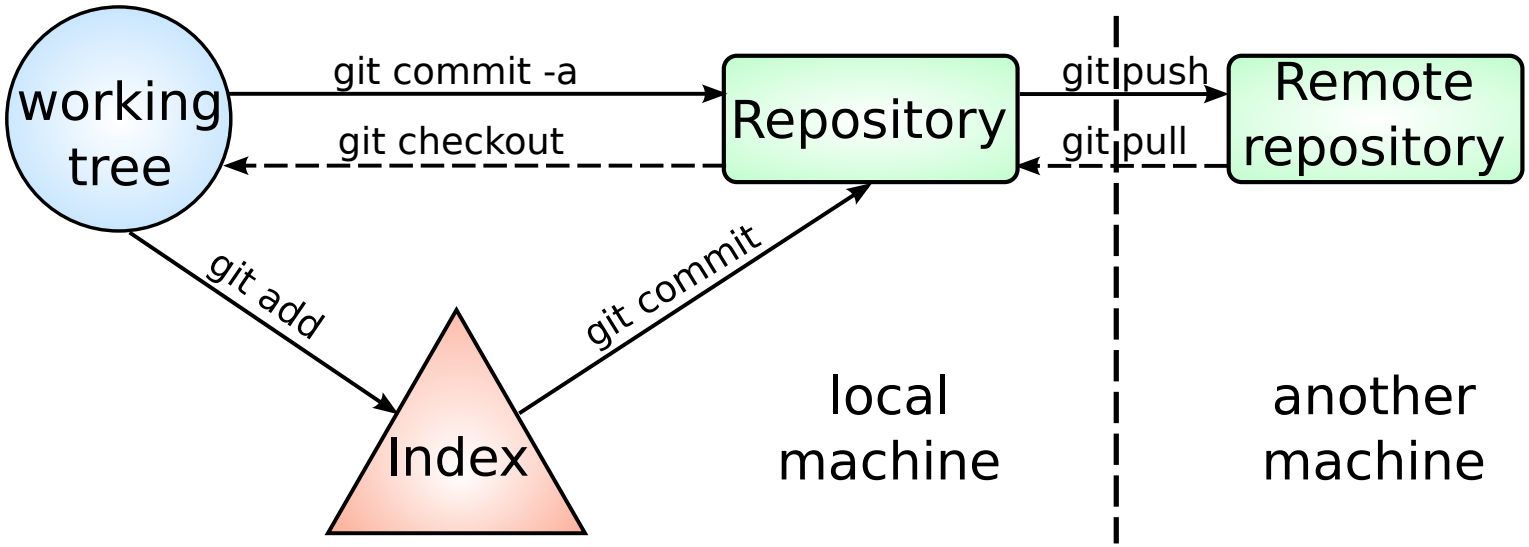
Subversion (VCS) vs Git (DVCS)

Subversion

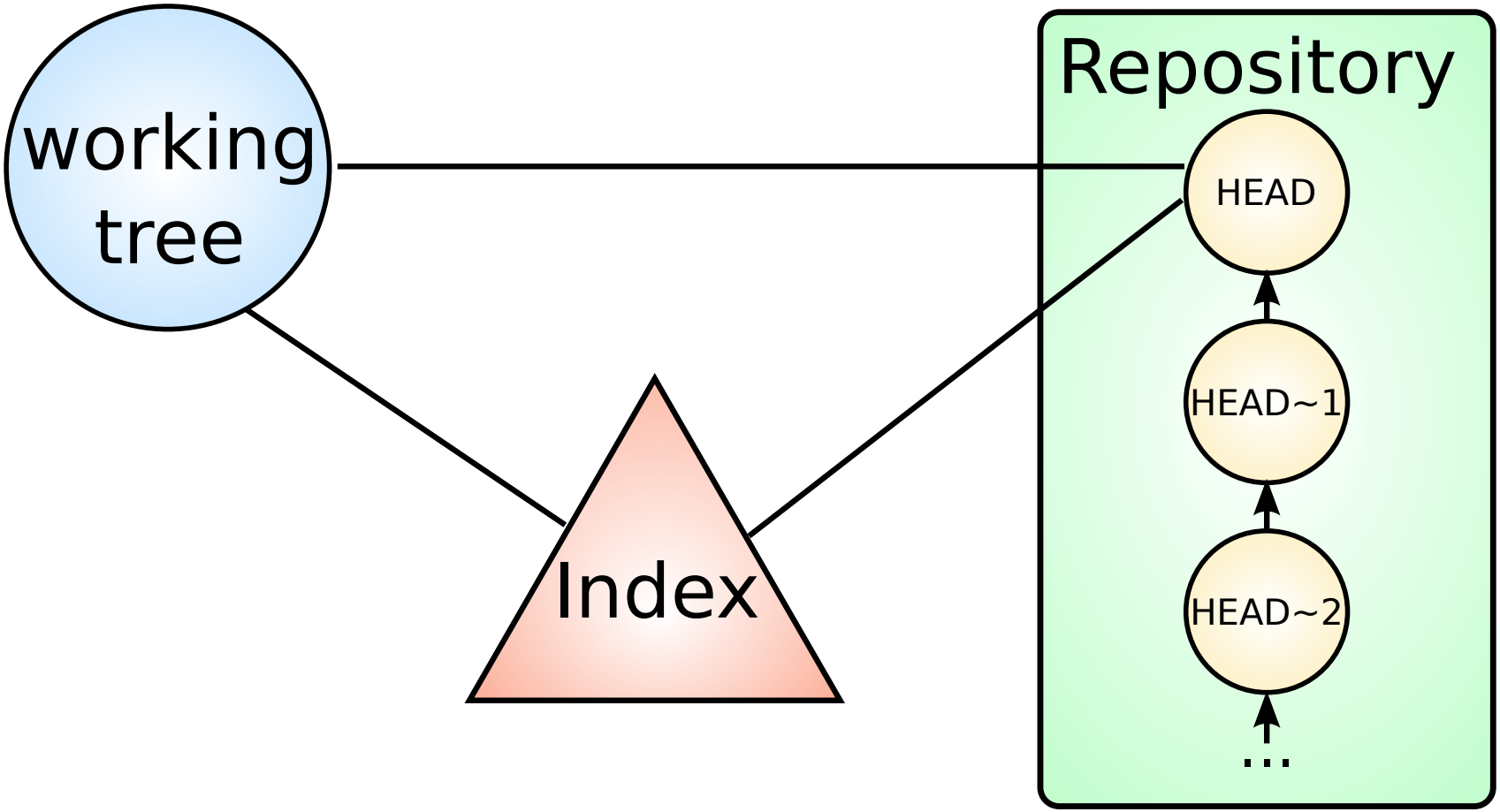


Requires the two client / server parts to work

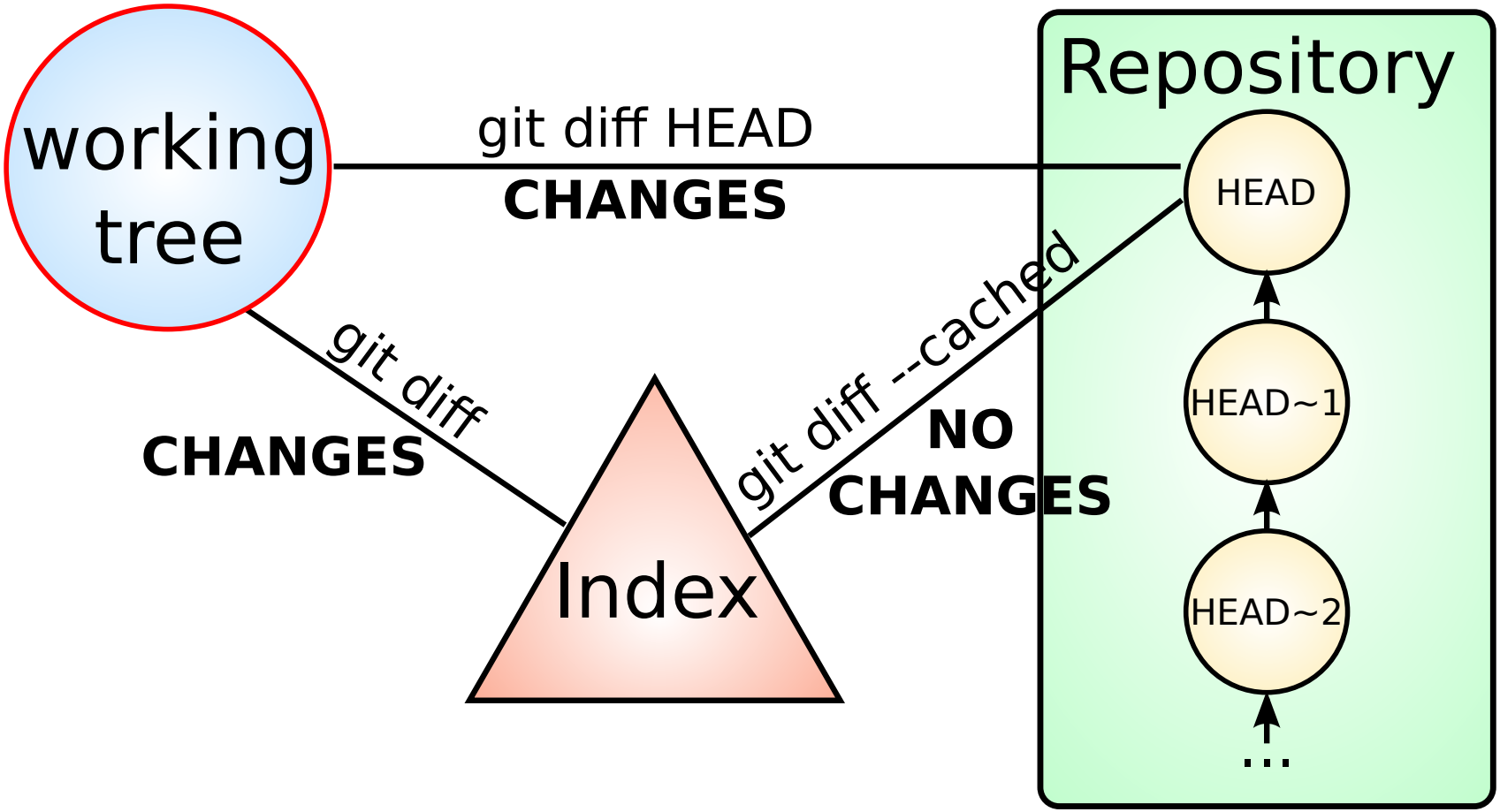
Git



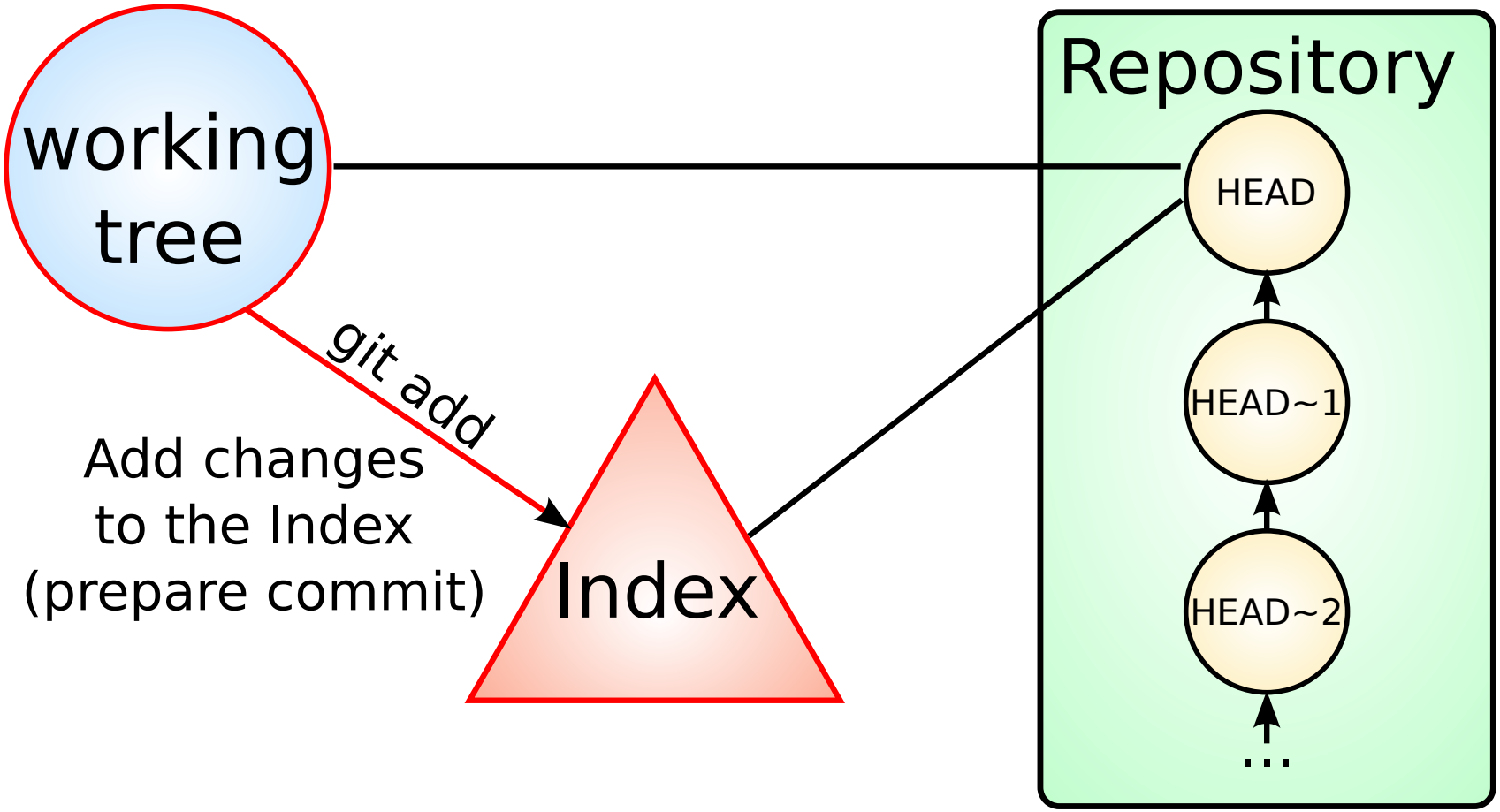
Workflow típico en git



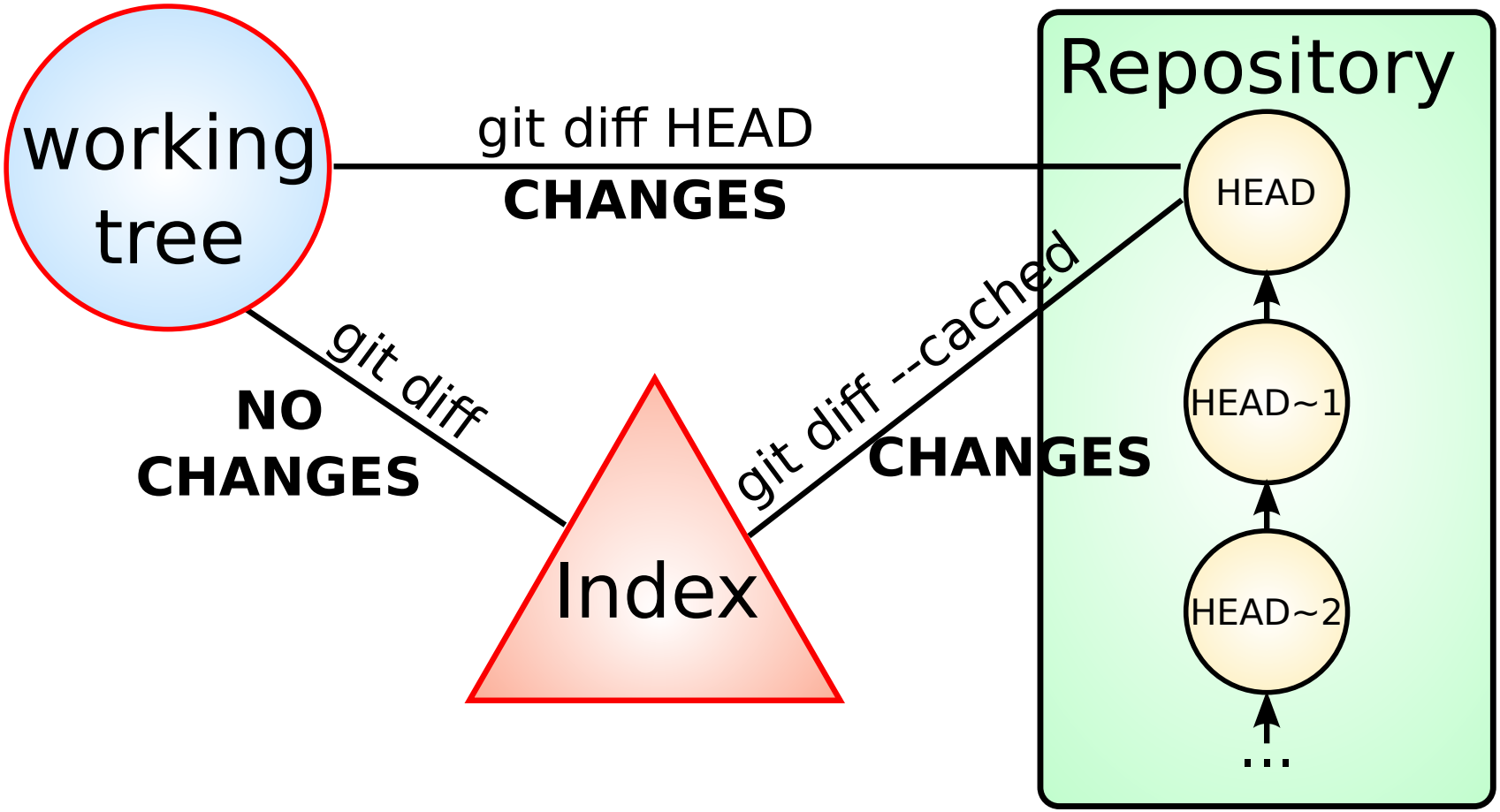
Workflow típico en git



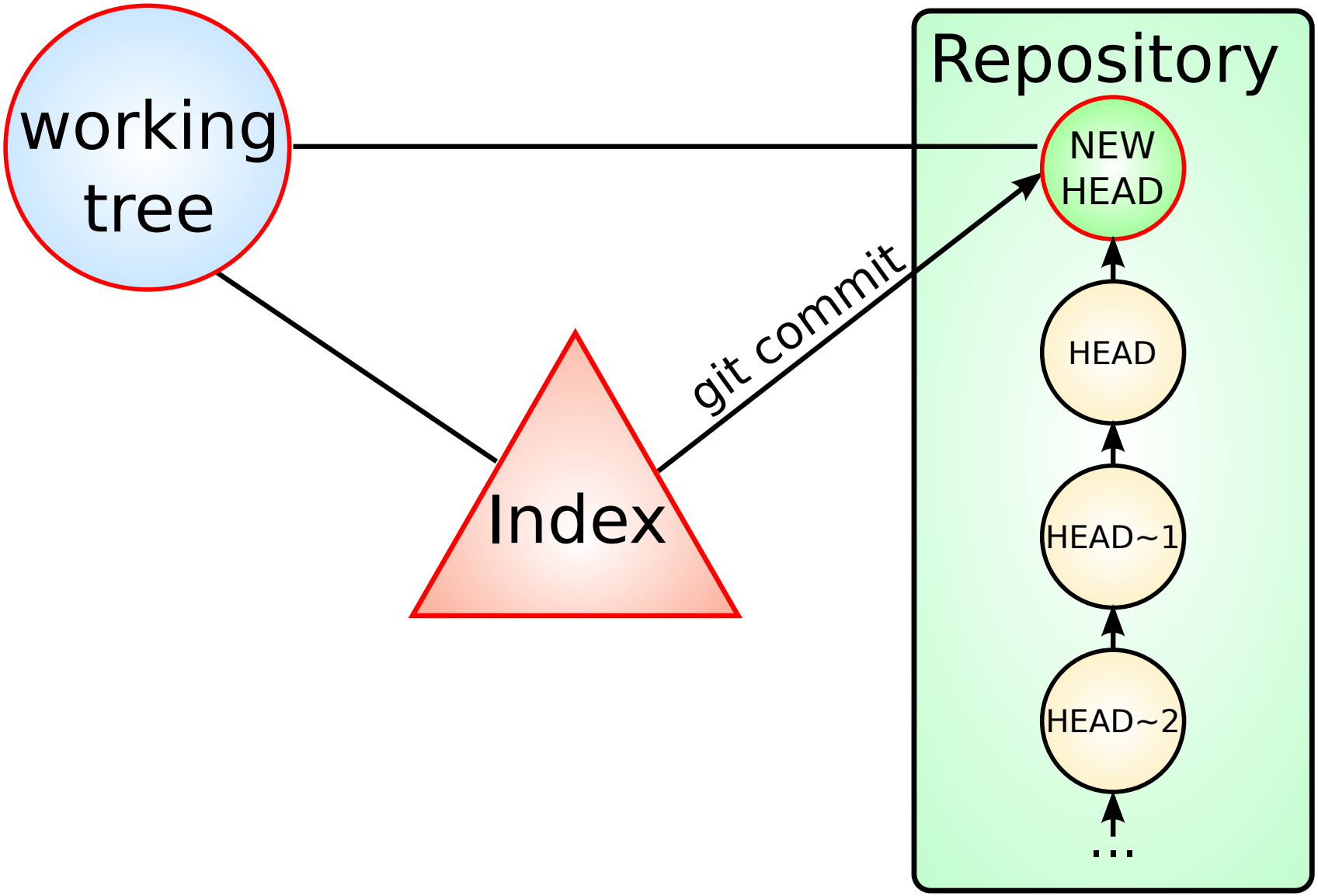
Workflow típico en git



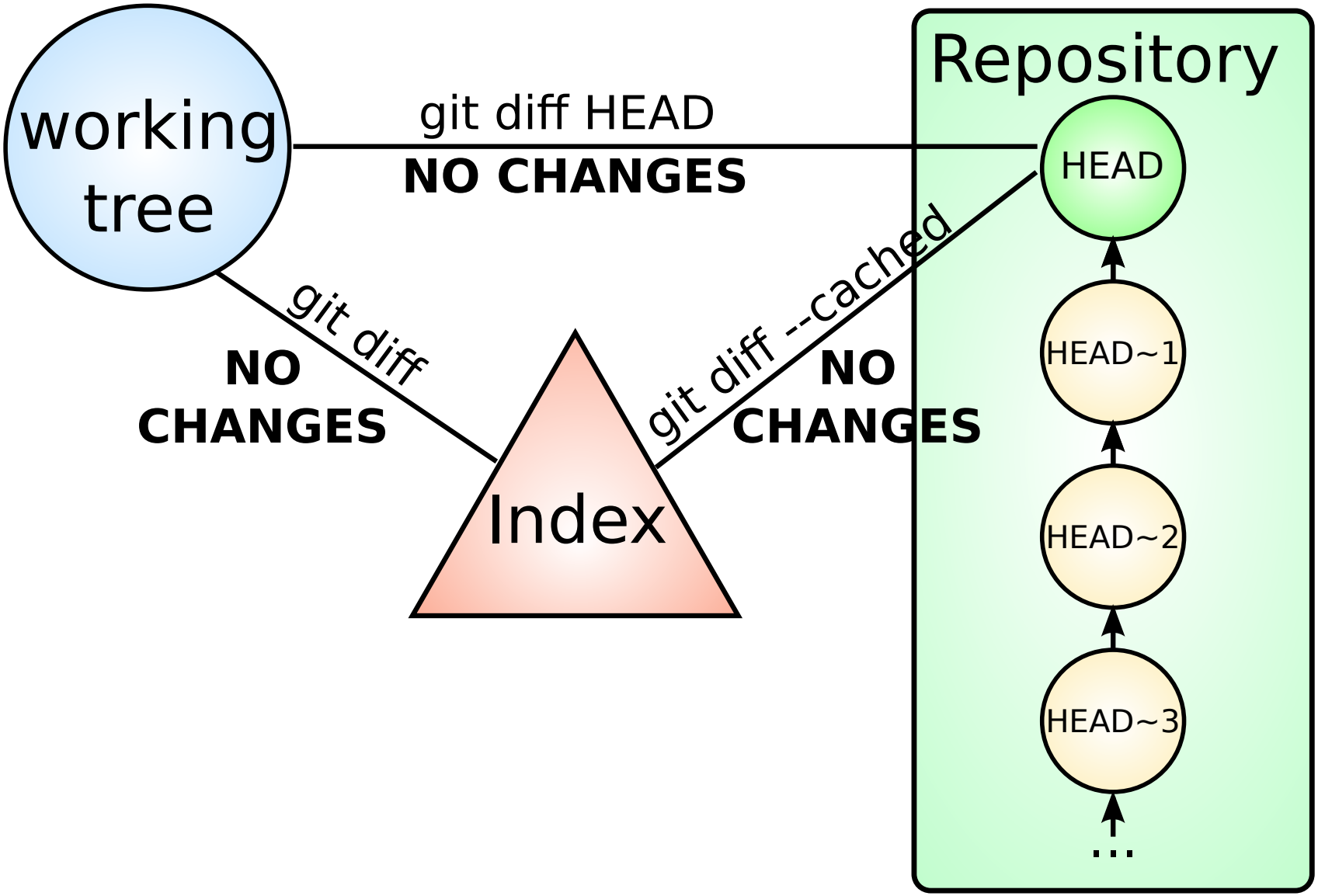
Workflow típico en git



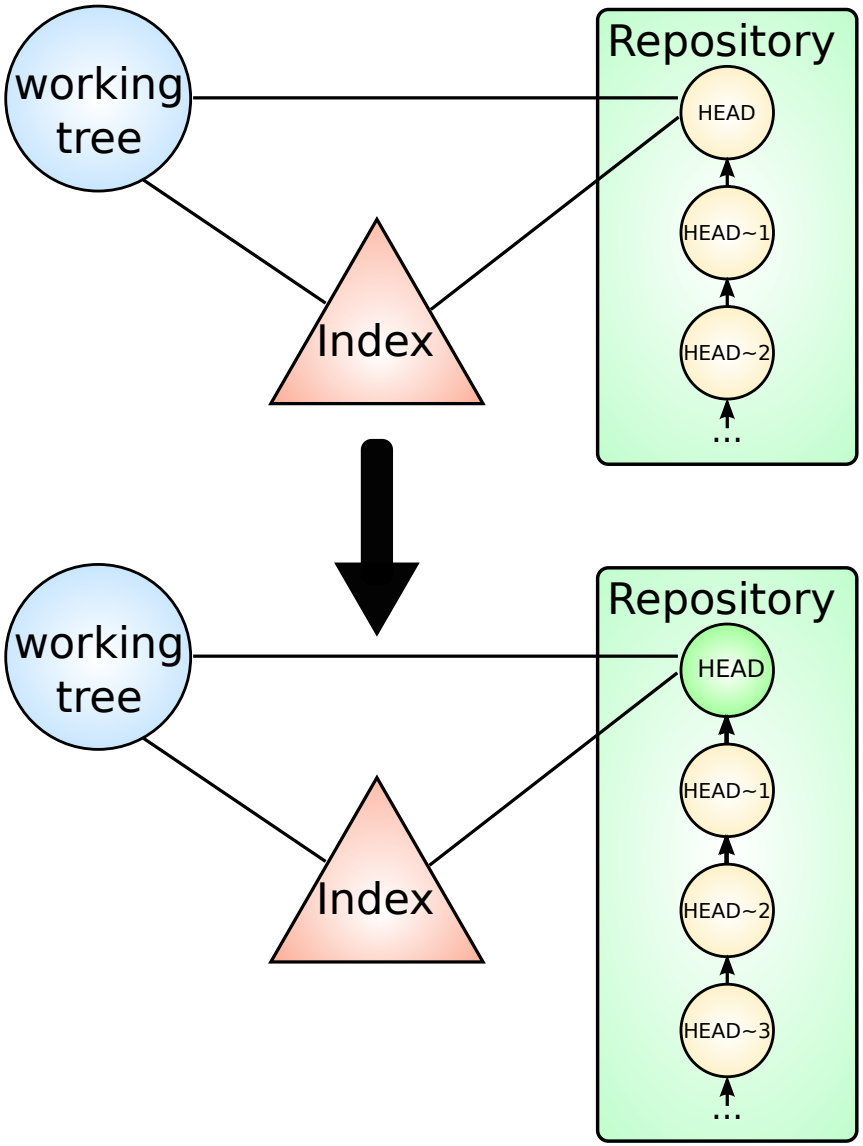
Workflow típico en git



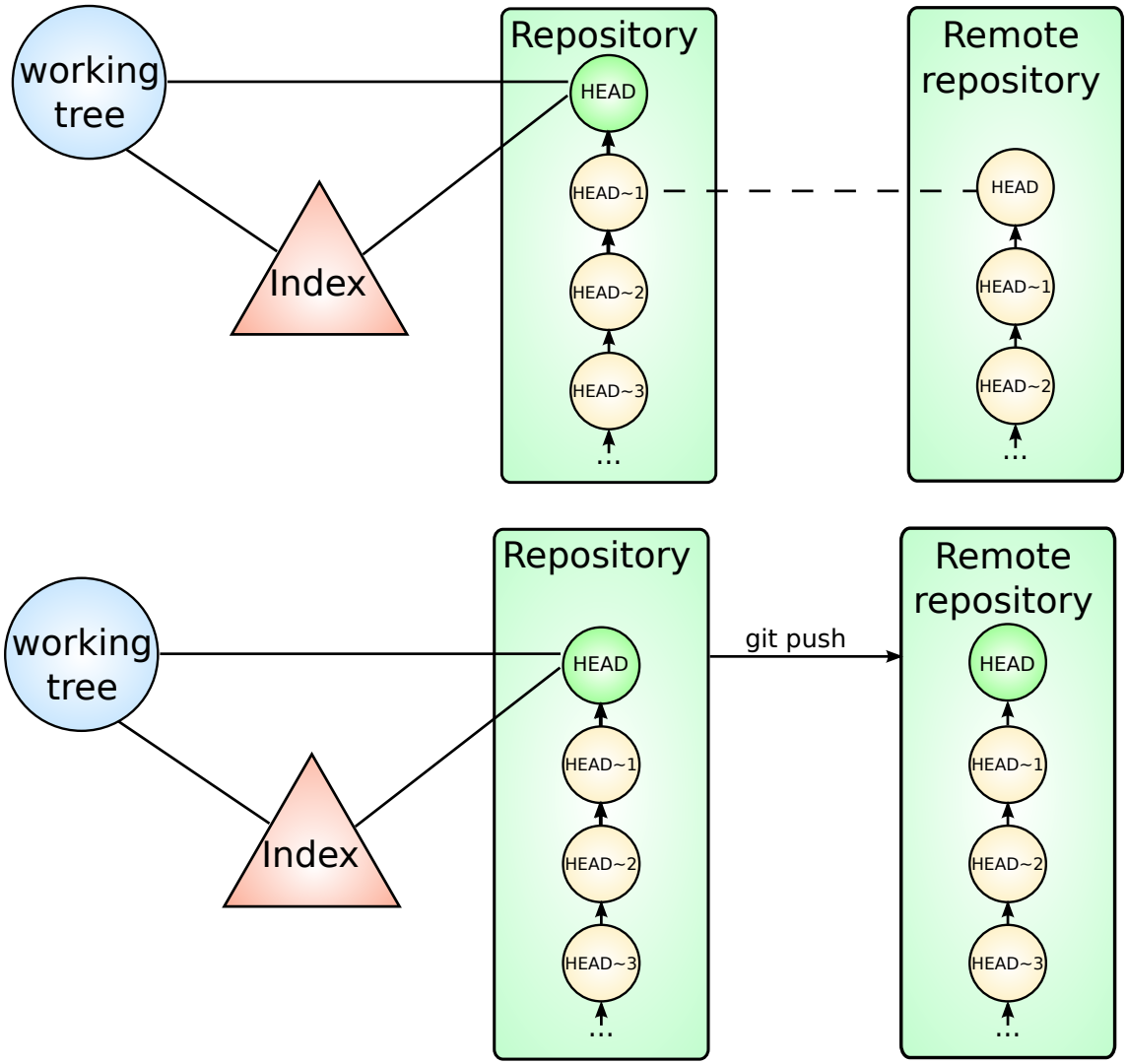
Workflow típico en git



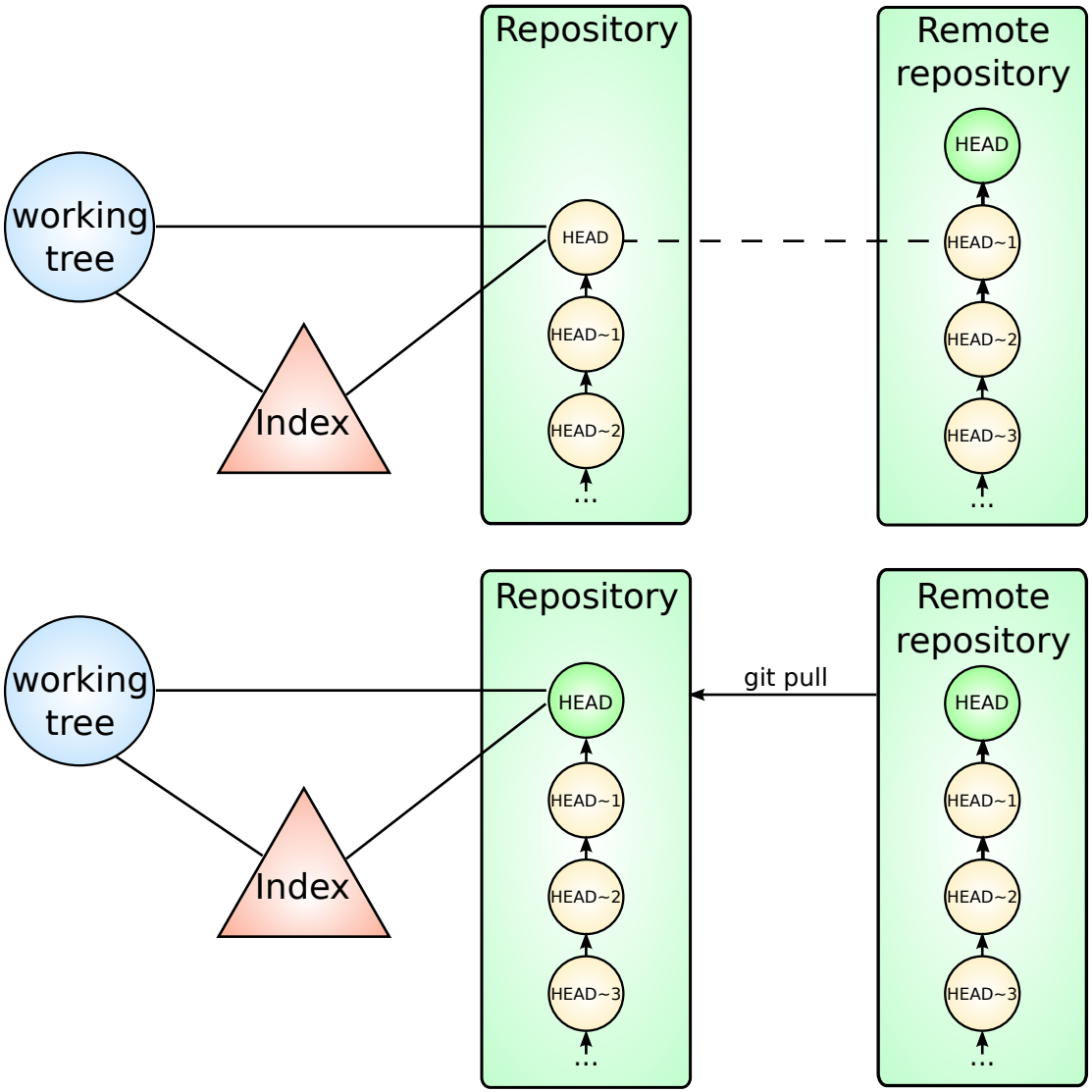
Workflow típico en git



Workflow típico en git - git push



Workflow típico en git - git pull



Workflow típico en git: comandos

Comandos básicos:

- `git init`: Inicializa el directorio para trabajar con *git*
- `git add`: Añade cambios en el *Working Tree* al *Index*
- `git commit`: Consolida en el repositorio los cambios del *Index*, creando un nuevo commit y actualizando el *HEAD*.
- `git diff`: Cambios entre el *Working Tree* y el *Index*
- `git show <commit>`: Visualiza el *commit* `commit`
- `git log`: Muestra la historia (*log*) desde *HEAD*

Workflow típico en git - Un ejemplo concreto

```
$ mkdir /tmp/git
$ cd /tmp/git

$ git init .                    # Initialize repository
Initialized empty Git repository in /tmp/git/.git/

$ touch A B C
$ git add A B C                 # Add new files to Index

$ git commit -m "Initial commit"  # First commit!
[master (root-commit) fa16ae4] Initial commit
 0 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 A
 create mode 100644 B
 create mode 100644 C
```

Workflow típico en git - Un ejemplo concreto

```
$ git diff --stat           # Working Tree <-> Index
$ git diff --stat HEAD      # Working Tree <-> HEAD
$ git diff --stat --cached  # Index <-> HEAD

  <<< MAKE SOME CHANGES IN THE WORKING TREE >>>

$ git diff --stat           # Working Tree <-> Index
 A |    3 +++
 C |    3 +++
 2 files changed, 3 insertions(+), 0 deletions(-)

$ git diff --stat HEAD      # Working Tree <-> HEAD
 A |    3 +++
 C |    3 +++
 2 files changed, 3 insertions(+), 0 deletions(-)

$ git diff --stat --cached  # Index <-> HEAD
```

Workflow típico en git - Un ejemplo concreto

```
$ git add A C                    # Move changes to Index

$ git diff --stat                # Working Tree <-> Index

$ git diff --stat HEAD           # Working Tree <-> HEAD
 A |    3 +++
 C |    3 +++
 2 files changed, 3 insertions(+), 0 deletions(-)

$ git diff --stat --cached  # Index <-> HEAD
 A |    3 +++
 C |    3 +++
 2 files changed, 3 insertions(+), 0 deletions(-)
```

Workflow típico en git - Un ejemplo concreto

```
$ git commit -m "My first patch :-)"   # Commit changes
[master 614c4e2] My first patch :-)
 2 files changed, 3 insertions(+), 0 deletions(-)

$ git diff --stat           # Working Tree <-> Index
$ git diff --stat HEAD      # Working Tree <-> HEAD
$ git diff --stat --cached  # Index <-> HEAD

$ git show --stat HEAD      # Show last commit's stats
commit 614c4e224284e2719fe040b64685b790606000a6
Author: Mario Sanchez Prada <msanchez@igalia.com>
Date:   Mon May 3 09:31:19 2010 +0200

    My first patch :-)

 A |    3 +++
 C |    3 +++
 2 files changed, 3 insertions(+), 0 deletions(-)
```

Workflow típico en git - Un ejemplo concreto

```
$ git log                          # Show history up-to-date
commit 614c4e224284e2719fe040b64685b790606000a6
Author: Mario Sanchez Prada <msanchez@igalia.com>
Date:   Mon May 3 09:31:19 2010 +0200

    My first patch :-)

commit fa16ae4ff64b3b968ca8991c90f123e62a6a009c
Author: Mario Sanchez Prada <msanchez@igalia.com>
Date:   Mon May 3 09:30:11 2010 +0200

    Initial commit
```

Workflow típico en git (comandos)

Otros comandos útiles:

- `git rm <file>`: Elimina `file` y prepara el *Index* con los cambios para el próximo *commit*.
- `git mv <fileA> <fileB>`: Renombra `fileA` como `fileB` y prepara el *Index* con los cambios.
- `git status`: Muestra el estado del *Working Tree* y el *Index*
- `git clone <URL>`: Clona un repositorio remoto
- `git push`: Envía cambios locales a un repositorio remoto.
- `git pull`: Trae cambios de un repositorio remoto al local.

Workflow típico en git (comandos): git status

```
<<< RIGHT BEFORE ADDING A AND C TO INDEX WITH 'git add A C' >>>

$ git status
# On branch master
# Changed but not updated:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
# modified:   A
# modified:   C
#
no changes added to commit (use "git add" and/or "git commit -a")

$ git add A

$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
# modified:   A
#
# Changed but not updated:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
# modified:   C
#
```

Comparación comandos Subversion - Git

Subversion	Git

`svnadmin create` *repo*
`svn import` *filepath*
	`git init`
	`git add` *filepath*
	`git commit`

`svn checkout` *URL*	`git clone` *URL*
`svn add|rm|mv` *file*	`git add|rm|mv` *file*
`svn diff`	`git diff`
`svn commit`	`git commit -a` `git push`
`svn update`	`git pull`
`svn status`	`git status`
`svn log`	`git log`
`svn blame` *file*	`git blame` *file*

Repositorios y ramas

Repositorios

Dos tipos de repositorios:

- **Repositorio local**: Directorio local de trabajo bajo control de versiones con *git* (contiene un directorio `.git/`).

  Se "crean" mediante alguno de los siguientes pasos:

  - `git init`: Inicializa el directorio actual para ser utilizado con *git* (necesario hacer `git add <file>` y git commit)
  - `git clone <URL>`: Clona un repositorio remoto en un directorio local, manteniendo un "enlace" con el original a través del repositorio remoto `origin`.

- **Repositorios remotos**: Cualquier repositorio ajeno el cual tenemos clonado en nuestro repositorio local de trabajo

  Se referencian con `git remote add <name> <URL>`
  `git clone <URL>` crea uno automáticamente: `origin`

Ramas

Una *rama* es una línea de trabajo paralela que nos permite realizar y probar ciertos cambios a partir de un punto dado sin alterar la línea de trabajo original desde la que fue creada.

Ramas

Una *rama* es una línea de trabajo paralela que nos permite realizar y probar ciertos cambios a partir de un punto dado sin alterar la línea de trabajo original desde la que fue creada.

Una vez que los cambios realizados en la *rama* han sido comprobados se pueden integrar en la rama original, resolviendo los “conflictos” que pudiesen haber surgido.

Ramas

Una *rama* es una línea de trabajo paralela que nos permite realizar y probar ciertos cambios a partir de un punto dado sin alterar la línea de trabajo original desde la que fue creada.

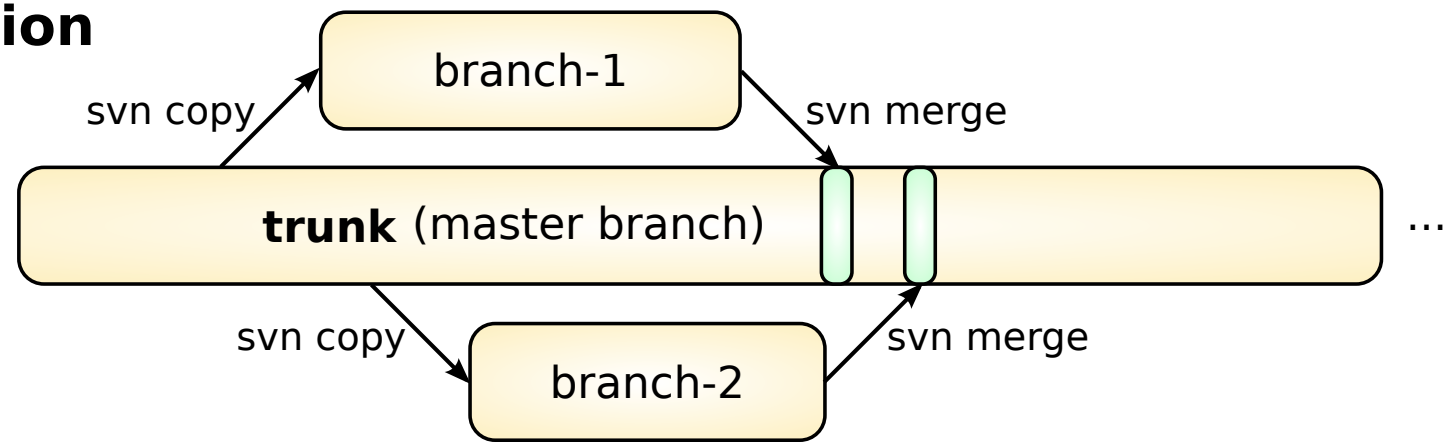
Una vez que los cambios realizados en la *rama* han sido comprobados se pueden integrar en la rama original, resolviendo los "conflictos" que pudiesen haber surgido.

Implementación:

- **En otros *(D)VCS's***: Directorios separados
- **En *git*** : Un *alias* al commit que será el *HEAD* de esa rama (coste casi cero, muy rápido).

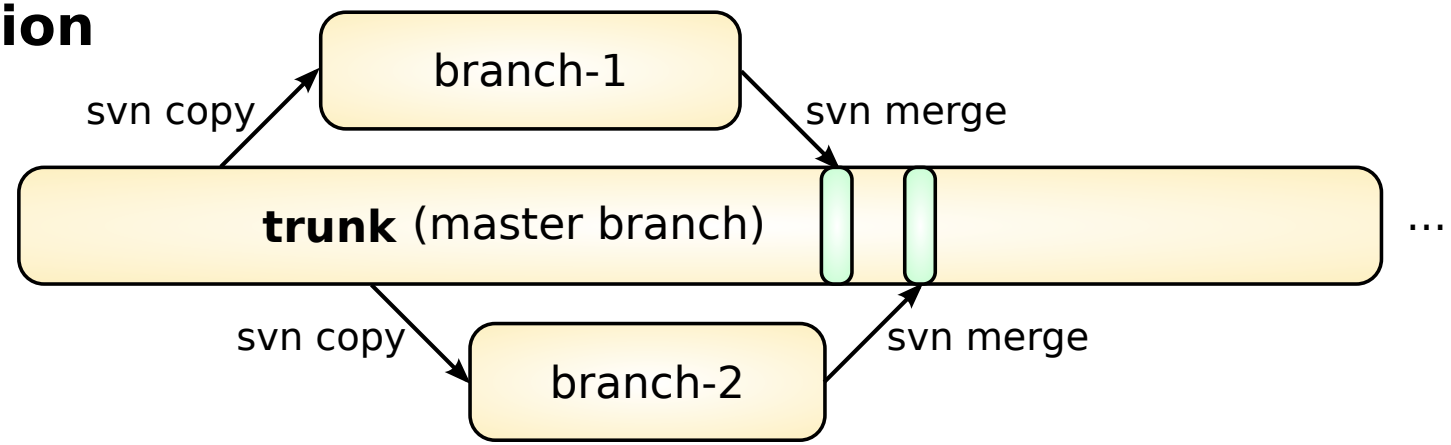
Ramas - Subversion vs Git

Subversion

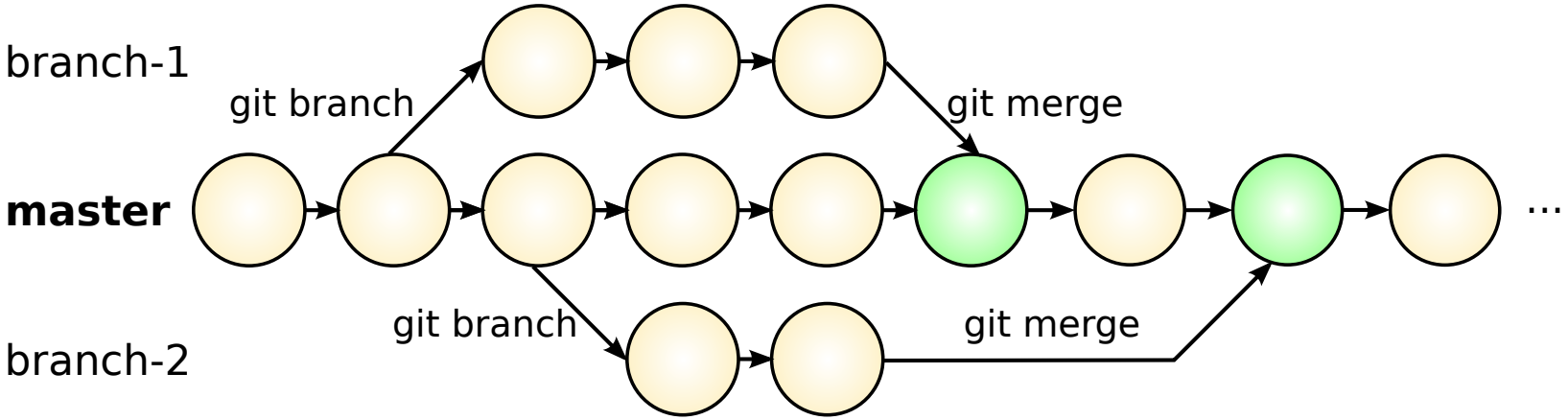


Ramas - Subversion vs Git

Subversion



Git



Ramas en git

Comandos habituales:

- Crear una rama:
  `git branch <branch-name> [<start-point>]`
- Borrar una rama:
  `git branch (-d | -D) <branch-name>`
- Renombrar una rama:
  `git branch -m <old-name> <new-name>`
- Listar las ramas actuales:
  `git branch [ -r | -a ]`
- Situarse en una rama:
  `git checkout [-b] <branch-name>`

Ramas en git - Un ejemplo concreto

```
$ git branch                    # List local branches
* master

$ git branch mybranch-1        # Create new branch

$ git branch mybranch-2        # Create new branch

$ git branch                    # List branches
* master
  mybranch-1
  mybranch-2

$ git checkout mybranch-1  # Change to other branch
Switched to branch 'mybranch-1'

$ git branch                    # List branches
  master
* mybranch-1
  mybranch-2
```

Ramas en git - Un ejemplo concreto

```
$ git branch -m mybranch-1 branch-1  # Rename branch
$ git branch -m mybranch-2 branch-2  # Rename branch

$ git branch                         # List branches
  master
* branch-1
  branch-2

$ git checkout master                # Change to branch
Switched to branch 'master'

$ git branch -d branch-1             # Delete branch
Deleted branch branch-1 (was 59f1309).

$ git branch -d branch-2             # Delete branch
Deleted branch branch-2 (was e721bce).

$ git branch                         # List branches
* master
```

¿Por qué usar ramas?

Algunos motivos:

- Son rápidas, cómodas y muy "baratas"
- Para probar una idea (experimentación)
- Separar trabajo público de privado
- Una rama por *bug/feature*
- Preparar una nueva *release* (integración)
- Como *backup* temporal (*snapshot*)
- Es la forma habitual de trabajar en *git*.

Ramas locales y ramas remotas

Dos tipos de ramas:

- **Ramas locales**: Ramas creadas localmente para trabajar en ellas realizando los cambios que sean necesarios.
- **Ramas remotas**: Ramas de un repositorio remoto que podemos asociar a ramas locales para trabajar con ellas.

Aclaraciones:
Tanto las ramas **locales** como las **remotas** están siempre en nuestro **repositorio local**

Las ramas **remotas** actúan como una *cache* de ramas que son locales en **repositorios remotos**.

Las ramas **locales** se envían/actualizan (*push*/*pull*) a/desde los **repositorios remotos** a través de sus ramas **remotas** asociadas.

Ramas locales y ramas remotas: epiphany browser

Ramas locales

```
$ git branch
  136292
  539716
  602547
  608980
* 611400
  alex
  master
```

Uso de ramas en este ejemplo:

- Una rama por *bug*
- Una rama experimental (`alex`)
- Una rama "principal" (`master`).

Ramas remotas

```
$ git remote
origin

$ git branch -r
  origin/HEAD -> origin/master
  origin/Release043
  origin/bookmarks-toolbars-changes
  origin/bookmarksbar-separation
  origin/css-fonts-branch
  origin/eog-menu-api
  origin/gnome-2-10
  [...]
  origin/master
  origin/mozilla-embed-strings
  origin/new-completion
  origin/new-downloader
  origin/pre-gnome-2-10
  origin/pre-gnome-2-14
  origin/pre-gnome-2-6
  origin/pre-gnome-2-8
  origin/release-1-2-1
  origin/webcore-branch
  origin/webkit
  origin/xulrunner
```

Ramas locales y remotas - Un ejemplo concreto

Listar ramas remotas y *trackearlas* localmente

```
$ git branch -a        # List ALL branches (local & remot
* master
  remotes/origin/master
  remotes/origin/other-branch

$ git branch private  # Create new private branch

$ git branch other-branch origin/other-branch
Branch other-branch set up to track
remote branch other-branch from origin.

$ git branch -a        # List ALL branches
* master
  private
  other-branch
  remotes/origin/master
  remotes/origin/other-branch
```

Ramas locales y remotas - Un ejemplo concreto

Enviar cambios hacia un repositorio remoto

```
<<< MAKE SOME CHANGES IN THE WORKING TREE >>>

$ git commit -a -m "Last commit"
[master 70f7c77] Last commit
 1 files changed, 1 insertions(+), 0 deletions(-)

$ git push
Counting objects: 4, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 272 bytes, done.
Total 3 (delta 1), reused 0 (delta 0)
To ssh://foo@bar.baz/var/git/myrepo.git
   855a39b..70f7c77  master -> master
```

Ramas locales y remotas - Un ejemplo concreto

Traer cambios desde un repositorio remoto

```
<<< WORKING TREE IS CLEAN (No changes) >>>

$ git pull
Updating 43f7dbc..70f7c77
Fast-forward
 body.tex                    |   10 +++++++++-
 closing.tex                 |    8 ++++++++
 cover.tex                   |    8 ++++++++
 opening.tex                 |   41 ++++++++++++-------------------
 toc.tex                     |    8 ++++++++
 7 files changed, 45 insertions(+), 30 deletions(-)
```

Actualizando las ramas

Dos situaciones típicas:

- **Añadir** a una **rama pública** cambios presentes en una **rama privada** (integrar cambios locales)

- **Modificar** una **rama privada** con código nuevo presente en **rama pública** desde la que fue creada (actualizar)

Actualizando las ramas

Dos situaciones típicas:

- **Añadir** a una **rama pública** cambios presentes en una **rama privada** (integrar cambios locales)
  - **No** podemos alterar la historia pasada de la rama pública
  - Necesitamos cambios "limpios" listos para ser aplicados a partir del *HEAD* de la rama pública
- **Modificar** una **rama privada** con código nuevo presente en **rama pública** desde la que fue creada (actualizar)

Actualizando las ramas

Dos situaciones típicas:

- **Añadir** a una **rama pública** cambios presentes en una **rama privada** (integrar cambios locales)
  - **No** podemos alterar la historia pasada de la rama pública
  - Necesitamos cambios "limpios" listos para ser aplicados a partir del *HEAD* de la rama pública
- **Modificar** una **rama privada** con código nuevo presente en **rama pública** desde la que fue creada (actualizar)
  - **Sí** podemos alterar la historia pasada de la rama privada
  - Resulta más interesante "cambiar el pasado" siempre que sea necesario para tener siempre "cambios limpios" desde la rama desde la cual fue creada (pudo haber cambiado)

Actualizando las ramas - git merge

Actualizar una **rama pública**: `git merge <branch>`

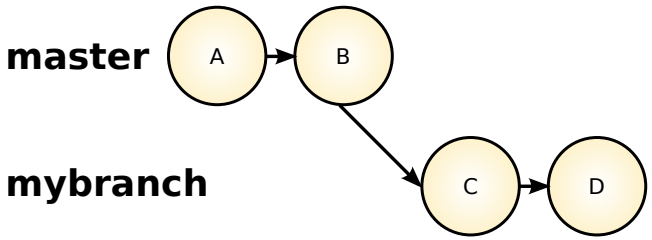
- **No** hay **cambios** en la rama pública desde que se creo la privada:

- **Hay cambios** en la rama pública desde que se creo la privada:

Actualizando las ramas - git merge

Actualizar una **rama pública**: `git merge <branch>`

- **No** hay **cambios** en la rama pública desde que se creo la privada:

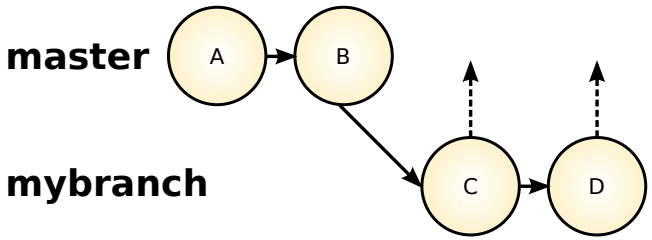


- **Hay cambios** en la rama pública desde que se creo la privada:

Actualizando las ramas - git merge

Actualizar una **rama pública**: `git merge <branch>`

- **No** hay **cambios** en la rama pública desde que se creo la privada:

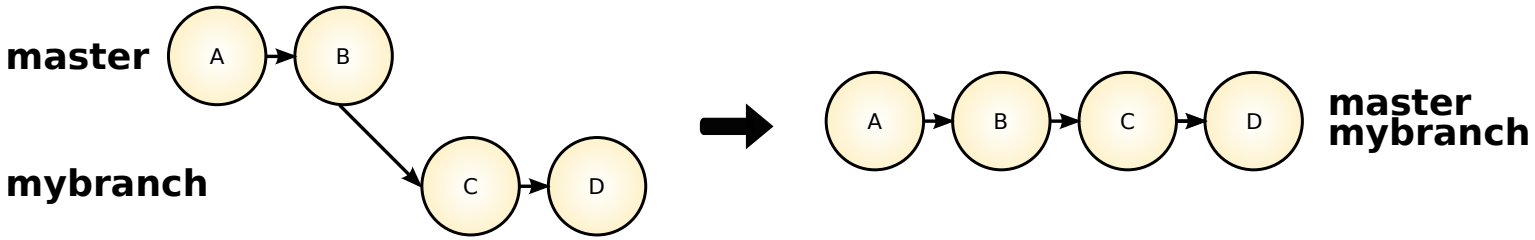


- **Hay cambios** en la rama pública desde que se creo la privada:

Actualizando las ramas - git merge

Actualizar una **rama pública**: `git merge <branch>`

- **No** hay **cambios** en la rama pública desde que se creo la privada:

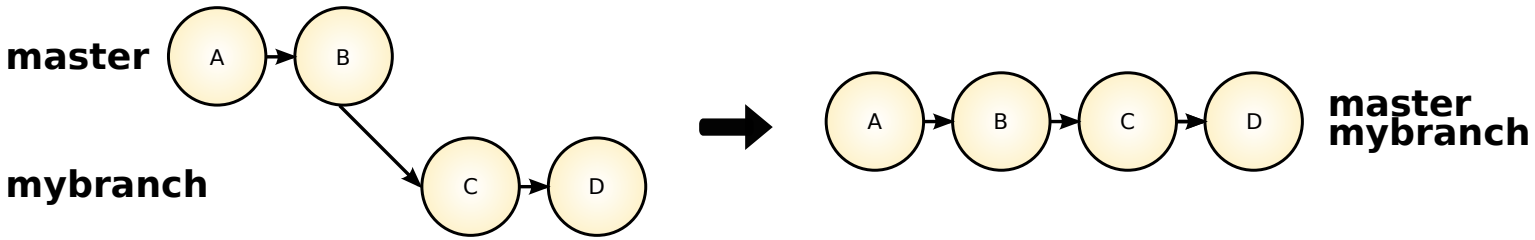


- **Hay cambios** en la rama pública desde que se creo la privada:

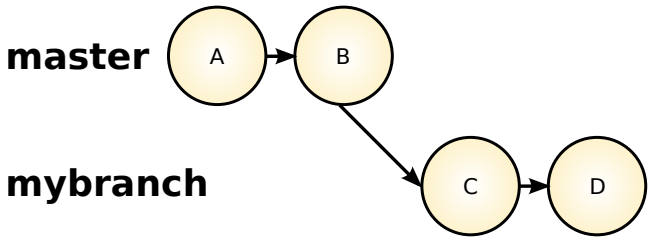
Actualizando las ramas - git merge

Actualizar una **rama pública**: `git merge <branch>`

- **No** hay **cambios** en la rama pública desde que se creo la privada:



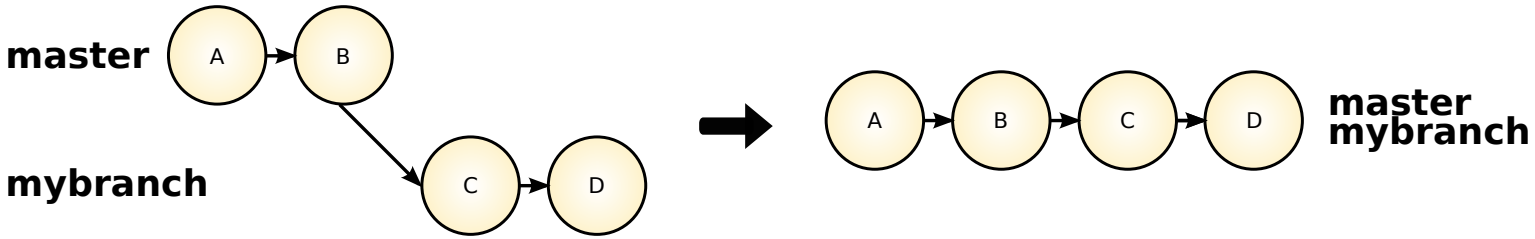
- **Hay cambios** en la rama pública desde que se creo la privada:



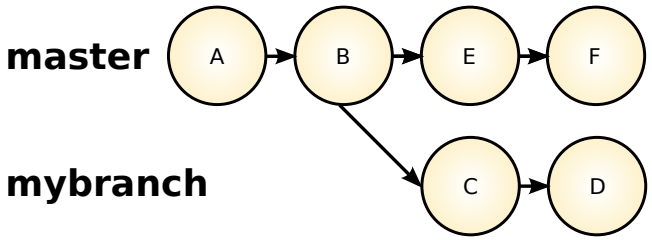
Actualizando las ramas - git merge

Actualizar una **rama pública**: `git merge <branch>`

- **No** hay **cambios** en la rama pública desde que se creo la privada:



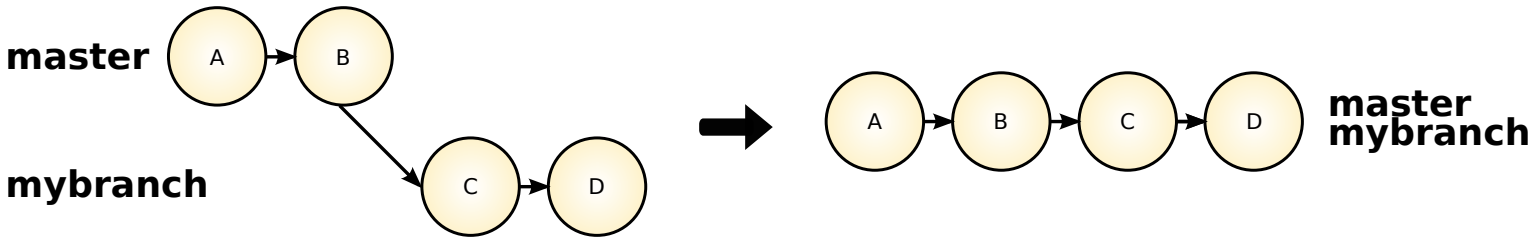
- **Hay cambios** en la rama pública desde que se creo la privada:



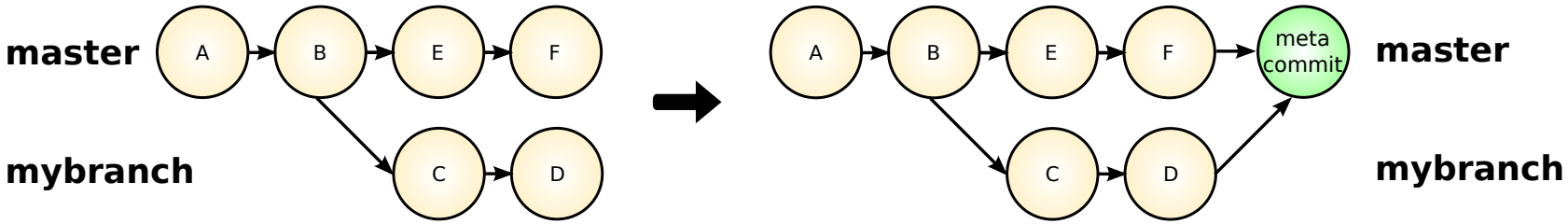
Actualizando las ramas - git merge

Actualizar una **rama pública**: `git merge <branch>`

- **No** hay **cambios** en la rama pública desde que se creo la privada:

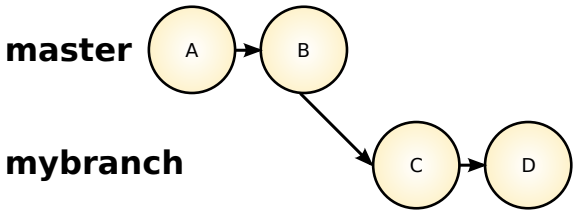


- **Hay cambios** en la rama pública desde que se creo la privada:



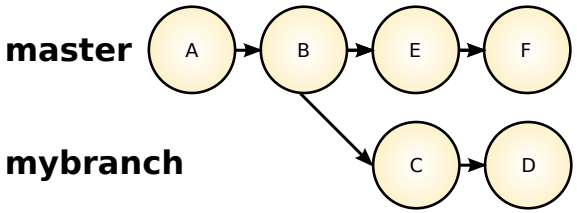
Actualizando las ramas - git rebase

Actualizar una **rama privada**: `git rebase <new-base>`



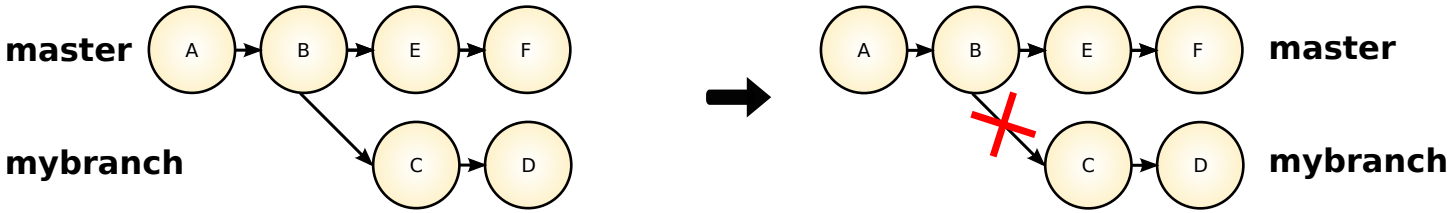
Actualizando las ramas - git rebase

Actualizar una **rama privada**: `git rebase <new-base>`



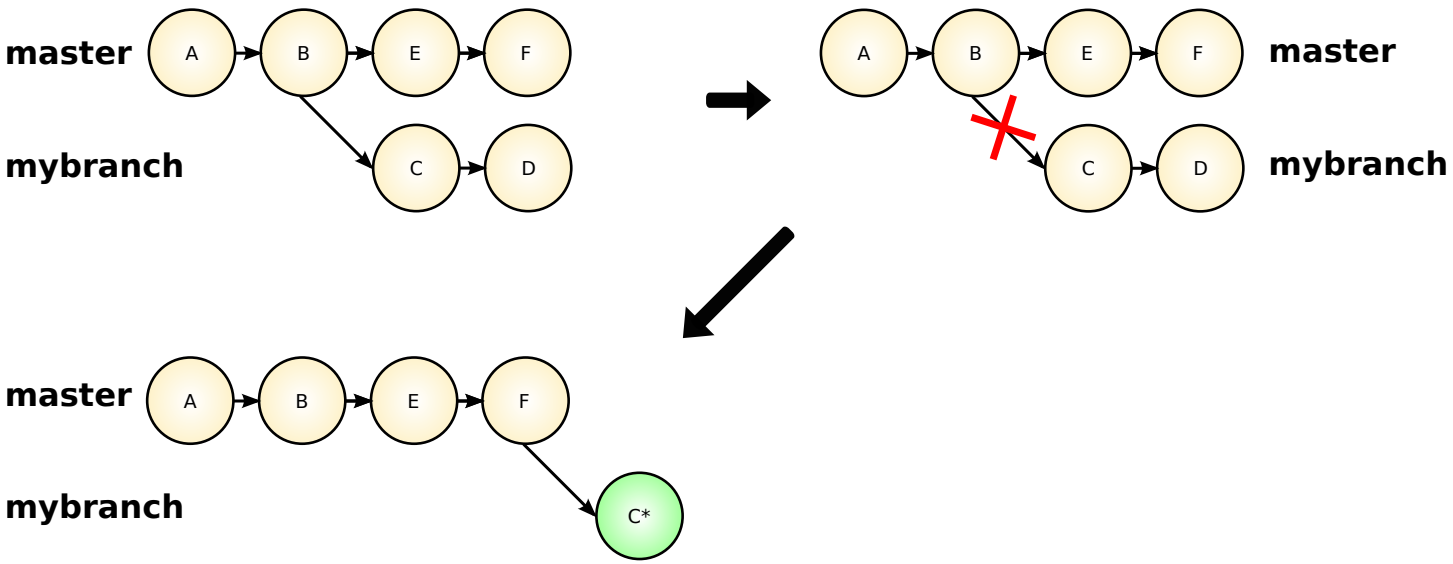
Actualizando las ramas - git rebase

Actualizar una **rama privada**: `git rebase <new-base>`



Actualizando las ramas - git rebase

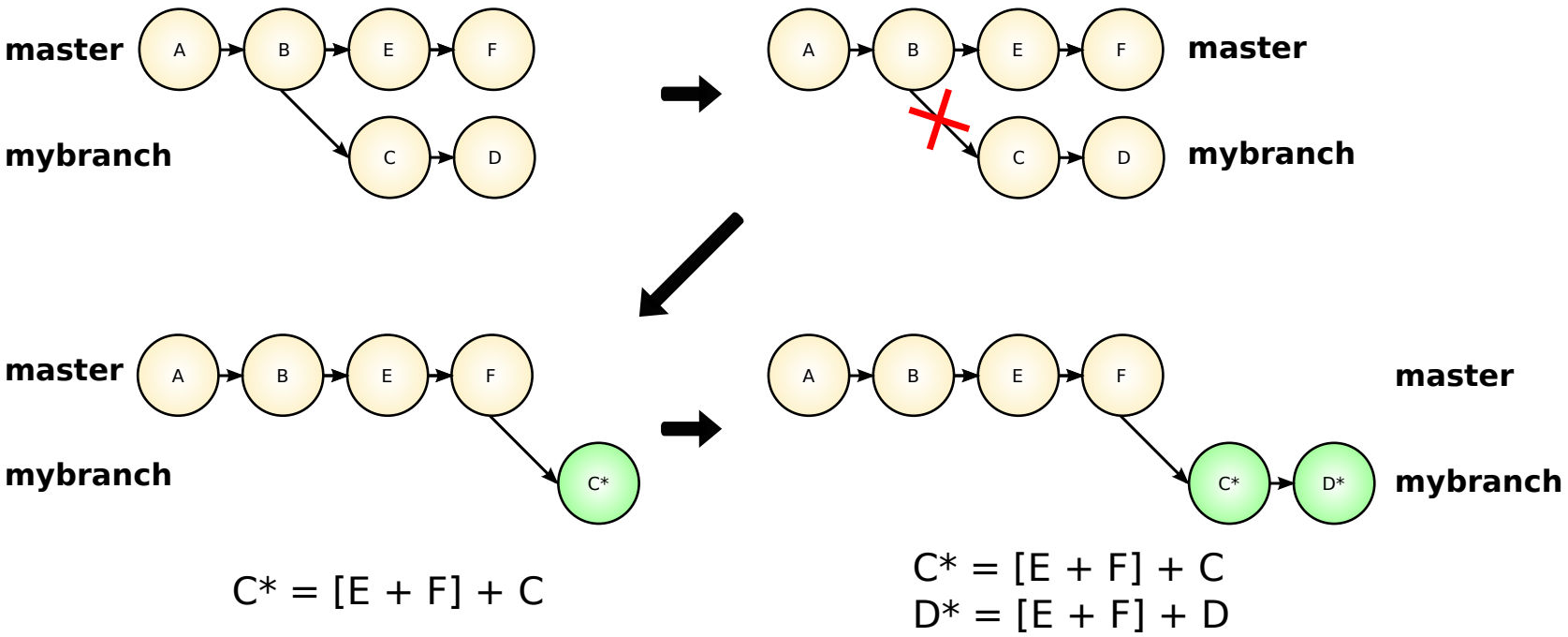
Actualizar una **rama privada**: `git rebase <new-base>`



C* = [E + F] + C

Actualizando las ramas - git rebase

Actualizar una **rama privada**: `git rebase <new-base>`



Consejo (siempre que sea posible)

Antes de actualizar una rama pública con `git merge` para luego hacer un `git push`, hacer primero un `git rebase` desde la rama privada, para evitar la aparición del *metacommit*.

```
$ git checkout mybranch # Change to private branch

$ git rebase master     # Rebase private branch

$ git checkout master   # Change back to master

$ git merge mybranch    # Merge changes from mybranch

$ git push              # Push to remote repository
```

Etiquetas y referencias

Etiquetas

Simplemente un *alias* para un *commit* concreto

- Crear una etiqueta sobre el *commit* actual (*HEAD*):
  `git tag [-a|-s] <tag-name>`
- Borrar una etiqueta:
  `git tag -d <tag-name>`
- Listar etiquetas:
  `git tag -l`
- Enviar una etiqueta a un repositorio remoto:
  `git push <repo-name> <tag-name>`
- Enviar todas las etiquetas al repositorio remoto:
  `git push --tags [repo-name]`

Etiquetas

¿Por qué usar etiquetas?

- Marcar *releases* de un proyecto
- Marcar un punto del desarrollo donde se introduce un *fix*
- Es una práctica habitual en la gestión de proyectos

Lightweight tags* VS *Tag objects

- *Lightweight tags* (*alias*): `git tag <tag-name>`
- *Tag objects*:
  - Anotados: `git tag -a <tag-name>`
  - Firmados: `git tag -s <tag-name>`

Etiquetas - Un ejemplo concreto

```
$ git tag -a TESTING_FIX  # Create new TAG
$ git tag -l              # List existing tags
TESTING_FIX

$ git show TESTING_FIX    # Show details for the TAG
tag TESTING_FIX
Tagger: Mario Sanchez Prada <msanchez@igalia.com>
Date:   Sun Mar 27 18:08:58 2011 +0200

Testing fix xyz

commit bdd27c0e37a91c2ede4793d2a56407e7dd787f8b
Author: Mario Sanchez Prada <msanchez@igalia.com>
Date:   Sat Mar 26 13:11:35 2011 +0100

[...]

$ git tag -d TESTING_FIX  # Delete TAG
Deleted tag 'TESTING_FIX' (was a0418a0)
```

Referencias

Referencias a *commits* concretos:

- Commit actual en la rama actual: *HEAD*
- *Hash id* para "un commit cualquiera":
  `3da14d804c3153c433d0435640bcdd06158b123e`
- Una rama: *mybranch*
- Una etiqueta: *mytag*

Referencias relativas

- Navegar hacia atrás en la historia (padre, abuelo...)

```
Commit 1 posición anterior: ref~1 o ref^
Commit 3 posiciones anterior: ref~3 o ref^^^
```

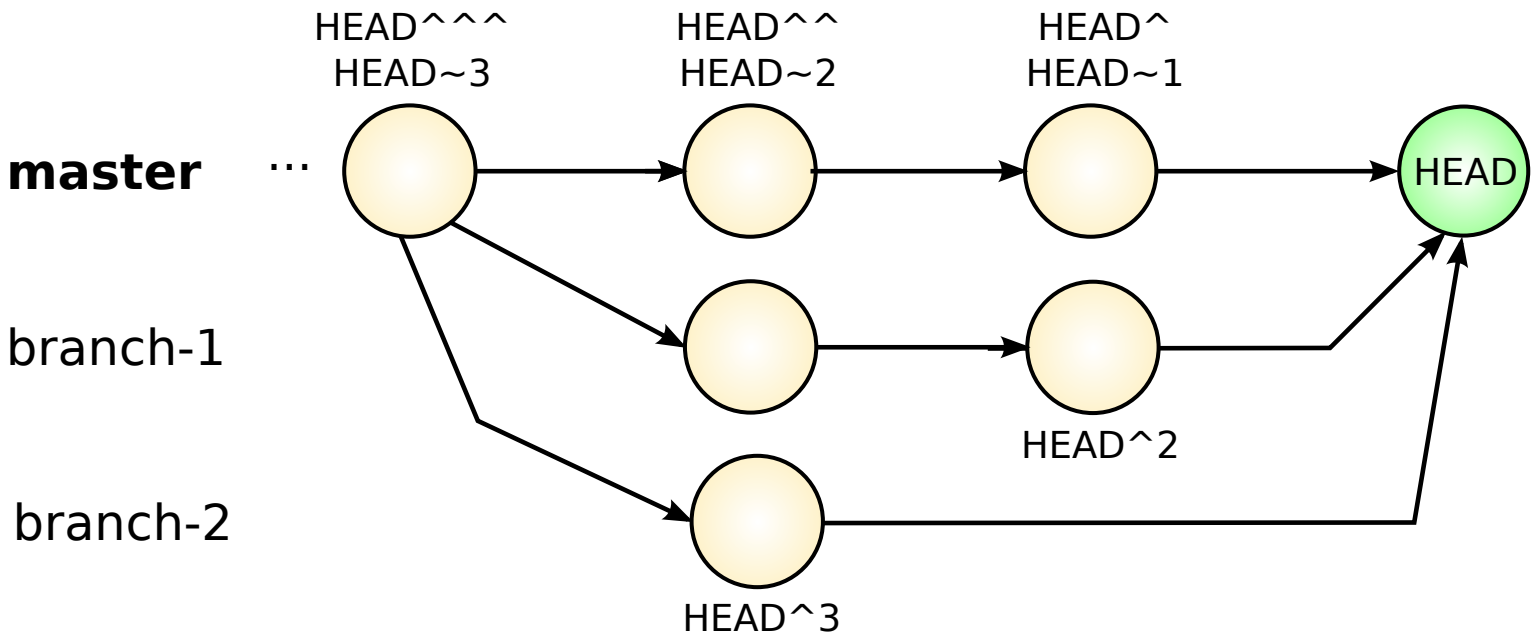
- Navegar por la lista de padres (interesante en *merges*)

```
 Primer padre de la referencia: ref^
Segundo padre de la referencia: ref^2
Enésimo padre de la referencia: ref^N
```

Referencias

ATENCIÓN! WARNING! ACHTUNG! ATTENZIONE!

```
REF~1 == REF^  (1er antepasado == padre)
REF~2 == REF^^ (2° antepasado == padre del padre)
REF~2 != REF^2 (2° antepasado != 2° padre)
```



Otros comandos

Otros comandos

Gestión del *Working Tree y el Index*

- Mostrar estado del *Working Tree* y el *Index*:
  `git status`
- Añade todos los cambios al *Index*:
  `git add -u`
- Añadir cambios al *Index* interactivamente (por *hunks*):
  `git add --patch`
- Cambios entre el *Working Tree* y el *Index*:
  `git diff`
- Cambios entre el *Working Tree* y el *HEAD*:
  `git diff HEAD`
- Cambios entre el *Index* y el *HEAD*:
  `git diff --cached`
- Guardar y restaurar datos en el *Working Tree* y el *Index*:
  `git stash` & `git stash apply`

Otros comandos

Gestión de commits

- Mostrar el historial de *commits*:
  `git log [commitID]`
- Mostrar los detalles de un *commit*:
  `git show [commitID]`
- Mostrar los autores de los últimos cambios en un fichero:
  `git blame <filepath>`
- Revertir un *commit* (crea un nuevo *commit* sobre *HEAD*):
  `git revert <commitID>`
- Modificar el *HEAD*, el *Index* y el *Working Tree*:
  `git reset [ --soft | --hard ] <commitID>`

Otros comandos

Cirugía de commits

- Encontrar *commits* en la rama actual que todavía no hayan sido integrados en otra rama de integración:
  `git cherry <integration-branch>`
- Copiar un commit concreto sobre *HEAD*:
  `git cherry-pick <commitID>`
- Reescribir el pasado manualmente):
  - `git rebase -i <commitID>`
  - `git commit --amend`
- Reescribir el pasado de forma automática:
  `git filter-branch <params>`
- Búsqueda dicotómica de *commits*:
  `git bisect <start> | <good> | <bad>`
- Consultar el histórico (y restaurar datos):
  `git reflog`

Otros comandos

Crear, aplicar y enviar parches

- Crear un fichero *diff* desde un punto dado:
  `git diff <base-commitID>`
- Aplicar un fichero *diff* (no crea un *commit*):
  `git apply <diff-file>`
- Crear parches "completos" desde un punto dado:
  `git format-patch <base-commitID>`
- Aplicar parches "completos" (manteniendo autor, log...):
  `git am <patchfile>`
- Envíar parches por correo:
  `git send-email <email-address>`

Donde ir a partir de aquí

- ***Git from the bottom up* (Excelente tutorial)**:
  `http://ftp.newartisans.com/pub/git.from.bottom.up.pdf`
- ***Git User's Manual***:
  `http://www.kernel.org/pub/software/scm/git/docs/v1.7.1/user-manual.html`
- ***Git tutorial manual page* (`man gittutorial`)**:
  `http://www.kernel.org/pub/software/scm/git/docs/v1.7.1/gittutorial.html`
- ***Everyday GIT With 20 Commands Or So***:
  `http://www.kernel.org/pub/software/scm/git/docs/v1.7.1/everyday.html`
- ***Gitorious: infrastructure for open source projects with Git***:
  `http://gitorious.org`

¿Preguntas?

¡Gracias!