GIT – Guía rapida

Notas:

- *Cambios locales* se refiere a los ficheros del directorio de trabajo (*working dir*) que hayan sido modificados desde el último commit.
- *COMMIT* es un indicador cualquiera en el repositorio: SHA1 de un commit, tag, HEAD (ultimo commit de la rama actual), HEAD~1 (antecesor de HEAD)… La mayoría de las veces, si no se indica se asume HEAD (último commit de la rama actual).

Enlaces:

- http://git-scm.org
- http://www-cs-students.stanford.edu/~blynn/gitmagic/intl/es/ch02.html

Inicializar repositorio

```
$ git init
$ git add .
$ git commit -m "Estado inicial"
```

Crear ramas

```
$ git branch <nombre>                # Crear la rama en el punto actual. Es necesario hacer checkout a la misma.
$ git branch <nombre> <COMMIT>       # Crea la rama a partir del commit dado. Es necesario hacer checkout.
$ git checkout -b <nombre>           # Crear rama en el punto actual y hacerle checkout.
$ git checkout -b <nombre> <COMMIT>  # Crear la rama a partir del commit dado y hacerle checkout.
$ git branch -m <actual> <nuevo>     # Renombrar la rama.
```

Al crear una rama nueva y hacerle checkout los cambios locales se trasladan a esa rama, con lo que el siguiente commit será sobre la rama nueva.

Moverse a una rama o a un commit específico

```
$ git checkout <COMMIT>           # No toca los cambios locales
$ git checkout -f <COMMIT>        # Sobreescribe los cambios locales
```

Se puede hacer una rama desde el commit actual para continuar el desarrollo (git branch). Si se hacen cambios en un commit intermedio (no es un *HEAD*) y se commitean sin hacer una rama, se crea un commit separado que sale del actual (rama sin nombre).

Fusionar ramas (merge)

```
$ git merge <nombre>              # Fusiona la rama indicada en la rama actual
```

Las diferencias se resuelven automáticamente si es posible. En caso de conflictos (código o ficheros binarios modificados en ambas ramas) el proceso se detiene (*merging*) a la espera de una resolución manual.

Resolver conflictos de fusionado:

Dentro de cada fichero en conflicto se añaden marcas alrededor del código conflictivo, mostrando al mismo tiempo la versión de una y otra rama (excepto en ficheros binarios).

```
$ git status                      # Muestra la situación actual del merge (Unmerged paths)
$ git diff                        # Muestra los ficheros conflictivos y las diferencias
$ git add <file>                  # Marca el fichero como corregido una vez resuelto el conflicto
$ git rm <file>                   # Marca el fichero como eliminado en la revisión resuelta
```

La sección “Unmerged paths” de *git status* muestra los ficheros que requieren atención. Debe resolverse cada conflicto manualmente dentro del fichero (eliminando las marcas agregadas por git) y marcarlo como resuelto con *git add*.

En vez de editar los ficheros es posible escoger una de las dos versiones disponibles (rama actual o rama que se está fusionando):

```
$ git checkout --ours -- <file>    # Obtener la versión del fichero en la rama actual
$ git checkout --theirs -- <file>  # Obtener la versión del fichero en la rama que se está fusionando con la actual
```

Para abortar la acción o anularla una vez realizada:

```
$ git reset --hard HEAD           # Abortar el proceso y volver a la situación anterior al intento de merge
```

```
$ git reset --hard HEAD            # Abortar el proceso y volver a la situación anterior al intento de merge
$ git reset --hard ORIG_HEAD       # Deshacer si ya se había confirmado con git commit
```

Una vez resueltos todos los conflictos se confirma el proceso:

```
$ git commit                       # Confirmar la fusión (merge) una vez resueltos todos los conflictos
```

Resolución gráfica de conflictos:

```
$ git mergetool                    # Inicia la herramienta gráfica de resolución de conflictos
```

La herramienta crea ficheros adicionales por cada fichero en conflicto (backup, base, local, remote) para que la herramienta de resolución pueda mostrarlos al usuario al mismo tiempo y éste establecer la versión final. Estos ficheros deberían borrarse automáticamente tras la edición (en caso de que persistan es necesario borrarlos manualmente).

La resolución básica sólo sirve para ficheros de texto. En ficheros binarios usar *git checkout –ours* o *git checkout –theirs* para escoger una de las dos versiones disponibles.

Configurar una herramienta gráfica para resolver conflictos:
http://www.davesquared.net/2010/03/easier-way-to-set-up-diff-and-merge.html (Windows)
http://gitguru.com/2009/02/22/integrating-git-with-a-visual-merge-tool (Mac)

Deshacer cambios

```
$ git revert <COMMIT>              # La recuperación hace un nuevo commit en la rama actual sin ir "hacia atrás" en la historia
$ git reset --hard                 # Deshace los cambios locales
$ git reset --hard HEAD~1          # Elimina el último commit
```

Recuperar una versión determinada de un fichero o path:

```
$ git reset <COMMIT> -- <path>     # git reset NO sobreescribe cambios locales
$ git reset -p <COMMIT> -- <path>  # Seleccionar interactivamente las partes a restaurar
$ git checkout <COMMIT> -- <path>  # Sobreescribe cambios locales sin preguntar
```

En Windows se puede abrir git-bash directamente en cualquier subcarpeta carpeta del proyecto (boton derecho – git bash here). Entonces para recuperar un fichero o path local:

```
$ git checkout <COMMIT> -- ./<path>
```

Guardar cambios actuales para recuperarlos después

Guarda los cambios desde el último commit. Al recuperarlos, si hay colisiones se hace un *merge*.

```
$ git stash                    # Guarda cambios hechos desde el ultimo commit
$ git stash pop                # Recupera los cambios guardados
$ git stash list               # Lista los estados guardados
$ git stash apply              # Aplica cambios guardados sin borrarlos de la lista
```

Marcar el commit actual (Tag)

```
$ git tag -s <nombre> -m <mensaje>
```

El tag queda firmado usando la firma GPG asociada al autor (ver Creating SSH keys).

El nombre identifica al tag y se usa en los demás comandos (ej. git checkout). Por ejemplo, v2.32.45r1

```
$ git tag                              # Mostrar lista de tags
$ git tag -n                           # Mostrar lista y descripción

$ git tag -d <nombre>                  # Eliminar Tag
$ git tag -a <nombre>                  # Crear Tag no firmado
$ git push --tags                      # Subir Tags al repositorio remoto
$ git push origin :refs/tags/<nombre>  # Eliminar Tag borrado localmente
```

Localizar ficheros con una cadena de texto

```
$ git grep <texto>                     # Mira en todos los ficheros del repositorio
$ git grep <texto> -- <ruta>           # Mira sólo en la ruta o rutas especificadas
                                       # Admite patrones (ej. *.cpp)
```

Trabajo con repositorios remotos

Obtener el repositorio desde otra localización (*fork*):

```
$ git clone <ruta al repositorio>      # Clonar y hacer checkout del HEAD de la rama actual
$ git clone -n <ruta al repositorio    # Clonar pero no hacer checkout
```

No hay que hacer *git init* ni crear directorio (se crea automáticamente a partir de la carpeta actual). La ruta puede se una carpeta local, carpeta en red, URL, o cualquier otra referencia a un repositorio remoto. Si es privado será necesario tener la clave SSH configurada adecuadamente (ejemplo).

Recibir los cambios desde el repositorio original:

```
$ git pull
```

Es equivalente a:

```
$ git fetch                            # Trae los cambios
$ git merge origin                     # Fusionarlos con la versión actual
```

Subir cambios al repositorio:

```
$ git push origin <branch>             # Subir sólo la rama indicada
$ git push --all                       # Subir y actualizar todas las referencias remotas
```

Gestionar Tags en el repositorio:

```
$ git push --tags                      # Subir Tags (no suben de otra forma)
$ git push origin :refs/tags/<nombre>  # Eliminar Tag borrado localmente
```

Borrar una rama remota:

```
$ git push origin :<branch>
$ git push origin --delete <branch>    # GIT versión 1.7.0+
```

Listar los repositorios remotos y sus URLs:

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

Cambiar la URL de un repositorio remoto (cambia ambos fetch y push)

```
$ git remote set-url origin <nueva URL>
```

Revertir un commit en local y también en el repositorio remoto:

(lo lógico es que no hubiera sido subido)

```
$ git reset --hard HEAD~1
$ git push origin +master:master
```

"The +master:master thing is necessary to tell git that you really do want to rewind the history here, (it's definitely not part of the normal flow)."

Tutorial
Push and delete branches

Tareas especiales

Añadir un nuevo fichero o patrón a .gitignore

Añadirlo a ***.gitignore*** sigue controlando aquellos ficheros que ya están en el repositorio, y

```
git rm <file>
```

eliminaría el fichero del directorio de trabajo.

Para dejar de controlar el fichero o patrón manteniendo las copias actuales añadirlo a .gitignore y entonces:

```
$ git rm --cached <file>
$ git rm --cached -r <pattern>        # Eliminar ocurrencias en todo el arbol
```

Borrar el fichero completamente del repositorio implica reescribir toda la historia. Nada recomendable.

Localizar el cambio que originó un problema:

```
$ git bisect start
$ git bisect bad                      # La versión actual va mal
$ git bisect good v2.6.13-rc2         # Esta versión es buena
```