ÍNDICE

1. Introducción

En un proyecto de trabajo colaborativo, cada miembro del equipo realiza las tareas que le han sido asignadas, pero al terminar cada uno de ellos, habrá que preguntarse:

- ¿Cómo se integran todas las partes?
- Si hay errores o cambios, ¿cómo se actualizan los nuevos cambios?
- Si hay una nueva versión, ¿cómo se gestionan los conjuntos de módulos compatibles con cierta versión?, ¿cómo se recuperan versiones anteriores?, ¿cómo se fusionan versiones? ...

En el documento Control de versiones se hace referencia a diferentes herramientas de control de versiones, aquí se tratará de mostrar como una de ellas (GIT) da respuesta a las preguntas anteriores.

Algunas de las características de GIT son:

- Gratis, de código abierto.
- Muy popular, disponible en múltiples plataformas.
- Distribuido. Las diferentes réplicas de los repositorios tienen una relación de igual a igual. Esta es una consideración técnica, en la práctica una de ellas suele adquirir el rol de repositorio principal.
- Basado en changesets. Si varios componentes del proyecto han cambiado respecto a la última versión, todos ellos son volcados al repositorio de una vez, esta entrega se considera atómica.
- No bloqueante. Es posible que varios repositorios locales estén trabajando de forma simultánea sobre los mismos componentes, posteriormente habrá que integrar las modificaciones efectuadas en todos ellos para obtener un versionado común.

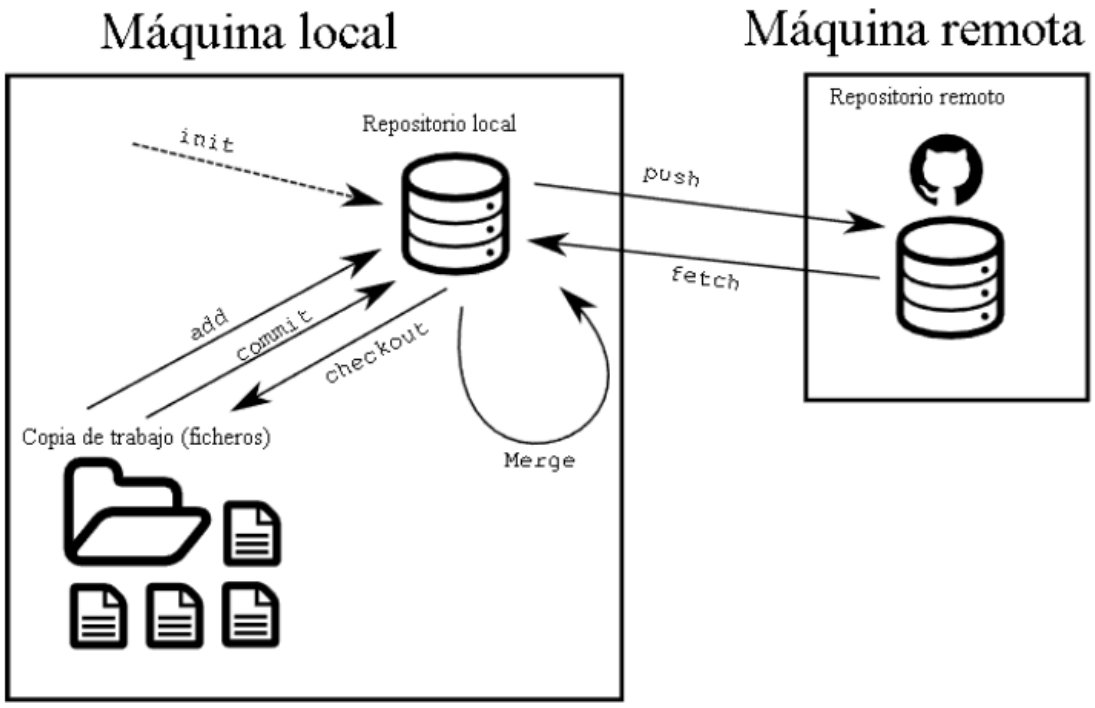
2. Tipos de herramientas de control de versiones.

Algunos de los términos de uso habitual en GIT son:

- **Repositorio**: base de datos con las sucesivas versiones. Un repositorio podrá estar compuesta de diversas líneas de desarrollo o ramas. Típicamente habrá un repositorio común y tantos repositorios locales como participantes en la actividad colaborativa.
- **Rama master o trunk**: rama principal del desarrollo.
- **Rama (Branch)**: línea de desarrollo paralela a la rama master. Podría recoger trabajos para diferentes clientes.
- **Área de trabajo**: ficheros sobre los que trabaja el programador.
- **Commit**: confirmación de una nueva versión (de la copia de trabajo al repositorio local).
- **Checkout**: (re)generación de la copia de trabajo a partir de la información del repositorio local.
- **Fusión (Merge)**: acumular cambios en una misma versión. Permite combinar diferentes branchs en el repositorio local.
- **Comentario**: cada commit deberá documentar sus efectos y motivaciones.
- **Push**: actualización del repositorio remoto con la información del repositorio local.
- **Fetch**: actualización del repositorio local con la información del repositorio remoto.
- **Pull**: actualización del repositorio local y del área de trabajo con la información del repositorio remoto.

3. Estructura de las herramientas de control de versiones.

En un proyecto colaborativo coordinado con GIT, un escenario típico de trabajo sería el siguiente.



En la figura se puede ver una máquina remota donde se encuentra el repositorio principal o remoto, sitio donde se pone en común todo el trabajo del conjunto de integrantes del equipo. Cada desarrollador tendrá su propio espacio de trabajo, en la figura se muestra una única máquina local.

Inicialmente, el entorno local tendrá un repositorio que será una réplica idéntica del repositorio remoto. Además, el programador dispondrá de su área de trabajo, donde irá actualizando el código. Cuando unos cambios cobran suficiente entidad como para constituir una versión, estos son guardados en el repositorio local. Finalmente, si el programador desea poner en común con los otros miembros del proyecto sus cambios, enviará sus actualizaciones desde el repositorio local al común.

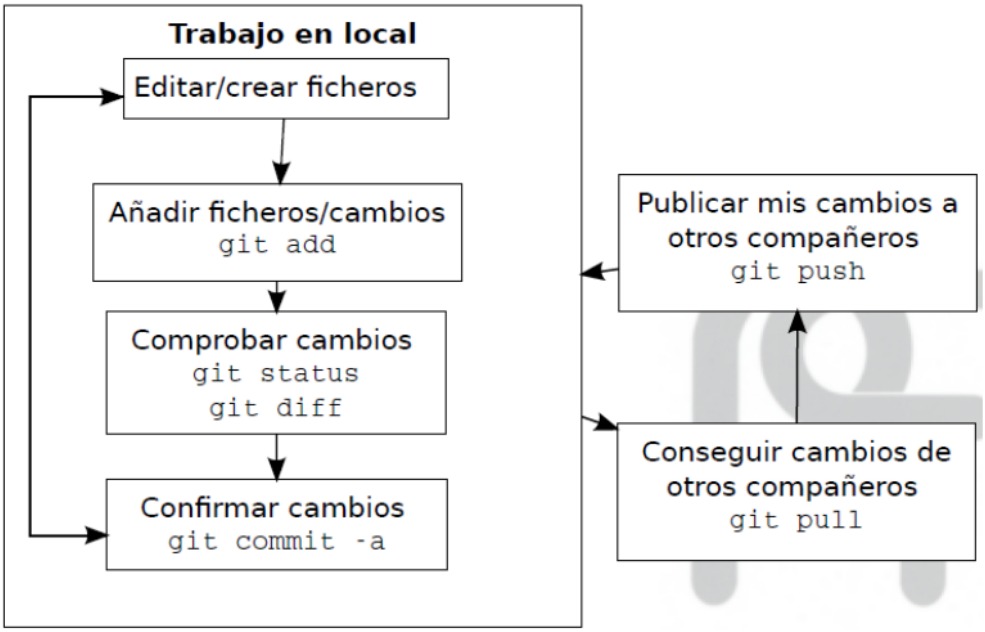
4. Flujo de trabajo.

El flujo de trabajo normal del programador consistirá en las siguientes actuaciones:

- **Modificar el área de trabajo**. Actualizando el contenido de los ficheros del proyecto o creando/eliminando algunos de ellos.
- **Seleccionar** aquellos **ficheros** que se desea formen parte de la nueva versión del proyecto. Los ficheros que no sean seleccionados, aunque hayan sido cambiados no serán guardados en el repositorio.
- **Comprobar cambios**. Habrá que cerciorarse que los cambios introducidos en el proyecto son los deseados.
- **Salvar las modificaciones** introducidas en el área de trabajo al repositorio local.

Los cuatro puntos anteriores se podrán repetir tantas veces como sea necesario. A continuación, habrá que:

- **Conseguir los cambios de otros compañeros**. Es muy probable que en el repositorio común existan actualizaciones que haya introducido algún otro colaborador, por lo que se hace necesario combinar esos cambios con los que queremos compartir nosotros.
- **Actualizar el repositorio común desde el local** para compartir nuestros cambios.



5. Resumen de comandos GIT.

A continuación se muestran algunos de los comandos más utilizados en GIT.

- **init:** Crea un nuevo repositorio en el directorio inicialmente vacío. Se almacena en el directorio oculto .git.

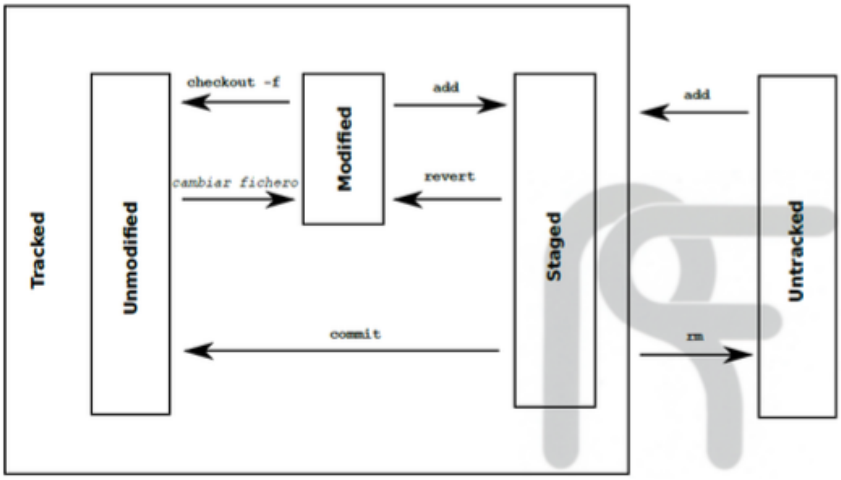
```
alvaro@debian8-64-alvarogonzalez:~/aplicacion-web$ git init
Initialized empty Git repository in /home/alvaro/aplicacion-web/.git/
```

- **clone:** Crea un nuevo repositorio, copia de uno ya existente. El repositorio replicado se indica mediante su URL. La copia de trabajo inicial será la de la rama MASTER.

```
alvaro@debian8-64-alvarogonzalez:~/aplicacion-web$ git clone \
     https://github.com/alvarogonzalezsotillo/aplicacion-php.git
Cloning into 'aplicacion-php'...
remote: Counting objects: 50, done.
remote: Compressing objects: 100% (35/35), done.
remote: Total 50 (delta 21), reused 39 (delta 12), pack-reused 0
Unpacking objects: 100% (50/50), done.
Checking connectivity... done.
```

- **add, rm, mv:** Permite identificar los cheros del área de trabajo que van a ser incluidos en el repositorio. Para seleccionar cheros se usa el comando add, para deseleccionarlos el comando a utilizar será rm. El borrado de cheros no tiene carácter retroactivo sobre las versiones anteriores.

- **status:** Imprime un resumen del estado de los cheros de la copia de trabajo. Los ficheros se pueden encontrar en los siguientes estados:
  - Untracked: no guardados en el repositorio. GIT ignora el fichero.
  - Unmodified: igual que en la rama activa del repositorio.
  - Modified: con cambios respecto al repositorio, pero que no está previsto que sean guardados al hacer *commit*.
  - Staged: el fichero está en el index. Será parte del próximo changeset que se añadirá al repositorio.



- **commit:** Crea una nueva versión en el repositorio local, incluyendo los cambios en estado *Staged* procedentes del área de trabajo.

  Los ficheros pasan a estar seleccionados como parte de las nuevas versiones con el comando add. El comando *git commit -a* incluye en el index todos los ficheros del área de trabajo diferentes a los disponibles en el repositorio local, a continuación, aplica un *commit* creando una nueva versión.

```
alvaro@alvaro-vaio:~/aplicacion-web$ git commit -a
[master 906fa20] Cambiados varios permisos de ejecución
 5 files changed, 411 insertions(+), 2 deletions(-)
 create mode 100644 09/estados-fichero-git.svg
 mode change 100755 => 100644 borrar-temporales-latex.sh
 mode change 100755 => 100644 generar-pdf.sh
 mode change 100755 => 100644 generar-pdfs.sh
 mode change 100755 => 100644 publicar-pdfs.sh
```

- **push:** Sube las versiones del repositorio local a un repositorio remoto. El repositorio remoto puede establecerse con:
  - git clone (en este caso se denomina origin).
  - git remote add

  El repositorio debe tener todas las versiones del repositorio remoto, en otro caso, deberá realizarse un git pull previo.

```
alvaro@debian8-64-alvarogonzalez:~/aplicacion-web$ git push
Password for 'https://alvarogonzalezsotillo@bitbucket.org':
To https://alvarogonzalezsotillo@bitbucket.org/alvarogonzalezsotillo/aplicacion-web.git
! [rejected]        master -> master (fetch first)
error: failed to push some refs to
'https://alvarogonzalezsotillo@bitbucket.org/alvarogonzalezsotillo/aplicacion-web.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

Observar en la figura que el comando *push* ha dado error, el motivo es que el repositorio remoto tiene recogidos cambios de otros colaboradores que no han sido volcados al repositorio local. Si no se combinan ambos, podría producirse pérdida de información, por lo que sugiere hacer un *pull* previo al *push*.

- **checkout:** extrae versiones del repositorio local a la copia de trabajo, la acción se puede realizar sobre diferentes elementos del repositorio:
  - Un fichero: git checkout fichero.
  - Una versión: git checkout hashdeversión.
  - Un branch: git checkout branch.

  También permite crear nuevas ramas, con *git checkout -b nombrerama*.

- **branch:** lista, crea o elimina ramas en un repositorio.

Otros comandos de interés son:

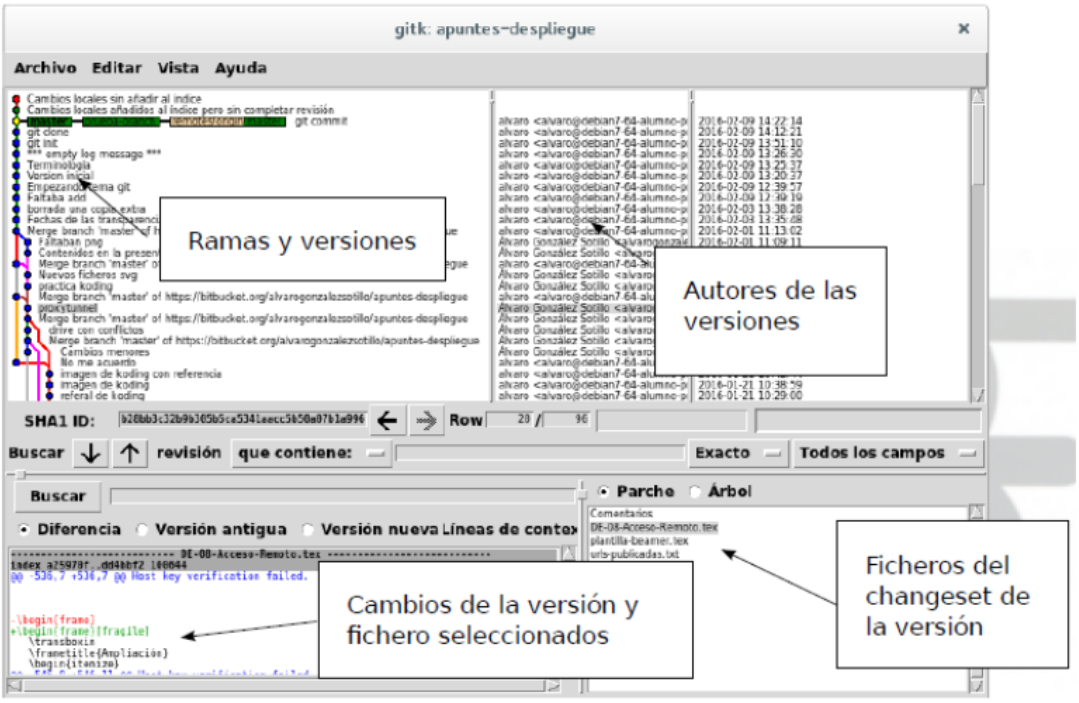
- **git log**: historia de cambios de un repositorio.
- **git diff**: muestra los cambios de los ficheros en estado modified.
- **git merge**: fusión de dos ramas.
- **git gui**: interfaz gráfica para utilizar GIT.

6. Gitk.

Gitk es un visor del repositorio local. Permite revisar ramas, commits y merges entre otros.

Es una aplicación útil para:

- Recuperar versiones antiguas de un fichero.
- Visualizar las ramas de un repositorio.
- Hacer un seguimiento de la actividad en el repositorio.


gitk: apuntes-despliegue
Archivo Editar Vista Ayuda
Ramas y versiones
Autores de las versiones
SHA1 ID:
Row
Buscar
revisión
que contiene:
Exacto
Todos los campos
Parche
Árbol
Diferencia
Versión antigua
Versión nueva
Líneas de contex
Cambios de la versión y fichero seleccionados
Ficheros del changeset de la versión