Un control de versiones es un sistema que registra los cambios realizados en un archivo o conjunto de archivos de código fuente a lo largo del tiempo, de modo que se puedan recuperar versiones específicas más adelante. Este sistema permite regresar a versiones anteriores de los archivos o de un proyecto completo, comparar cambios, ver quién lo modificó, cuándo, etc.

Al ser un sistema distribuido, Git nos permite crear un repositorio que será alojado en un servidor (GitHub, GitLab, Bitbucket, etc). Clonamos el repositorio a nuestra máquina local y una vez ahí podemos hacer todo los cambios (commits) que queramos, sin tener necesidad de conexión.
Cabe aclarar, que no es necesario contar con un servicio de hosting para usar Git. El sólo hecho de tenerlo instalado en nuestra máquina, nos permite crear un repositorio local.

Cómo funciona el control de versiones.

Git maneja sus datos como un conjunto de copias instantáneas. A medida que se van confirmando los cambios y se suben, lo que hace Git es compararlo con el archivo anterior y sólo guarda la diferencia (delta) entre ambos archivos y le adjudica una referencia.

Git tiene tres estados en los que pueden estar los archivos:

1. Commited: significa que los datos están almacenados de manera segura en tu base de datos local.
2. Modified: se modificó el archivo pero todavía no se ha confirmado en la base de datos.
3. Staged: se ha marcado un archivo modificado en su versión actual para que vaya en la próxima confirmación.(commit)

En el directorio .git, se almacenan los metadatos y la base de datos de objetos del proyecto, que es lo que se copia cuando clonamos un repositorio.

El flujo de trabajo básico en Git es algo así:
1. Modificas una serie de archivos en tu directorio de trabajo.
2. Preparas los archivos, añadiéndolos a tu área de preparación.
3.Confirmas los cambios, lo que toma los archivos tal y como están en el área de preparación y almacena esa copia instantánea de manera permanente en tu directorio de Git.
Si una versión concreta de un archivo está en el directorio de Git, se considera confirmada (committed). Si ha sufrido cambios desde que se obtuvo del repositorio, pero ha sido añadida al área de preparación, está preparada (staged). Y si ha sufrido cambios desde que se obtuvo del repositorio, pero no se ha preparado, está modificada (modified).

Clonar un repositorio:

Para clonar un repositorio existente, necesitamos la URL del proyecto. Esta se encuentra en la web del servicio de hosting.
Desde una terminal de linux, sobre el directorio que queramos instalar el proyecto, usamos el siguiente comando:

```
$ git clone <URLdelProyecto>
```

Este comando inicializa un directorio .git, descarga toda la información de ese repositorio y saca una copia de trabajo de la última versión.

Área de preparación:
Una vez que empezamos a hacer modificaciones en el proyecto, debemos mover esas modificaciones al índice para poder realizar el commit posterior. Para ello usamos el comando:

```
$ git add .
```

Este comando añade al índice todos los archivos nuevos o que hayan sido modificados. Los archivos que no hayan sido agregados con git add, no podrán ser confirmados pero se mantendrán modificados en el disco local.
Si queremos agregar un archivo en particular, usamos el siguiente comando:

```
$ git add <nombreDelArchivo>
```

Confirmar Cambios:
Una vez preparados los archivos a confirmar, debemos guardar una "instantánea" del proyecto, que podremos usar para comparar o volver a ella luego. Con el siguiente comando:

```
$ git commit -m "BreveDescripciónDelCommit"
```

A continuación, tenemos que ejecutar
$ git push …….
Git nos mostrará los datos de la confirmación: rama modificada, qué número de checksum tiene el commit, cuántos archivos cambiaron y cantidad de líneas añadidas o eliminadas.

Para ver qué modificaciones hemos hecho a lo largo del tiempo, usamos el comando:

```
$ git log
```

Podemos limitar la info, agregando el número de salida que esperamos. Ejemplo, si queremos ver los últimos 3 commits, usamos:

```
$ git log -3
```

Branch (rama):
Las ramas o 'branches', se usan para poder desarrollar funcionalidades aisladas unas de otras. Cuando se crea un repositorio, existe una rama principal por defecto, que es la master.
Podemos crear nuevas ramas para desarrollar con tranquilidad y fusionarlas a la rama principal, cuando hayamos terminado.
Para crear una rama nueva y pararnos sobre ella, usamos:

```
$ git checkout -b nombreDeLaRama
```

A partir de este momento, todos los cambios que hagamos, se verán reflejados en la rama que hemos creado.
Si queremos cambiar de rama tenemos un comando que es:
$ git checkout <rama>

Llegado el momento, tendremos que fusionar (merge) nuestra rama con la master, para agregar los cambios al proyecto principal. Antes, es necesario pararnos sobre la rama master.

Para movernos a otra rama, usamos:

```
$ git checkout <nombreDeLaRamaQueQueremosIr>
```

en nuestro caso:

```
$ git checkout master
```

Para fusionar ambas ramas, usamos el comando:

```
$ git merge nombreDeLaRamaAFusionar
```

Esta es la versión simple del merge , pero merece un capítulo aparte. Recomendamos la lectura del libro “Git Pro”, el capítulo destinado a merge se encuentra disponible en el siguiente enlace:
https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging

Proyectos remotos:

Para traer los metadatos de proyectos remotos, usamos:

```
$ git fetch nombreDelRemoto
```

Este comando traerá del proyecto los metadatos que no tenemos en nuestra máquina local.
Si queremos traer y combinar los datos automáticamente con nuestro archivo local, debemos usar:

```
$ git pull
```

Cuando queramos subir los cambios a la rama principal:

```
$ git push origin master
```

Para ello, debemos tener permisos de lectura y escritura.

Participación en proyectos.

Si se quiere participar de un proyecto existente del cual no tenemos permiso de escritura (push), debemos hacer un **Fork** (bifurcación).
El fork lo que hace es copiar el repositorio original en el servidor y es esa copia del repositorio la que vamos a clonar a nuestra máquina local. Sobre este repositorio podemos trabajar libremente y luego, remitir esos cambios al repositorio original mediante un Pull Request.
Para crear un fork debemos ir a la web donde está alojado el proyecto en el que queremos trabajar y presionar el botón “Fork”, que nos redireccionará a una página nueva del proyecto en nuestra cuenta y con nuestra propia copia del código fuente.
Si la dirección del proyecto está ubicada en la siguiente dirección:
https://github.com/nombreDelProyecto

Nos creará un fork con nuestro nombre de usuario, de la siguiente manera:
https://github.com/nombreDeUsuario/nombreDelProyecto

Clonamos esta copia localmente, creamos una branch y trabajamos sobre ella.
Una vez que trabajamos sobre ella y realizamos todos los cambios, vamos a nuestra web de GitHub (o el servicio que usemos) y en el apartado “Branches”, localizamos nuestra rama e iniciamos el Pull Request desde ahí. Debemos ponerle un título y una breve descripción, para que el propietario evalúe qué cambios hicimos y decida mergear o no.

Cómo actualizar tu Fork si ocurrieron cambios en el repo original posteriores a la copia

En el caso de que hayas hecho un fork y se realizaron modificaciones posteriores en el repo original, para actualizar tu copia se deben realizar los siguientes pasos:

```
$ git remote add upstream nombreRepoOriginal
```

Se genera la metadata

```
$ git fetch upstream
```

Se valida la versión del branch

```
$ git branch -vv
```

Se valida el log

```
$ git log upstream/master
```

Se realiza el merge, siempre se debe estar parado en la rama original de la que se quieren tomar los cambios

```
$ git merge upstream/master
```

Fuente: clase del día 28-03-2018, dictada por Diego Sánchez y Git Pro (https://git-scm.com/book/en/v2)