UNIVERSIDAD DE EL SALVADOR

FACULTAD MULTIDISCIPLINARIA DE OCCIDENTE

DEPARTAMENTO DE INGENIERIA Y ARQUITECTURA

PROGRA 2.5

Objetivo:

Que el estudiante conozca o amplie sus conocimientos sobre el control de versiones

Requisitos:

- Distribucion de GNU/Linux
- Conexión a internet.
- Ganas de aprender xd

1- ¿Qué es el control de versiones y porque debería importarte?

El control de versiones es un sistema que registra los cambios realizados sobre un archivo o conjunto de archivos a lo largo del tiempo, de modo que puedas recuperar versiones específicas más adelante.

Te permite revertir archivos a un estado anterior, revertir el proyecto entero a un estado anterior, comparar cambios a lo largo del tiempo, ver quién modificó por última vez algo que puede estar causando un problema, quién introdujo un error y cuándo, y mucho más. Usar un VCS también significa que si fastidias o pierdes archivos, puedes recuperarlos fácilmente.

Instalación en Linux

En Linux es muy sencillo de instalar, desde la propia página WEB nos muestran cómo instalarlo en varias distribuciones:

DISTRO	COMANDO PARA SU INSTALACIÓN
Debian/Ubuntu	`$ apt-get install git`
Fedora	`$ yum install git`
Gentoo	`$ emerge --ask --verbose dev-vcs/git`
Arch Linux	`$ pacman -S git`
FreeBSD	`$ cd /usr/ports/devel/git && make install`
Solaris 11 Express	`$ pkg install developer/versioning/git`
OpenBSD	`$ pkg_add git`

Instalación en Windows

Debemos acceder al apartado de descargas: http://git-scm.com/downloads y obtenerr la versión de Windows. Es un sencillo programa de instalación en modo de asistente (el clásico siguiente-siguiente-siguiente).

Yo recomiendo marcar durante la instalación la opción “Git Bash Here” que nos permite a través del botón derecho del ratón abrir un shell (bash de mingw) en la carpeta que indiquemos. Resulta más cómodo.

Primeros pasos:

Lo primero que hemos de hacer es situarnos en la carpeta raíz del proyecto (Ej: /home/user/(NetBeansProjects/ejemplo-git) y crear un repositorio. Si estamos en linux desde el terminal navegamos hasta la carpeta y en Windows con la opción del menú contextual “Git bash here” podemos abrir una consola en esa misma carpeta. Tecleamos los siguientes comandos:

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

¿Qué hemos hecho?

1- Hemos creado el repositorio (git init)
2- Hemos añadido TODOS los ficheros/carpetas al repositorio (git add .). Es decir, le notificamos que los tenga en cuenta para cualquier cambio.
3- Realizamos el primer commit del proyecto obteniendo una imagen de su estado inicial (git commit -m “MENSAJE PARA EL COMMIT”)

Comandos para el uso de GIT

Crear ramas

Cualquier sistema de control de versiones moderno tiene algún mecanismo para soportar distintas ramas. Cuando hablamos de *branching* o trabajo con ramas, significa que tu has tomado la rama principal de desarrollo (`master`) y a partir de ahí has continuado trabajando sin seguir la rama principal de desarrollo.

Una rama es un puntero a un commit específico en el repositorio.

```
$ 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 para cambiar a la rama.
$ 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
$ git branch -d <nombre>             # Borrar 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.

Listar ramas

```
$ git branch                        # Listar todas las ramas
$ git branch -v                     # Mostrar último commit en cada rama y su situación respecto a su rama remota (si hay)
$ git branch --merged               # Mostrar ramas que se han fusionado con la actual, y por tanto pueden borrarse
$ git branch --no-merged            # Mostrar ramas con trabajos sin funsionar. Intentar borrarlas dará un error.
```

Fusionar ramas (merge)

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

Trabajo con repositorios remotos

Para poder colaborar en cualquier proyecto Git, necesitas saber cómo gestionar tus repositorios remotos. Los repositorios remotos son versiones de tu proyecto que se encuentran alojados en Internet o en algún punto de la red. Puedes tener varios, cada uno de los cuales puede ser de sólo lectura, o de lectura/escritura, según los permisos que tengas. Colaborar con otros implica gestionar estos repositorios remotos, y mandar (*push*) y recibir (*pull*) datos de ellos cuando necesites compartir cosas.

Obtener el repositorio desde otra localización

```
$ 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 ser una carpeta local, carpeta en red, URL, o cualquier otra referencia a un repositorio remoto.

Recibir los cambios desde el repositorio original:

```
$ git pull
```

Subir cambios al repositorio:

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

Cambiar la URL de un repositorio remoto

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

ENLACES EXTERNOS

http://librosweb.es/libro/pro_git/