Acerca de Git y el control de versiones

El control de versiones es una práctica esencial para los programadores. Es el acto de llevar un registro de los cambios que se realizan en un conjunto de archivos. Por extensión, el control de versiones es imprescindible para que las organizaciones puedan llevar registro del software que desarrollan.
Para los programadores, desarrollar software no es fácil, a veces hay que probar muchas alternativas hasta conseguir una solución aceptable. Otras veces, una solución compleja se logra después de varios avances incrementales. Las **versiones** son cada una de esas alternativas y cada uno de esos avances. Llevar registro de los avances e intentos es una buena práctica, que se logra mediante el control de versiones.
Para las organizaciones, mantener software que ha sido lanzado al mercado es una tarea compleja, que se ve facilitada por el control de versiones. Si aparece un error (*bug*), la organización necesita saber qué versión contiene el *bug*, para poder corregirlo.

Repositorios

Un repositorio es un área de almacenamiento para archivos. Suele ser un directorio o carpeta, que contiene todos los archivos de cierto proyecto. Una forma manual (y muy poco efectiva) de control de versiones podría ser algo así:

```
usuario@PC:~/git$ ls              → ls es un comando que permite ver una lista de archivos.
programa-v1.py
programa-v2.py
programa-v2.2.py
programa-v3.py
programa-vfinal.py
programa-vfinalCorregido.py
programa-vfinalUltimaCorreccion.py
programa-vfinalUltimaCorreccionAhoraSi.py
```

Ante cada cambio, se le agrega un número al nombre de archivo, hasta llegar al final. Esto puede parecer sensato al principio, pero después de un par de meses, ¿quién va a recordar qué cambio hizo Fulano entre la v2.2 y la v3? Probablemente ni el propio Fulano se acuerde qué hizo...
Al trabajar con múltiples archivos en la misma carpeta, otra forma manual de control de versiones podría consistir en hacer copias de la carpeta entera. Pero claramente, este modo intuitivo de versionar controles, se convierte en un problema apenas el proyecto comienza a avanzar en su desarrollo.

Commit

Hacer commit[1] es registrar un cambio en un repositorio. En general, los sistemas de control de versiones no registran los cambios cada vez que se guarda un archivo, sino solamente cuando alguien hace un commit explícitamente.

1 Podríamos traducir el sustantivo "commit" como "confirmación", y el verbo “commit” como “confirmar”. Pero utilizaremos el término en inglés, porque es el más difundido en ambientes técnicos. En adelante, utilizaremos la palabra inglesa *commit* como sustantivo, y "hacer *commit*" como verbo.
Por favor, no lo conjuguen: es horrible oír frases como “lo que *commiteaste* el otro día”.

Git

Git[2] fue creado en 2005 por Linus Torvalds, pensado para optimizar el desarrollo del kernel Linux. Es Software Libre, distribuido bajo la licencia GPLv2 de GNU.es un sistema distribuido de control de versiones (DVCS, por sus siglas en inglés). Esto significa que no es necesario un servidor de Git. Ni siquiera hace falta tener conexión a una red para usar git.
Otros sistemas de control de versiones ponen el código en un nodo central, y se les da acceso a los desarrolladores para leer y escribir en un repositorio, en donde el código está almacenado. Este repositorio, suele estar en otro dispositivo, por lo que se requiere una red. A medida que se suman desarrolladores al proyecto, se incrementa la carga del servidor. Algunas acciones críticas sobre el código requerirán permisos especiales. Obviamente, el servidor que contiene el repositorio debe estar disponible para que los desarrolladores puedan trabajar. Este "uptime" implica costos extra de hardware y supervisión. Además, el servidor es un punto crítico de falla.
Git invierte esta lógica, dandole a cada desarrollador un repositorio de control de versiones propio. Cada repositorio corre íntegramente en la máquina local del desarrollador. Todos los desarrolladores pueden acceder a cualquier parte del historial del proyecto, comparar versiones, "ramificar" el proyecto y realizar cualquier otra operación que normalmente requeriría permisos especiales y acceso remoto a un servidor. Se trata de una situación a la que algunas personas les puede costar acostumbrarse. En lugar de pedirle permiso a los escpecialistas que administran el servidor, cada desarrollador puede hacer lo que necesita en su versión local. Obviamente, esto implica que hay que aprender acerca de las operaciones necesarias.
Desde luego, el hecho ser distribuido implica también algunas desventajas. Ciertos proyectos requieren no solamente coordinación, sino también un gran control sobre el trabajo del equipo, lo que puede dificultarse cuando no hay un servidor centralizado.
Esta idea de darle a cada desarrollador su propio repositorio, hace de Git un sistema de control de versiones distribuido (o descentralizado). Muchos grandes proyectos como el kernel Linux y el CMS Drupal, tienen miles de desarrolladores alrededor del mundo, y utilizan Git para coordinar su trabajo.

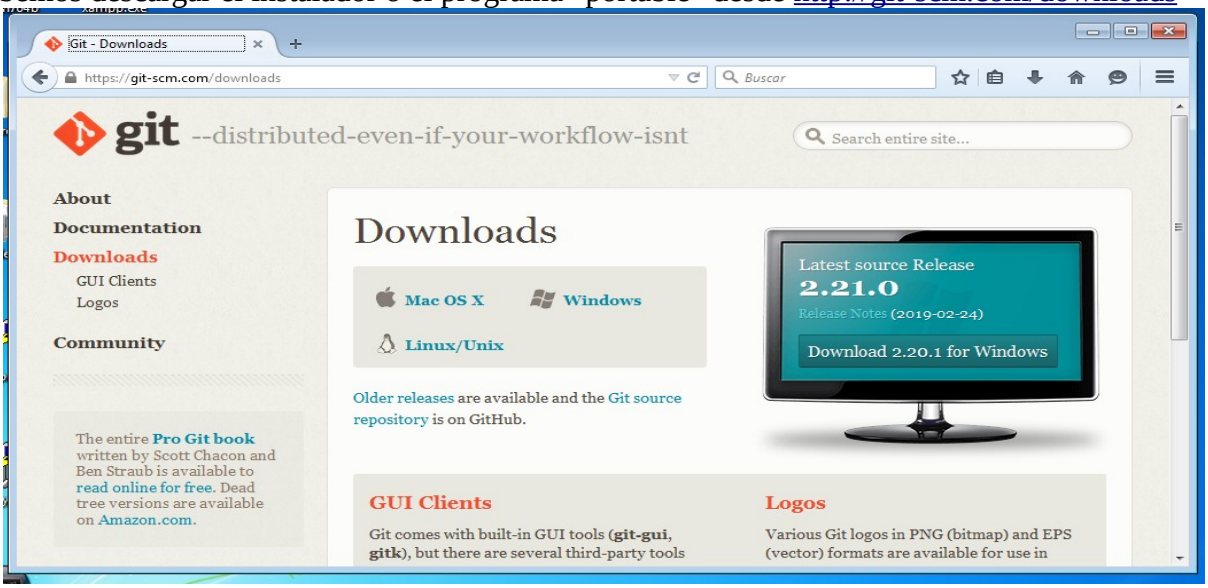
Instalación en sistemas GNU/Linux

En distribuciones derivadas de Debian (como Ubuntu, Huayra o Mint), es posible que git venga instalado por defecto. Para eso, podemos abrir una terminal y ejecutar el comando `git`. Si se reconoce el comando, es porque ya está instalado. Si no, se puede instalar ejecutando como root:
`apt-get install git`
O en Ubuntu y Mint: `sudo apt-get install git`

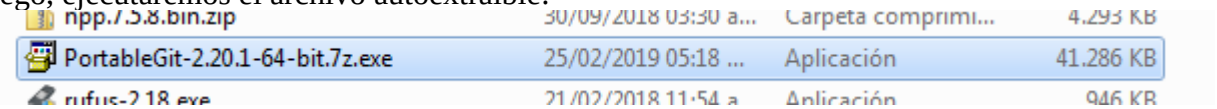
2 La letra "g" se pronuncia suave, como en "guitarra"

Instalación en Sistemas Windows

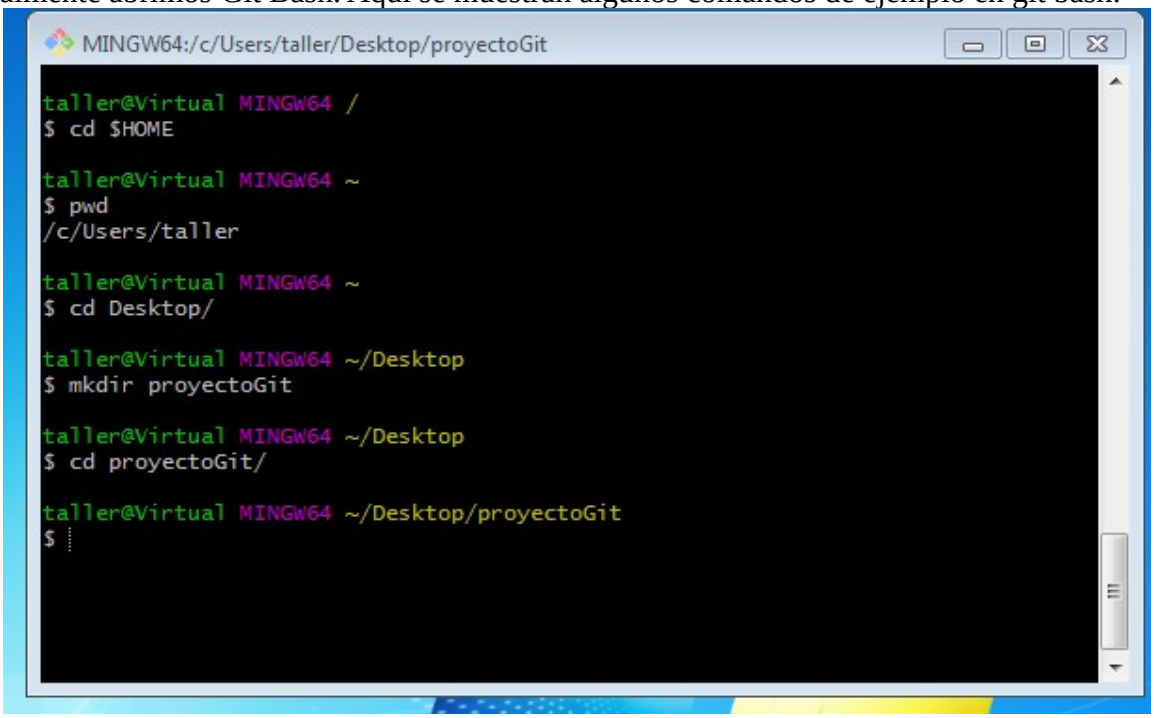
Debemos descargar el instalador o el programa “portable” desde http://git-scm.com/downloads



Luego, ejecutaremos el archivo autoextraible:



Y finalmente abrimos Git Bash. Aquí se muestran algunos comandos de ejemplo en git bash:



A partir de aquí, el uso de la consola (ya sea git bash en Windows o la terminal de GNU/Linux) es idéntico sin importar el sistema operativo.

Algunos comandos básicos de la terminal

cd: Cambiar de directorio (carpeta).

Este comando se utiliza para movernos por el árbol de carpetas. Por ejemplo, si queremos ir a una subcarpeta de la carpeta actual, llamada “equis”, ejecutaremos el comando **`cd equis`** Si queremos salir de la carpeta actual a la carpeta contenedora, ejecutaremos **`cd ..`** Si queremos ir a la carpeta personal del usuario ejecutaremos **`cd $HOME`**

pwd: “¿En qué carpeta estoy?”

Este comando simplemente devuelve la ubicación actual

mkdir: Crear carpeta

Este comando crea una nueva carpeta. Se utiliza escribiendo **`mkdir nombreDeLaCarpetaNueva`**

ls: “¿Qué hay aquí?”

Este comando permite ver el contenido de la carpeta actual.

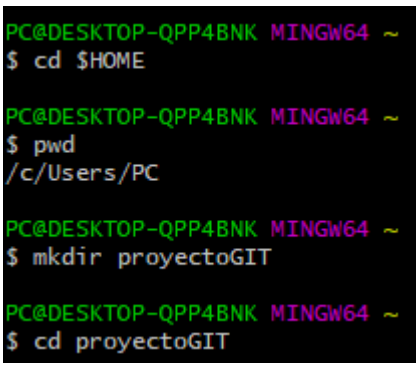
Configurando nuestro primer proyecto

A continuación crearemos una carpeta que será nuestro primer proyecto. Desde la terminal de GNU/ Linux o GitBash en Windows, ejecutaremos los siguientes comandos.
Primeramente, nos “moveremos” a la carpeta personal, con el comando

```
cd $HOME
```

Luego, crearemos la carpeta con un nombre adecuado, e ingresamos a ella.

```
mkdir proyectoGIT
cd proyectoGIT
```



“proyectoGIT” es un nombre inventado, puede tener el nombre que se prefiera.

Una vez dentro de nuestra carpeta de trabajo, debemos indicar que deseamos que esta carpeta se convierta en un proyecto de git. Hacemos esto con el comando

```
git init
```

A continuación, configuraremos nuestra identidad (nombre y correo electrónico) y el editor de nuestra preferencia[3].
Para configurar el editor, utilizaremos el comando

```
git config core.editor "ruta al editor"
```

Por ejemplo, para utilizar Sublime Text:

```
git config core.editor "’c:/Program Files/Sublime Text 3/sublime_text.exe’ -w”
```

Esto configurará el editor solamente para este proyecto. Si deseamos configurarlo para todos nuestros proyectos, debemos agregar `--global`:

```
git config --global core.editor "’c:/Program Files/Sublime Text 3/sublime_text.exe’ -w”
```

3 Por defecto, el editor de git es vim, una herramienta sumamente potente y productiva, pero bastante difícil de aprender a usar. Cada uno puede elegir el editor de su preferencia, pero teniendo en cuenta que, si no se dispone del tiempo necesario para aprender a usar vim, es mejor cambiar el editor por otro más intuitivo.

```
PC@DESKTOP-QPP4BNK MINGW64 ~/proyectoGIT
$ git init
Initialized empty Git repository in C:/Users/PC/proyectoGIT/.git/

PC@DESKTOP-QPP4BNK MINGW64 ~/proyectoGIT (master)
$ git config core.editor "'c:/Program Files/Sublime Text 3/sublime_text.exe' -w"

PC@DESKTOP-QPP4BNK MINGW64 ~/proyectoGIT (master)
$ git config core.editor --global "'c:/Program Files/Sublime Text 3/sublime_text.exe' -w"
```

En la imagen vemos ambas alternativas. Si se decide configurar globalmente (último comando), no hace falta ejecutar la configuración local (segundo comando).

Configuramos también nuestro nombre y correo electrónico, con los comandos:
`git config user.email` *`direccion_de_correo`*
`git config user.name` *`nombre_de_usuario`*

```
PC@DESKTOP-QPP4BNK MINGW64 ~/proyectoGIT (master)
$ git config user.email mi-correo@electronico.com

PC@DESKTOP-QPP4BNK MINGW64 ~/proyectoGIT (master)
$ git config user.name Juan
```

Aquí también es posible utilizar la opción `--global`, para identificarnos con nuestro nombre y correo en todos los proyectos que utilicen git.
Para ver cómo quedó nuestra configuración, podemos usar el comando
`git config -l`
(el último carácter es la letra “L” minúscula). Esto nos mostrará la lista de las configuraciones por defecto y, al final, las tres opciones que hemos definido.

```
PC@DESKTOP-QPP4BNK MINGW64 ~/proyectoGIT (master)
$ git config -l
core.symlinks=false
core.autocrlf=true
core.fscache=true
color.diff=auto
color.status=auto
color.branch=auto
color.interactive=true
help.format=html
rebase.autosquash=true
http.sslbackend=openssl
http.sslcainfo=C:/Program Files/Git/mingw64/ssl/certs/ca-bundle.crt
core.repositoryformatversion=0
core.filemode=false
core.bare=false
core.logallrefupdates=true
core.symlinks=false
core.ignorecase=true
core.editor='c:/Program Files/Sublime Text 3/sublime_text.exe' -w
user.email=mi-correo@electronico.com
user.name=Juan
```