Manual básico de uso de GIT

Conceptos básicos y terminología

GIT es un Systema de Control de versiones distribuido (DCVS). Cada desarrollador tiene una copia completa del repositorio, por lo que en caso de perdida del repositorio central, cualquier repositorio cliente se puede usar como respaldo. También esto permite que la mayoría de las operaciones se puedan realizar fuera de línea. Solo se requiere conexión al repositorio principal para recuperar los últimos cambios y subir los cambios propios.

Repositorio local: Es una copia privada completa del repositorio remoto.

Directorio de Trabajo y Area de Ensayo (Staging Area): El directorio de trabajo es el espacio donde el desarrollador realizará cambios a los archivos. A diferencia de otros CVS GIT no controla cada uno de los cambios realizados a cualquier archivo. A la hora de realizar un “commit” solo los archivos que están en la Staging Area serán considerados.

“Commit”: Representa el estado corriente del repositorio. Contiene todos los cambios realizados sobre los objetos del repositorio.

“Branches”: Son usadas para crear otra línea de desarrollo. Por omisión, GIT tiene una rama “master”. Usualmente se crea un “branch” para trabajar en una nueva funcionalidad, y una vez que el desarrollo está completo, la “branch” se combina (merge) con la rama “master” y se borra la “branch”.

“Tags” (Etiquetas): Se utilizan para asignarle un nombre significativo a una versión específica del repositorio. Los desarrolladores las utilizan para designar versiones específicas cuando son liberadas.

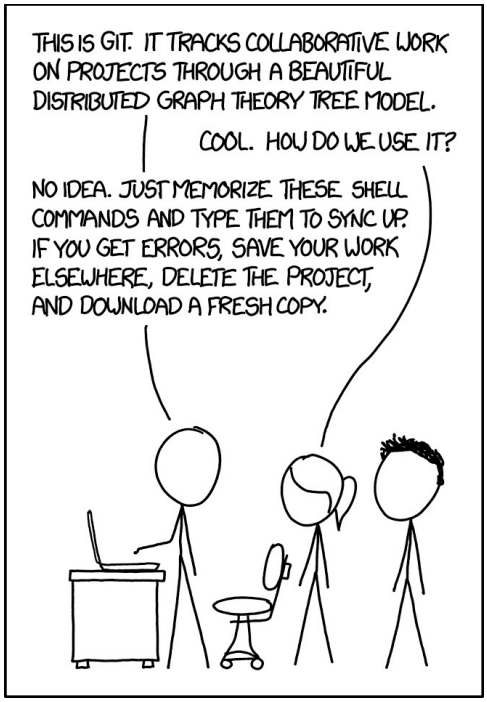
“Clone” (Clonar): crea una instancia de un repositorio. La operación de clonado no solo actualiza la copia de trabajo sino que crea una copia local completa del repositorio.

“Pull”: copia los cambios desde un repositorio remoto a una instancia local. Es utilizada para sincronizar dos instancias de un repositorio.

“Push”: copia los cambios de una instancia local a una remota de un repositorio.

Revisión: Es la versión de código. Las revisiones en GIT estan representadas por “commits”, que son identificados por hashes SHA1.

URL: Es la ubicación de un repositorio GIT.



Instalación

Distribuciones basadas en Debian
```
$ sudo apt install git
```

Distribuciones basadas en Red Hat
```
$ sudo yum install git
```

Generación de claves y configuración del acceso por SSH

Abrir git bash o una terminal en Linux.

```
$ ssh-keygen -t rsa -b 4096 -C "your_email@example.com"
```

Cuando se le indique ingresar un nombre de archivo, presione enter para dejar el valor por omisión.

```
> Enter a file in which to save the key (/c/Users/you/.ssh/id_rsa):[Press enter]
```

Ingrese una contraseña

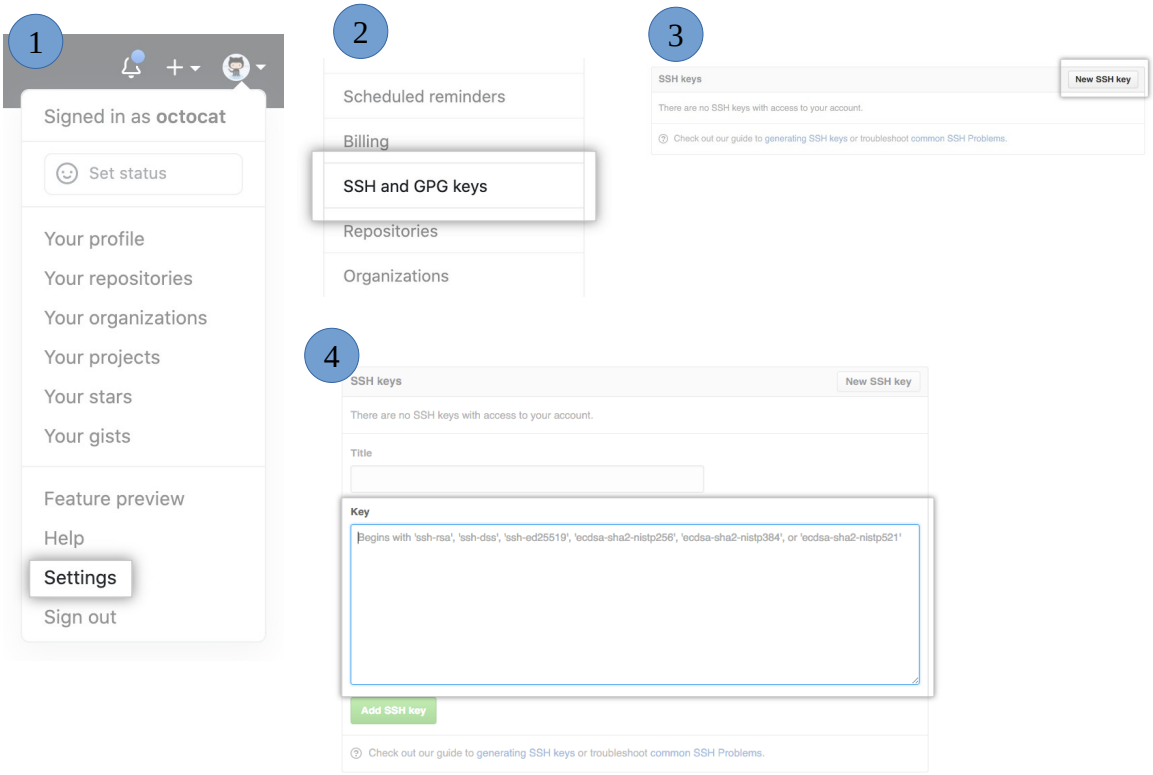
```
> Enter passphrase (empty for no passphrase): [Type a passphrase]
> Enter same passphrase again: [Type passphrase again]
```

Copiar el contenido de la clave pública y agregarlo en su cuenta de github
Para copiar el contenido de la clave pública, puede usar el comando cat.

```
$ cat .ssh/id_rsa.pub
```

y seleccionarla y copiarla usando el ratón.

En su cuenta de github, va al menú de su perfil, en la esquina superior derecha de la pantalla.



Para pasar las claves a su instalación de WSL ** Solo usuarios de Windows.
Desde la terminal de WSL

```
$ cp -r /mnt/c/<su-usuario>/.ssh ./
```

Cargar la contraseña de la clave pública y que no la pida cada vez que accedemos al repositorio remoto.

```
$ eval $(ssh-agent.exe)
Agent pid 2041

$ ssh-add
Enter passphrase for /c/Users/<usuario>/.ssh/id_rsa:
Identity added: /c/Users/<usuario>/.ssh/id_rsa (sucorreo@gmail.com)
```

**Los mismos comandos se ejecutan en Linux, pero los nombres de archivo son distintos.

Configuracion del ambiente GIT

Nombre de usuario

```
$ git config --global user.name "Jerry Mouse"
```

Dirección de correo electrónico

```
$ git config --global user.email "jerry@ibm.com"
```

Colores en los comandos de git en la consola

```
$ git config --global color.ui true
$ git config --global color.status auto
$ git config --global color.branch auto
```

Editor de textos

```
$ git config --global core.editor vim
```

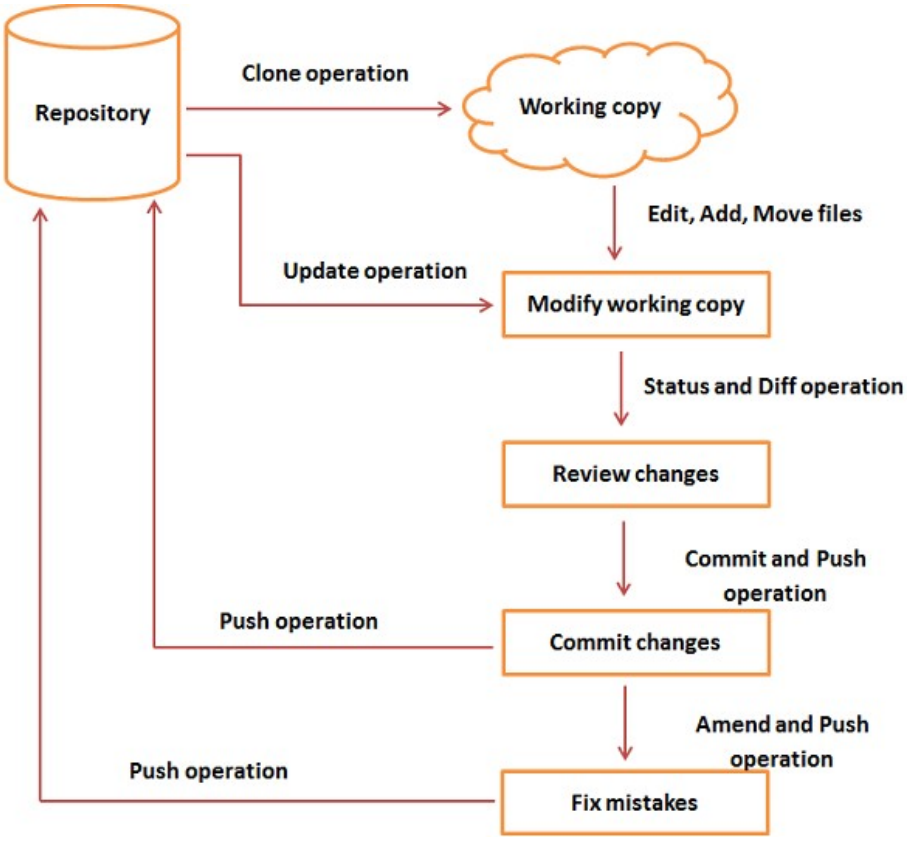
Herramienta para merges

```
$ git config --global merge.tool vimdiff
```

Listar la configuración

```
$ git config --list
```

Flujo de trabajo



El flujo de trabajo, en general, es el siguiente:

- Clonar el repositorio git
- Regularmente se debe actualizar la copia para tener los cambios de otros desarrolladores
- Realizar los cambios y agregar los que se quieren hacer commit a la “stage area”
- Se hace “commit” de los cambios en la instancia local. Si todo está OK, esos cambios se suben (“push”) al respositorio remoto.
- En el caso de que algo esté mal, se corrige el último commit y se suben los cambios al repositorio remoto.

Crear un repositorio

Se crea un directorio para el repositorio, se inicializa, se agrega un archivo, se hace el “commit” y para finalizar se sube al repositorio remoto.

```
$ mkdir proyecto
$ cd proyecto
$ git init
Initialized empty Git repository in /home/tom/proyecto/.git/
```

Se edita el archivo README.md para iniciar el repositorio

```
$ git status -s
?? README

$ git add .
```

Notar el . Indicando el directorio de trabajo

```
$ git status -s
A README

$ git commit -m 'Initial commit'
```

El comando anterior producirá el siguiente resultado:

```
[master (root-commit) 19ae206] Initial commit
1 files changed, 1 insertions(+), 0 deletions(-)
create mode 100644 README
```

Si no se usa la opción -m, git abrirá el editor para la editar la descripción del commit. Es recomendable escribir una descripción de los cambios lo más detallada posible.

```
$ git log
commit 19ae20683fc460db7d127cf201a1429523b0e319
Author: Tom Cat <tom@ibm.com>
Date: Wed Sep 11 07:32:56 2013 +0530

Initial commit
```

Para sincronizar el repositorio local con un repositorio remoto se debe ejecutar lo siguiente:

```
$ git remote add origin git@github.com:tom/proyecto
```

Deberá proporcionar la contraseña de la cuenta de github. Puede utilizar gitlab u otro servidor git.

```
$ git push origin master
```

El comando producirá la siguiente salida:

```
Counting objects: 3, done.
Writing objects: 100% (3/3), 242 bytes, done.
Total 3 (delta 0), reused 0 (delta 0)
To gituser@git.server.com:project.git
* [new branch]
master −> master
```

Clonado de un repositorio

```
$ git clone git@github.com:tom/proyecto
Initialized empty Git repository in /home/tom/proyecto
remote: Counting objects: 3, done.
Receiving objects: 100% (3/3), 241 bytes, done.
remote: Total 3 (delta 0), reused 0 (delta 0)
```

Una vez que realice cambios sobre el contenido del repositorio sigue el mismo proceso descrito en el apartado anterior

Una vez realizados cambios en el directorio de trabajo (creación y modificación de archivos) se deben agregar al “staging area” con el comando **git add.** Para ver los cambios que están pendientes se puede usar **git status**.

```
$ git status
# On branch main
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#   ansible.cfg
#   inventario
#   update.yml
nothing added to commit but untracked files present (use "git add" to track)
```

```
$ git add .

$ git status
# On branch main
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#   new file:   ansible.cfg
#   new file:   inventario
#   new file:   update.yml
#
```

Una vez agregados los cambios al staging area, se puede realizar el commit al repositorio local.

```
$ git commit -m "Agrego archivo de configuración, inventario y primer playbook"
[main bcca8fc] Agrego archivo de configuración, inventario y primer playbook
 3 files changed, 508 insertions(+)
 create mode 100644 ansible.cfg
 create mode 100644 inventario
 create mode 100644 update.yml
```

ATENCION: Si no se usa la opción -m para escribir un comentario en línea, se abrirá el editor de texto para hacerlo. NO se puede hacer un commit sin una descripción.

Una vez que hicimos el commit a nuestro repositorio local, ahora podemos sincronizar el repositorio local con el repositorio remoto haciendo **git push**.

```
$ git push
warning: push.default is unset; its implicit value is changing in
...

**Cuando se le indique debe ingresar la contraseña correspondiente a la clave de SSH

Counting objects: 6, done.
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 8.23 KiB | 0 bytes/s, done.
Total 5 (delta 0), reused 0 (delta 0)
To https://github.com/emverdes/ansible_taller2021
   ebf5c7e..bcca8fc  main -> main
```

Git status nos permite ver el estado del proyecto y si hay cambios pendientes.

```
$ git status
# On branch main
nothing to commit, working directory clean
```

El ciclo **add -> commit -> push** se repite cada vez que se realicen cambios en el proyecto.

ATENCIÓN: Si está trabajando en un proyecto con otros colaboradores **recuerde ejecutar git pull antes de hacer algún cambio.**

Obtener los últimos cambios de un repositorio

```
$ cd proyecto

$ git pull
```

Para ver los cambios realizados en el repositorio remoto

```
$ git log
```

Antes de hacer cambios en su instancia local es recomendable actualizar.

Suspender cambios

Ud. puede estar trabajando en una nueva implementación, pero debe parar para atender un problema reportado por un cliente. No puede guardar los cambios que tiene hechos, pero tampoco perderlos. La solución en poner esos cambios en el freezer con “stash”.

Stash toma sus archivos modificados y los guarda en un espacio temporal desde donde los podrá recuperar posteriormente.

```
$ git stash
Saved working directory and index state WIP on master: e86f062 Added my_strcpy function
HEAD is now at e86f062 Added my_strcpy function
```

Puede ver una lista de los cambios “stasheados”

```
$ git stash list
stash@{0}: WIP on master: e86f062 Added my_strcpy function
```

Para recuperar los cambios que están en “stash”

```
$ git stash pop
```

Revertir cambios

Si modificó un archivo equivocado o borró un archivo que no debía, puede revertir esos cambios usando el comando checkout.

```
$ git checkout nombre_archivo
```

Manejar ramas (“branches”)

La creación de ramas (“branches”) permite crear una nueva línea de desarrollo. Por ejemplo, podemos tener una rama de “producción” para corrección de errores y otra para la próxima versión donde desarrollamos la nueva funcionalidad.

Crear una rama

```
$ git branch nueva_rama

$ git branch
* master
nueva_rama
```

Cambiar de rama

```
$ git checkout nueva_rama
```

```
Switched to branch ‘nueva_rama’

$ git branch
master
* nueva_rama
```

Borrar una rama

```
$ git checkout master

$ git branch -D nueva_rama
```

Aplicar cambios de una rama a master

```
$ git checkout master

$ git merge nueva_rama
```