****

**UNIVERSIDAD PRIVADA DE TACNA**

**FACULTAD DE INGENIERIA**

**Escuela Profesional de Ingeniería de Sistemas**

**INFORME DE LABORATORIO Nº 03**

**“INSTALACION Y CONFIGURACION DE GIT”**

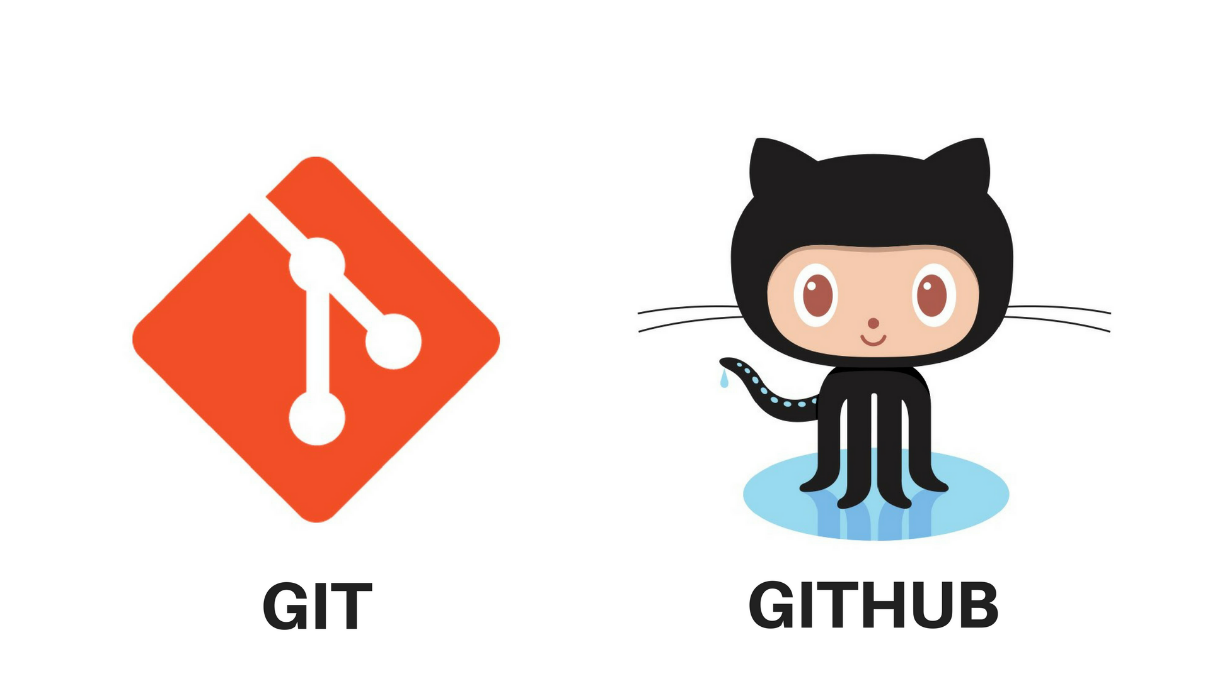
Curso: Programación II

Docente: Ing. Enrique Lanchipa Valencia

**Renzo Mamani Campos**

**Tacna – Perú**

**2020**

INDICE

[**I. INFORMACIÓN GENERAL 3**](#_Toc52722673)

[**- Objetivos: 3**](#_Toc52722674)

[**- Equipos, materiales, programas y recursos utilizados: 3**](#_Toc52722675)

[**II. MARCO TEORICO 3**](#_Toc52722676)

[**- Que es el GIT: 3**](#_Toc52722677)

[**III. PROCEDIMIENTO 6**](#_Toc52722678)

[**IV. CONCLUSIONES 11**](#_Toc52722679)

[**V. WEBGRAFIA 11**](#_Toc52722680)

**INFORME DE LABORATORIO Nº 03**

**TEMA: INSTALACION Y CONFIGURACION DE GIT**

1. **INFORMACIÓN GENERAL**

* **Objetivos:**

- Aprender el proceso de instalación y configuración del programa GIT

* **Equipos, materiales, programas y recursos utilizados:**
* Computadora con sistema operativo Windows 10
* Instalador del GIT

1. **MARCO TEORICO**

## Que es el GIT:

Git es un software de [control de versiones](https://es.wikipedia.org/wiki/Control_de_versiones) diseñado por [Linus Torvalds](https://es.wikipedia.org/wiki/Linus_Torvalds), pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando éstas tienen un gran número de archivos de [código fuente](https://es.wikipedia.org/wiki/C%C3%B3digo_fuente). Su propósito es llevar registro de los cambios en archivos de computadora y coordinar el trabajo que varias personas realizan sobre archivos compartidos.

Al principio, Git se pensó como un motor de bajo nivel sobre el cual otros pudieran escribir la interfaz de usuario o [front end](https://es.wikipedia.org/wiki/Front-end_y_back-end" \o "Front-end y back-end) como [Cogito](https://es.wikipedia.org/wiki/Cogito_(software)) o [StGIT](https://es.wikipedia.org/w/index.php?title=StGIT&action=edit&redlink=1" \o "StGIT (aún no redactado)). Sin embargo, Git se ha convertido desde entonces en un sistema de control de versiones con funcionalidad plena. Hay algunos proyectos de mucha relevancia que ya usan Git, en particular, el grupo de [programación](https://es.wikipedia.org/wiki/Programaci%C3%B3n) del [núcleo Linux](https://es.wikipedia.org/wiki/N%C3%BAcleo_Linux).

El [mantenimiento del software](https://es.wikipedia.org/wiki/Mantenimiento_de_Software) Git está actualmente (2009) supervisado por Junio Hamano, quien recibe contribuciones al código de alrededor de 280 programadores. En cuanto a derechos de autor Git es un [software libre](https://es.wikipedia.org/wiki/Software_libre) distribuible bajo los términos de la versión 2 de la [Licencia Pública General de GNU](https://es.wikipedia.org/wiki/GNU_General_Public_License).

CARACTERISTICAS:

* + Fuerte apoyo al desarrollo no lineal, por ende rapidez en la gestión de ramas y mezclado de diferentes versiones. Git incluye herramientas específicas para navegar y visualizar un historial de desarrollo no lineal. Una presunción fundamental en Git, es que un cambio será fusionado mucho más frecuentemente de lo que se escribe originalmente, conforme se pasa entre varios programadores que lo revisan.
  + Gestión distribuida. Al igual que [Darcs](https://es.wikipedia.org/wiki/Darcs" \o "Darcs), [BitKeeper](https://es.wikipedia.org/wiki/BitKeeper" \o "BitKeeper), [Mercurial](https://es.wikipedia.org/wiki/Mercurial), [SVK](https://es.wikipedia.org/w/index.php?title=SVK&action=edit&redlink=1), [Bazaar](https://es.wikipedia.org/wiki/Bazaar_(software)" \o "Bazaar (software)) y [Monotone](https://es.wikipedia.org/wiki/Monotone" \o "Monotone), Git le da a cada programador una copia local del historial del desarrollo entero, y los cambios se propagan entre los [repositorios](https://es.wikipedia.org/wiki/Repositorio) locales. Los cambios se importan como ramas adicionales y pueden ser fusionados en la misma manera que se hace con la rama local.
  + Los almacenes de información pueden publicarse por [HTTP](https://es.wikipedia.org/wiki/HTTP), [FTP](https://es.wikipedia.org/wiki/File_Transfer_Protocol), [rsync](https://es.wikipedia.org/wiki/Rsync" \o "Rsync) o mediante un protocolo nativo, ya sea a través de una conexión TCP/IP simple o a través de cifrado [SSH](https://es.wikipedia.org/wiki/SSH). Git también puede emular servidores [CVS](https://es.wikipedia.org/wiki/CVS), lo que habilita el uso de clientes CVS pre-existentes y módulos IDE para CVS pre-existentes en el acceso de repositorios Git.
  + Los repositorios Subversion y svk se pueden usar directamente con git-svn.
  + Gestión eficiente de proyectos grandes, dada la rapidez de gestión de diferencias entre archivos, entre otras mejoras de optimización de velocidad de ejecución.
  + Todas las versiones previas a un cambio determinado, implican la notificación de un cambio posterior en cualquiera de ellas a ese cambio (denominado autenticación criptográfica de historial). Esto existía en [Monotone](https://es.wikipedia.org/wiki/Monotone" \o "Monotone).
  + Resulta algo más caro trabajar con ficheros concretos frente a proyectos, eso diferencia el trabajo frente a CVS, que trabaja con base en cambios de fichero, pero mejora el trabajo con afectaciones de código que concurren en operaciones similares en varios archivos.
  + Los renombrados se trabajan basándose en similitudes entre ficheros, aparte de nombres de ficheros, pero no se hacen marcas explícitas de cambios de nombre con base en supuestos nombres únicos de nodos de sistema de ficheros, lo que evita posibles y desastrosas coincidencias de ficheros diferentes en un único nombre.
  + Realmacenamiento periódico en paquetes (ficheros). Esto es relativamente eficiente para escritura de cambios y relativamente ineficiente para lectura si el reempaquetado (con base en diferencias) no ocurre cada cierto tiempo.

ORDENES BASICAS:

* git init:

Esto crea un subdirectorio nuevo llamado. git, el cual contiene todos los archivos necesarios del repositorio – un esqueleto de un repositorio de Git. Todavía no hay nada en tu proyecto que esté bajo seguimiento.

* git fetch:

Descarga los cambios realizados en el repositorio remoto.

* git merge *<nombre\_rama>*:

Impacta en la rama en la que te encuentras parado, los cambios realizados en la rama “nombre\_rama”.

* git pull:

Unifica los comandos *fetch* y *merge* en un único comando.

* git commit -m "<mensaje>":

Confirma los cambios realizados. El “mensaje” generalmente se usa para asociar al *commit* una breve descripción de los cambios realizados.

* git push origin *<nombre\_rama>*:

Sube la rama “nombre\_rama” al servidor remoto.

* git status:

Muestra el estado actual de la rama, como los cambios que hay sin commitear.

* git add *<nombre\_archivo>*:

Comienza a trackear el archivo “nombre\_archivo”.

* git checkout -b *<nombre\_rama\_nueva>*:

Crea una rama a partir de la que te encuentres parado con el nombre “nombre\_rama\_nueva”, y luego salta sobre la rama nueva, por lo que quedas parado en esta última.

* git checkout -t origin/*<nombre\_rama>*:

Si existe una rama remota de nombre “nombre\_rama”, al ejecutar este comando se crea una rama local con el nombre “nombre\_rama” para hacer un seguimiento de la rama remota con el mismo nombre.

* git branch:

Lista todas las ramas locales.

* git branch -a:

Lista todas las ramas locales y remotas.

* git branch -d *<nombre\_rama>*:

Elimina la rama local con el nombre “nombre\_rama”.

* git push origin *<nombre\_rama>*:

Commitea los cambios desde el branch local origin al branch “nombre\_rama”.

* git remote prune origin:

Actualiza tu repositorio remoto en caso que algún otro desarrollador haya eliminado alguna rama remota.

* git reset --hard HEAD:

Elimina los cambios realizados que aún no se hayan hecho *commit*.

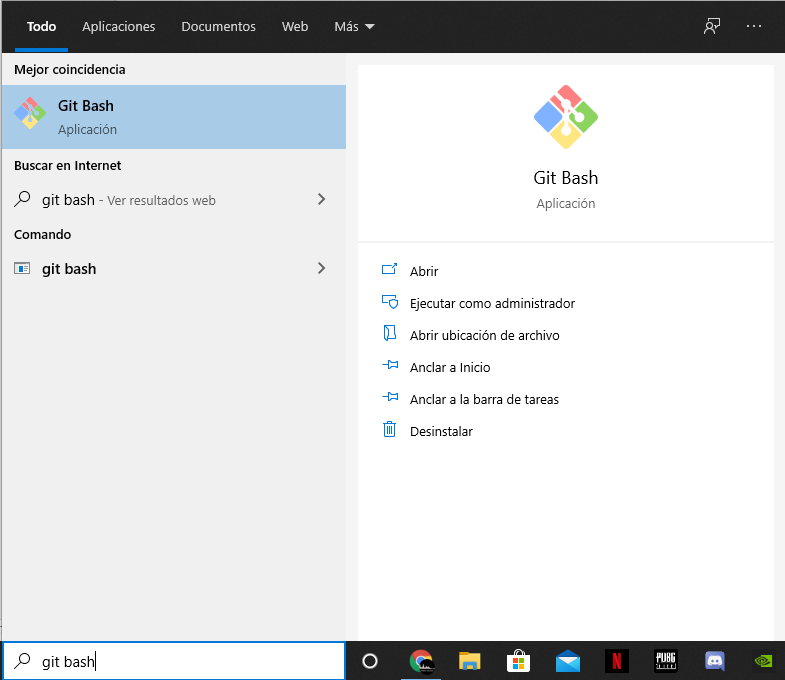
* git revert *<hash\_commit>*:

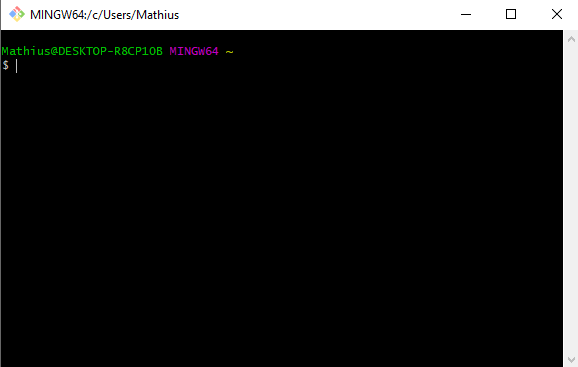
Revierte el *commit* realizado, identificado por el “hash\_commit”.

1. **PROCEDIMIENTO**

**Paso 1:**

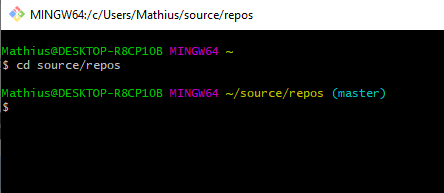
1. Buscamos la aplicación Git Bash y entramos





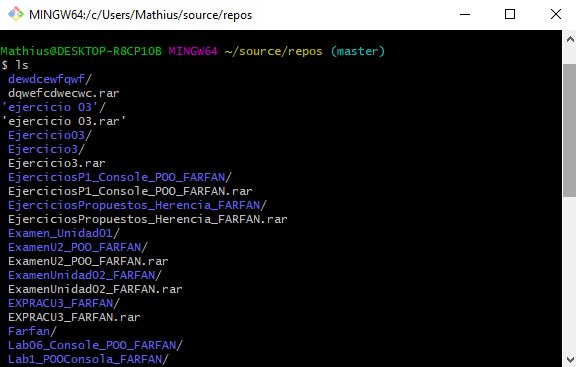
**Paso 2:**

1. Entramos a la carpeta donde tenemos los proyectos



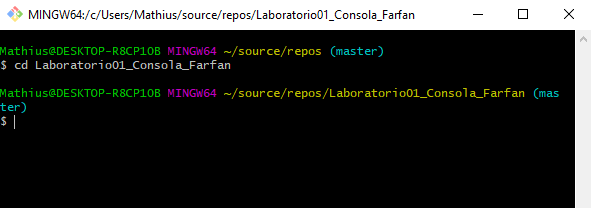
**Paso 3:**

1. Con el comando LS podemos ver todos los proyectos que tenemos en esa carpeta



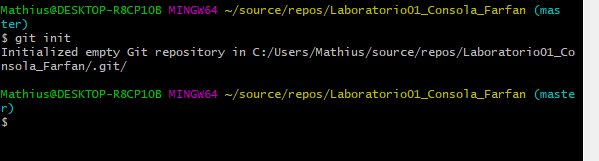
**Paso 4:**

1. Escogemos que proyecto queremos subir al GIT y entramos



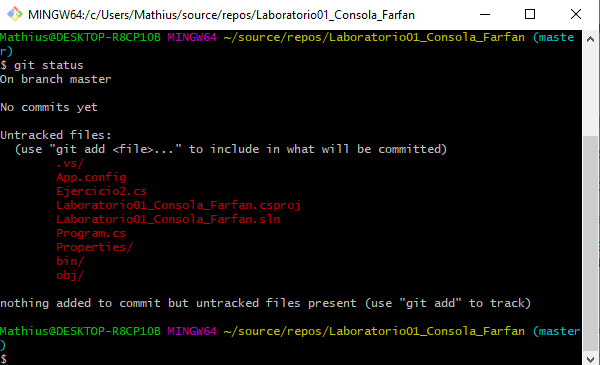
**Paso 5:**

1. Utilizamos el GIT INIT para inicializar el repositorio



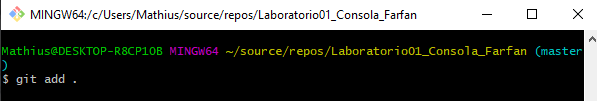
**Paso 6:**

1. Utilizamos GIT STATUS para ver el estado de los archivos, si están en rojo es que no están subidos.



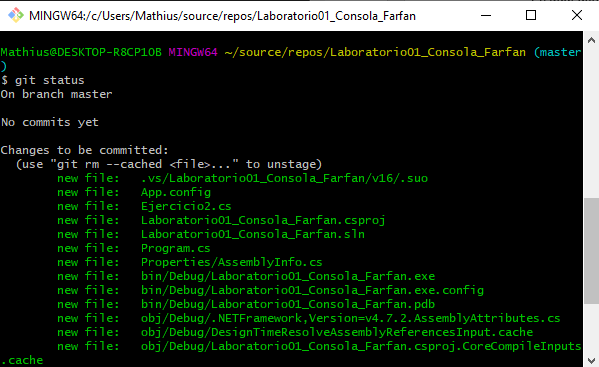
**Paso 7:**

a. Utilizamos GIT . para subir todos los archivos.



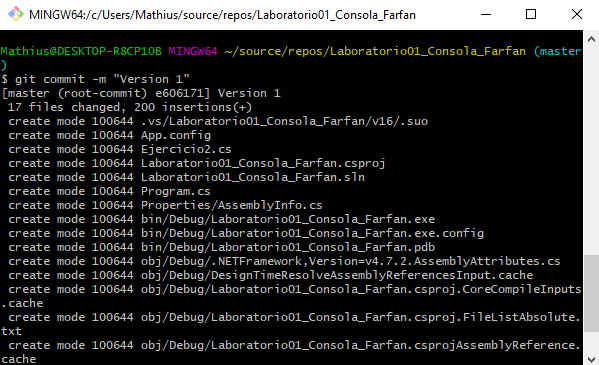
**Paso 8:**

1. Volvemos a utilizar GIT STATUS para confirmar que se subieron los archivos.



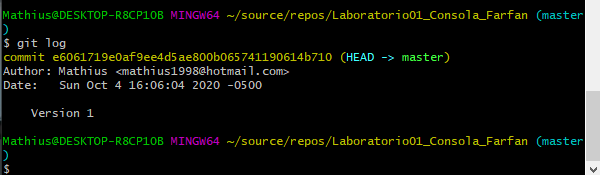
**Paso 9:**

1. Con GIT COMMIT -M creamos una versión.



**Paso 10:**

1. Con GIT LOG confirmamos que se creo correctamente la versión, lista para subir.



# CONCLUSIONES

Concluyo como estudiante que tanto como el git bash ( localmente) y git hub tienen diferentes comandos para interactuar con facilidad y subir , cambiar de ramas y crear commits en la red social de programadores.

# WEBGRAFIA

* GIT - <https://git-scm.com/>
* Instalación del GIT - <https://desarrolloweb.com/articulos/entiende-instala-configura-git.html>

