

**Es un sistema de control de versiones (Version Control System o VCS) 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.**

**Los sistemas de control de versiones centralizados (Centralized Version Control Systems o CVCSs ) son CVS, donde en un centro se mantiene por separado el usuario que genero el agregado son sus Subversion, y Perforce. Un único servidor contiene todos los archivos versionados, y varios clientes que descargan los archivos desde ese lugar central.Sistemas de control de versiones distribuidos**

**Los sistemas de control de versiones distribuidos (Distributed Version Control Systems o DVCSs , como Git, Mercurial, Bazaar o Darcs), poseen clientes que cuando descargan la última instantánea de los archivos, replican completamente el repositorio. Así, si un servidor muere, y estos sistemas estaban colaborando a través de él, cualquiera de los repositorios de los clientes puede copiarse en el servidor para restaurarlo. Cada vez que se descarga una instantánea, en realidad se hace una copia de seguridad completa de todos los datos**

**https://uniwebsidad.com/libros/pro-git**

**https://rogerdudler.github.io/git-guide/index.es.html**

**Instalar desde:**

[https://# git-scm.com/downloads](https://git-scm.com/downloads)

...siguiente ->siguiente ->siguiente ->siguiente ->

agregar al PATH y un bash (use git and optiona Unix tools from the Windows Command Prompt)

siguiente ->siguiente ->siguiente ->siguiente →….

**Teoria**

<https://rogerdudler.github.io/git-guide/index.es.html>

<https://git-scm.com/book/es/v1/Empezando>

<https://www.youtube.com/watch?v=HiXLkL42tMU>

Bash – consola

Lugares que componen tu repositorio

A) Working directory area o Directorio de trabajo

donde uno graba lo generado en el IDE, + imagenes, temporales, notas,etc

B) Staging Area o Index Área de preparado para el guardado

C) HEAD - Internal Repository donde se apunta el ultimo commit realizado

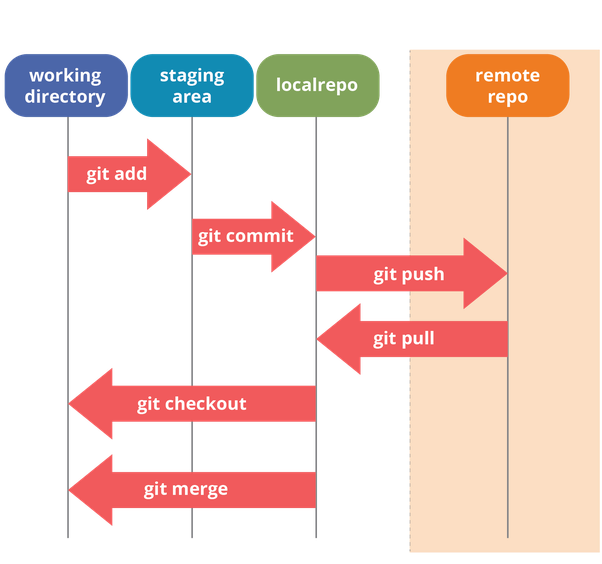
External Repository https://gitlab.com || https//github.com

Ramas generan versiones aisladas una de otra

Master es la rama/branch "por defecto" cuando creas un repositorio.

Durante el desarrollo se crean con versiones distintas

Luego se fusiónalas a la rama principal (master branch) cuando termines de optar por el mejor camino.



**Comandos de consola**

md CURSO\_PYTHON

cd CURSO\_PYTHON

# git init Crea un repositorio de # git vacío o reinicia el que ya existe

# git config --global user.name "User"

# git config --global user.email "[email@server.com](mailto:email@server.com)"

# git config --list

# git clone </path/to/repository> <directorio\_nuevo>

Clona un repositorio interno dentro de un directorio nuevo

Internal repository staging ==> working area

Crea una copia local del repositorio ejecutando

# git clone <[username@host](mailto:username@host):/path/to/repository> <directorio\_nuevo>

Clona un repositorio remoto dentro de un directorio nuevo

External repository staging ==> working area

# git add <archivo / directorio>

Agrega contenido de carpetas al indice para hacer commit

working ==> staging

# git add . <archivos / directorios>

Agrega todo contenido de carpetas al indice para hacer commit

working ==> staging

# git commit <archivo / directorio> “empaqueta los items en una versión finalizada”

Graba los cambios en tu repositorio

staging ==> Internal repository

# git commit -m "Commit + mensaje sin entrar al editor"

# git pull

Para actualizar tu repositorio local al commit más nuevo, ejecuta en tu directorio de trabajo para *bajar* y *fusionar* los cambios remotos.

Se aplicarán todos los cambios más recientes de la rama que estás usando.

# git pull <origen> master

# git pull <https://github.com/CursosAGT/xxx.git> master

# git remote **add** origin <server>

https://github.com/CursosAGT/xxx.git

# git remote **rename** origin <cambio\_de\_git>

cambio “origin” de un repositorio en linea a otro

# git remote add origin <nuevo\_server>

https://github.com/path/CURSO\_xxx.git

# git push <archivo / directorio>

Actualiza referencias remotas junto con sus objetos asociados

staging ==> External repository

# git push origin <branch>

# git push -u origin master (–all si hay mas de una rama)

# git push --set-upstream origin master

# git mv <archivo / directorio>

Mueve o cambia el nombre a archivos, directorios o enlaces

simbólicos

# git diff Muestra los cambios entre commits, commit y árbol de trabajo, etc

<source\_branch / rama\_fuente> <target\_branch/rama\_objetivo>

# git merge <branch / rama> “alternativa”

Fusiona dos ramas de desarrollo juntas, la activa (master), y la alternativa

Si bien “git” fusionara por si solo la mayor cantidad de cambios, deberás solucionar conflictos manualmente. “git” mostrara las diferencias con conflictos. La forma de revisar las diferencias es # git diff <source\_branch / rama\_fuente> <target\_branch / rama\_objetivo>

Luego de tomar la decisión debes marcar los fusionados con git add <Archivo>

# git rebase

Vuelve a aplicar commits en la punta de otra branch / rama

# git fetch origin

Descarga objetos y referencias de otro repositorio.

Si quieres deshacer todos los cambios locales y commits, puedes traer la última versión del servidor y apuntar a tu copia local principal

de esta forma

# git fetch origin master

Realiza un fetch e integra con otro repositorio o branch / rama local

# git reset

Reinicia el HEAD actual a un estado especifico.

# git reset --hard origin/master

Para actualizar tu repositorio local al commit más nuevo, ejecuta

en tu directorio de trabajo para bajar y fusionar los cambios remotos.

# git bisect

use la búsqueda binaria para encontrar el commit que introdujo el bug

# git grep

Imprime las líneas que concuerdan con el patron.

# git log

Muestra los logs de los commits.

# git show

Muestra varios tipos de objetos.

# git status

Muestra el estado o área de trabajo en que esta.

# git rm -rf <archivo / directorio>

Remueve del árbol de trabajo y del indice.

# git checkout <archivo / directorio>

Restaurar los cambios en el directorio de trabajo.

# git checkout -b <branch / rama\_alternativa>"

Crea una rama nueva ”alternativa”

# git checkout <branch / rama>

Cambia de una branch / rama de trabajo a otra.

# git checkout <nombre de la etiqueta / tag>

# git branch

lista los proyectos y ramas ”master branch ”

# git branch <branch / rama\_alternativa>"

Crea una branch / rama nueva ”alternativa”

# git branch -d <branch / rama\_alternativa>"

Borra una branch / rama ”alternativa”

# git tag <0.0.1\_desde\_oficina>

Crea <0.0.1\_desde\_oficina>

# git tag lista todo los tags

# git tag -v <0.0.1\_desde\_oficina> verifica el tag

# git tag -d <0.0.1\_desde\_oficina> borra el tag

Es una buena practica crear etiquetas para cada nuevo commit con un tag único de hasta 10 caracteres que remite al commit id al cual quieres referirte con tu etiqueta. Obten el commit id con git log

Este comando reemplaza los cambios en tu directorio de trabajo con el último contenido de HEAD. Los cambios que ya han sido agregados al Index, así como también los nuevos archivos, se mantendrán sin cambio.

Por otro lado, si quieres deshacer todos los cambios locales y commits, puedes traer la última versión del servidor y apuntar a tu copia local principal de esta forma

git fetch origin

git reset --hard origin/master

PR pullrequest

https://gist.github.com/domingogallardo/5bd3c185f3162d7a0c49

Ejercicio 1 crear una carpeta

generar un 1ro.txt “hola UTN”

win R ==> cmd

cd hasta ubicarme en el directorio

# git init ==> inicializar un proyecto en esa carpeta

# git status

generar un 2do.txt con “UTN 2019”

# git status

# git commit

==>

//# git config --global user.name "User"

//# git config --global user.email "email@server.com"

# git config --global core.editor vim

git config --list

user.name= User

user.email= email@server.com

color.status=auto

color.branch=auto

color.interactive=auto

color.diff=auto

# git commit

==>

Agrego Nombre y fecha

Esc wq!

# git commit

Muestra el seguimiento de cada archivo

puedes registrar cambios (añadirlos al Index) usando

git add <filename>

git add .

Este es el primer paso en el flujo de trabajo básico. Para hacer commit a estos cambios usa

git commit -m "Commit message"

Ahora el archivo esta incluído en el HEAD, pero aún no en tu repositorio remoto.

# git log

Muestra que y quien ha modificado

modifico 2do.txt “UTN 2020”

# git status

- (Menos ) lo que se ha quitado

+(Mas) lo que sa ha agregado

# git checkout - - 2do.txt

modifico un 2do.txt “UTN 2019 2do c ”

# git status

# git add 2do.txt

# git commit

# git log

# git status

# git log

Ejercicio 2

generar un ejemplos.txt “que es la teoría de clases al revés”

# git status

generar un .gitignore con “ejemplos.txt”

clear

ls

# git status

==>

ubicar entre el ls y el status las diferencias

**Comandos de consola agrego un segundo hilo**

# git branch "lista los proyectos y ramas” master

# git branch alternativa "genero una rama nueva” alternativa

# git branch "lista los proyectos y ramas” alternativa master

# git checkout alternativa "cambio de rama” de master a alternativa

Tus cambios están ahora en el HEAD de tu copia local. Para enviar estos cambios a tu repositorio remoto ejecuta

git push origin master

Reemplaza master por la rama a la que quieres enviar tus cambios.

Si no has clonado un repositorio ya existente y quieres conectar tu repositorio local a un repositorio remoto, usa

git remote add origin <server>

Ahora podrás subir tus cambios al repositorio remoto seleccionado.

Las ramas son utilizadas para desarrollar funcionalidades aisladas unas de otras. La rama *master* es la rama "por defecto" cuando creas un repositorio. Crea nuevas ramas durante el desarrollo y fusiónalas a la rama principal cuando termines.



Crea una nueva rama llamada "feature\_x" y cámbiate a ella usando  
git checkout -b feature\_x  
 vuelve a la rama principal  
git checkout master  
 y borra la rama  
git branch -d feature\_x  
Una rama nueva *no estará disponible para los demás* a menos que subas (push) la rama a tu repositorio remoto  
git push origin <branch>

Ejercicio

3

generar un alternativa.txt “rama alternativa”

# git status

# git add alternativa.txt

# git commit -m “rama alternativa”

==> -m permite agregar comentario sin entrar al editor de texto

# git log

==> muestra las ramas y donde estoy parado

# git checkout master

# git checkout -f master

**github**

echo "# CURSO\_xxx" >> README.md

# git init

# git add README.md

# git commit -m "first commit"

# git remote add origin https://github.com/path/CURSO\_xxx.git

# git push -u origin master

==> usuario

==> contraseña

**gitlab**

//# git config --global user.name "User"

//# git config --global user.email "email@server.com"

##### Create a new repository

# git clone https://gitlab.com/path/CURSO\_xxx.git

cd practicas\_python

touch README.md

# git add README.md

# git commit -m "add README"

# git push -u origin master

##### Push an existing folder

cd existing\_folder

# git init

# git remote add origin https://gitlab.com/path/CURSO\_xxx.git

# git add .

# git commit -m "Initial commit"

# git push -u origin master

##### Push an existing # git repository

cd existing\_repo

# git remote rename origin old-origin

# git remote add origin https://gitlab.com/path/CURSO\_xxx.git

# git push -u origin --all

# git push -u origin --tags

Notas:

**Opciones para el mensaje del commit**

-F, --archivo <carpeta> leer mensaje desde un archivo

--author <autor> sobrescribe el autor del commit

--date <fecha> sobrescribe la fecha del commit

-m, --message <mensaje> mensaje del commit

-c, --reedit-message <confirmar> reusar y editar el mensaje de un commit especifico

-C, --reuse-message <confirmar>reusar el mensaje de un commit especifico

--fixup <confirmar> usar mensaje de formato autosquash para arreglar el commit especificado

-squash <confirmar> usar el mensaje de formato autosquash para realizar squash al commit especificado

--reset-author el autor del commit soy yo ahora (usado con -C/-c/--amend)

-s, --signoff agregar Signed-off-by: (firmado por)

-t, --template <carpeta> usar archivo de template especificado

-e, --edit forzar la edición del commit

--cleanup <default> cómo quitar espacios y #comentarios de mensajes

--status incluir status en el template del mensaje de commit

-S, --gpg-sign[=<key-id>] firmar commit con GPG

**Opciones para el contenido del commit**

-a, --all confirmar todos los archivos cambiados

-i, --include agregar archivos específicos al índice para confirmar

--interactive agregar archivos interactivamente

-p, --patch agregar cambios interactivamente

-o, --only sólo confirmar archivos específicos

-n, --no-verify evitar los capturadores (hooks) de pre-commit y commit-msg

--dry-run mostrar lo que sería incluído en el commit

--short mostrar status de manera concisa

--branch mostrar información de la rama

--porcelain output formato-maquina

--long mostrar status en formato largo (default)

-z, --null terminar entradas con NUL

--amend enmendar commit previo

--no-post-rewrite gancho bypass post reescritura

-u, --untracked-archivos[=<modo>]