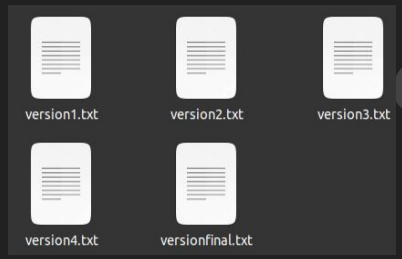
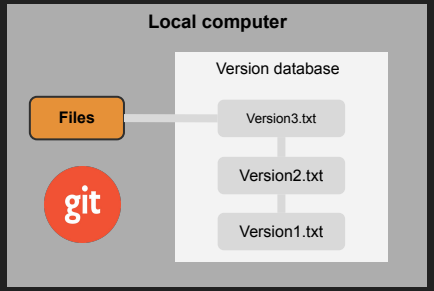
Introducción a GIT

sebasjp

Qué es GIT?

- Es un sistema de control de versiones que originalmente fue diseñado para operar en un entorno Linux. Actualmente GIT es compatible con Linux, MacOS y Windows.
- Git está optimizado para guardar todos estos cambios de forma incremental.





- En lugar de guardar un mismo archivo varias veces con diferente nombre para versionar, git nos ayuda hacerlo de manera automática, solo guardando los cambios necesarios.
- Además maneja los cambios que otras personas hagan sobre los mismos archivos.
- Git permite recuperar una versión antigua del proyecto de manera precisa.

git

Comenzando con GIT

- Para trabajar con GIT utilizaremos la terminal de nuestro computador.

1. Crearemos una carpeta en nuestro computador

Abriremos la terminal y nos ubicamos dentro de la carpeta creada

2. Creamos nuestro primer repositorio: **git init**

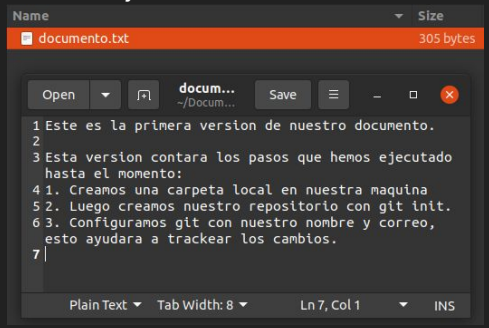
Esto creará una carpeta oculta **.git** donde se encuentra la base de datos de versiones

3. Dado que git está optimizado para trabajar en equipo, debemos configurar nuestro nombre y correo:

```
git config --global user.email "tu@email.com"
```

```
git config --global user.name "tu nombre"
```


4. Creamos un documento txt dentro de la carpeta de trabajo.



5. Ejecutamos el comando **git status** en la terminal

```
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        documento.txt

nothing added to commit but untracked files present (use "git add" to track)
```

5. Ejecutamos el comando **git add documento.txt** y nuevamente **git status**

```
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   documento.txt
```

5. Finalmente **git commit -m "primera version"** envía los cambios al repo

```
git commit -m "Primera version del documento /primer commit: contiene los pasos hasta la config del repositorio local"
[master (root-commit) 577f546] Primera version del documento /primer commit: contiene los pasos hasta la config del repositorio local
 1 file changed, 7 insertions(+)
 create mode 100644 documento.txt
```

Ciclo básico de trabajo en GIT

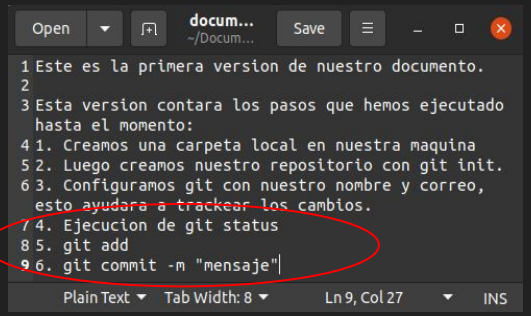
Working directory → `$ git add` → **Staging Area** → `$ git commit` → **Git repository**

Es una copia de una versión del proyecto, son archivos que se encuentran en disco para que se puedan usar o modificar.

Es un área donde se guardan temporalmente los archivos/cambios, para luego ser enviados al repositorio.

.git directory. Es el repositorio local donde Git almacena los metadatos y la base de datos de objetos para el proyecto. Es donde se guarda el historial de cambios de nuestros archivos

Actualicemos el documento.txt con los comandos que se han ejecutado



`$ git status`

```
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committ
ed)
  (use "git restore <file>..." to discard changes in work
ing directory)
        modified:   documento.txt

no changes added to commit (use "git add" and/or "git com
mit -a")
```

`$ git add documento.txt`

`$ git commit -m “mensaje”`

```
 git commit -m "Segunda version del documento: comandos b
asicos aniadidos"
[master 260a26f] Segunda version del documento: comandos
basicos aniadidos
 1 file changed, 3 insertions(+), 1 deletion(-)
```

Ciclo de vida o estado de los archivos en GIT

- **Untracked:** Son archivos que NO viven dentro de Git, solo en el disco duro. Nunca han sido afectados por **$ git add**, así que Git no tiene registros de su existencia.

```
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will b
e committed)
        documento.txt

nothing added to commit but untracked files present
(use "git add" to track)
```

- **Staged:** son archivos en Staging. Viven dentro de Git y hay registro de ellos porque han sido afectados por el comando **$ git add**, aunque no sus últimos cambios. Git ya sabe de la existencia de estos últimos cambios, pero todavía no han sido guardados definitivamente en el repositorio porque falta ejecutar el comando **$ git commit**.

```
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   documento.txt
```

- **Tracked:** viven dentro de Git, no tienen cambios pendientes y sus últimas actualizaciones han sido guardadas en el repositorio gracias a los comandos **$ git add** y **$ git commit**

- **Unstaged:** Son archivos que viven dentro de Git pero estàn desactualizados, sus últimas versiones solo están guardadas en el disco duro.

```
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committ
ed)
  (use "git restore <file>..." to discard changes in work
ing directory)
        modified:   documento.txt

no changes added to commit (use "git add" and/or "git com
mit -a")
```

Comandos para mover archivos entre los estados de GIT

- **$ git status**: nos permite ver el estado de todos nuestros archivos y carpetas.
- **$ git add**: nos ayuda a mover archivos del *untracked* o *unstaged* al estado *Staged*. Podemos usar git **$ git add nombre-del-archivo** para añadir archivos y carpetas individuales o **$ git add .** para mover todos los archivos de nuestro proyecto (tanto *untrackeds* como *unstageds*).
- **$ git reset HEAD**: nos ayuda a sacar archivos del estado *Staged* para devolverlos a su estado anterior. Si los archivos venían de *Unstaged*, vuelven allí. Y lo mismo se venían de *Untracked*.
- **$ git commit -m "descripcion o mensaje"**: nos ayuda a mover archivos de *Unstaged* a *Tracked*.
- **$ git rm**: este comando necesita alguno de los siguientes argumentos para poder ejecutarse correctamente:
  - **$ git rm --cached**: Mueve los archivos que le indiquemos al estado Untracked.
  - **$ git rm --force**: Elimina los archivos de Git y del disco duro.

Comandos para analizar cambios

$ git log: Permite ver la historia de los commits

```
commit 260a26f6f2ca5d5acfa07e3aad01d8f2ffdddcf6 (HEAD -> master)
Author: sebasjp <                                              >
Date:   Sat Mar 26 15:26:46 2022 -0500

    Segunda version del documento: comandos basicos aniadidos

commit 577f546af7e6d045ff5e63e71ca481102d9ad4b3
Author: sebasjp <                                              >
Date:   Sat Mar 26 13:46:16 2022 -0500

    Primera version del documento /primer commit: contiene los pasos hasta la c
onfig del repositorio local
```

$ git diff commit1 commit2: Permite comparar los cambios realizados entre dos commits

```
git diff 577f546af7e6d045ff5e63e71ca481102d9ad4b3 260a26f6
f2ca5d5acfa07e3aad01d8f2ffdddcf6
diff --git a/documento.txt b/documento.txt
index 3c5272e..fa1666d 100644
--- a/documento.txt
+++ b/documento.txt
@@ -4,4 +4,6 @@ Esta version contara los pasos que hemos e
jecutado hasta el momento:
 1. Creamos una carpeta local en nuestra maquina
 2. Luego creamos nuestro repositorio con git init.
 3. Configuramos git con nuestro nombre y correo, esto ayu
dara a trackear los cambios.
+
+4. Ejecucion de git status
+5. git add
+6. git commit -m "mensaje"
```

Cambios añadidos

Git checkout

$ git checkout: es un comando muy útil que permite que nos movamos entre **versiones**, **crear ramas** y **movernos entre ramas.**

- **$ git checkout <commit>**: nos permite regresar a una versión anterior del proyecto. **$ git checkout <commit> documento.txt** permite regresar a una versión anterior del archivo especificado.
- **$ git checkout master**: nos permite ir a la última versión enviada del proyecto.

Nota: Es importante tener cuidado con **$ git checkout** ya que si tenemos cambios en unstaged o untracked, al realizar el checkout estos cambios se pueden perder.

Qué es una rama?

En GIT, una **rama** o **branch** es una versión del código del proyecto sobre el que estás trabajando. Estas ramas ayudan a mantener el orden en el control de versiones y manipular el código de forma segura. Por defecto, el proyecto se crea en una rama llamada master.

Se puede ver como el mapa lineal de los commits que se han realizado al proyecto.

Todos los commits se aplican sobre una rama. Por defecto, siempre empezamos en la rama master y generamos nuevas ramas a partir de esta, para crear flujos de trabajo independientes.

$ git checkout -b nombre-rama: Con este comando creamos una rama desde algún commit de la rama master.

Ramas y Merge

Rama Master: usualmente es la versión en producción de los proyectos. Libre bugs y desde donde cualquier nueva versión o desarrollo debería partir.

Rama Features: son ramas que se crean de acuerdo a las nuevas funcionalidades/versiones que se agregaran al proyecto.

Rama HotFix: ramas que se crean a partir de la master, se utilizan para corregir errores y bugs de la versión en producción.

Para unir los cambios entre ramas, se utiliza git merge. Esto usualmente se hace desde la rama master con otra rama. Se debe tener en cuenta que puede haber conflicto entre las ramas, que deben ser resueltos de manera manual.