



Fundamentos de GIT y GitHub

Fundamentos de GIT (Parte I)

***Gestionar el código fuente
utilizando GitHub para
mantener un repositorio de
código remoto seguro y
permitir trabajo concurrente.***

- Unidad 1: Ambiente de desarrollo y sus elementos de configuración.
- Unidad 2: Elementos de la interfaz, navegación e interacción.
- Unidad 3: Fundamentos de GIT y GitHub.



Te encuentras aquí



¿Qué aprenderás en esta sesión?

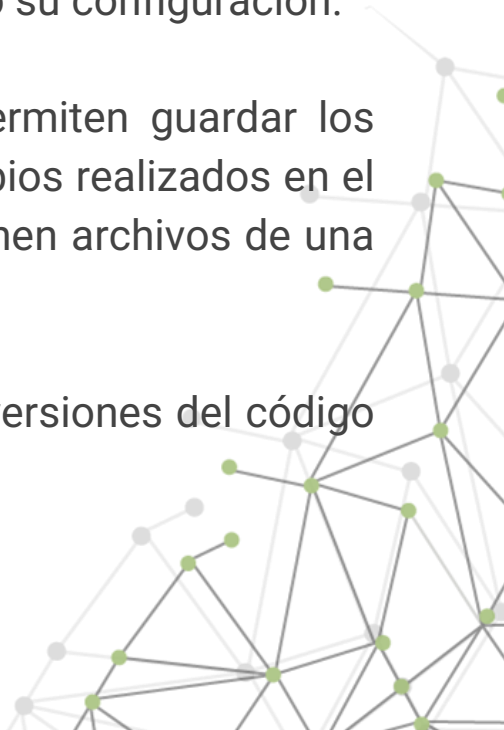
Utilizar un sistema de control de versiones para administrar los cambios en un proyecto

Si quieres compartir tu
código y sus
actualizaciones con otra
persona, ¿cómo lo
haces?



Control de versiones

- Es la gestión de los cambios que se realizan sobre un producto o su configuración.
- Los sistemas de control de versiones son programas que permiten guardar los cambios en el código, teniendo acceso a la historia de los cambios realizados en el proyecto, evitando que al hacerlo manual se modifiquen o eliminen archivos de una copia incorrecta del código.
- Los sistemas de control de versiones permiten administrar las versiones del código del mismo proyecto en forma centralizada.



/* Git */

¿Qué es?

Según su web oficial, Git es un software de control de versiones, gratuito y de código abierto diseñado para manejar proyectos de todos los tamaños con rapidez y eficiencia en forma.



Características

- Git es un sistema de control de versiones que gestiona localmente los cambios en los archivos versionados.
- Toda la información de versionamiento se encuentra en forma local en la carpeta .git.
- Al inicializar Git se crea por defecto la rama principal, **master** o **main**.
- Es software libre.



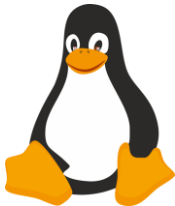


No es lo mismo que GitHub.

Instalación

Proceso

Descargar desde la [página oficial](#).

	Para Windows se descarga el instalador correspondiente .
	Para Mac se instala directamente con homebrew . <code>\$ brew install git</code>
	Para Linux se instala directamente con el gestor de paquetes correspondiente. <code>\$ apt-get install git</code> <code>\$ yum install git</code> <code>\$ apk add git</code>

Configuración



- Con Git instalado, es necesario personalizar el entorno.
- Esto se hace solamente una vez y se mantiene entre actualizaciones.
- `git config` es la herramienta que permite obtener y asignar variables de configuración de Git y se guardan en un archivo de configuración, generalmente un archivo oculto en el Home del usuario (archivo `~/.gitconfig`)

Configuración

Establecer tu identidad

- Debes establecer tu nombre de usuario y correo electrónico.
- Esta información se usa para guardar cada versión del código.

```
$ git config --global user.name "John Doe"
```

```
$ git config --global user.email john Doe@example.com
```

- Puedes comprobar tu configuración ejecutando el comando:

```
git config --list
```



¿Cómo obtener ayuda?

Visita *Git-scm.com*

- Si alguna vez necesitas ayuda para usar Git puedes revisar la página del manual ejecutando uno de estos comandos:

```
git
```

```
git help <verb>
```

```
git <verb> --help
```

```
man git-<verb>
```

- Donde <verb> es algún comando de git, por ejemplo:

```
git help config
```

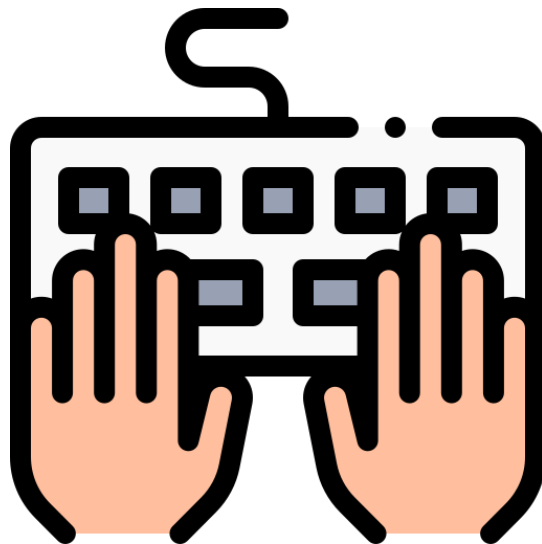
ACTIVIDAD

- ❑ Instalar Git
- ❑ Verificar la instalación de Git ejecutando el comando `git help`

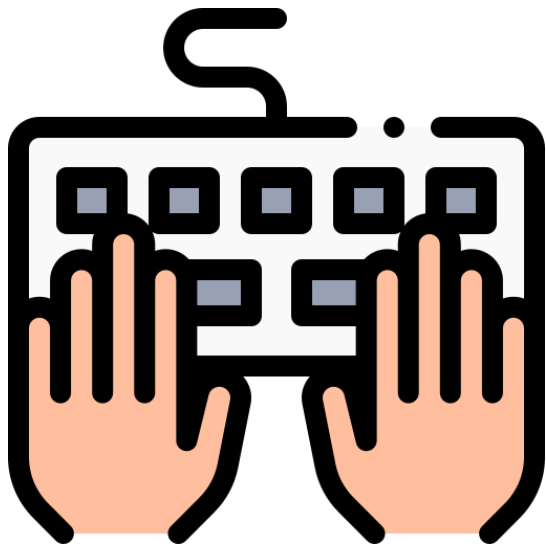


Inicialización del proyecto

- Para que un proyecto utilice Git, se debe ejecutar en la raíz del proyecto el comando: `git init`
- Este comando crea un subdirectorio llamado `.git`, que contiene todos los archivos necesarios para el repositorio.
- Esto significa que aún tu proyecto no está bajo seguimiento, pero ha sido inicializado con la estructura de un proyecto Git.



Inicialización del proyecto

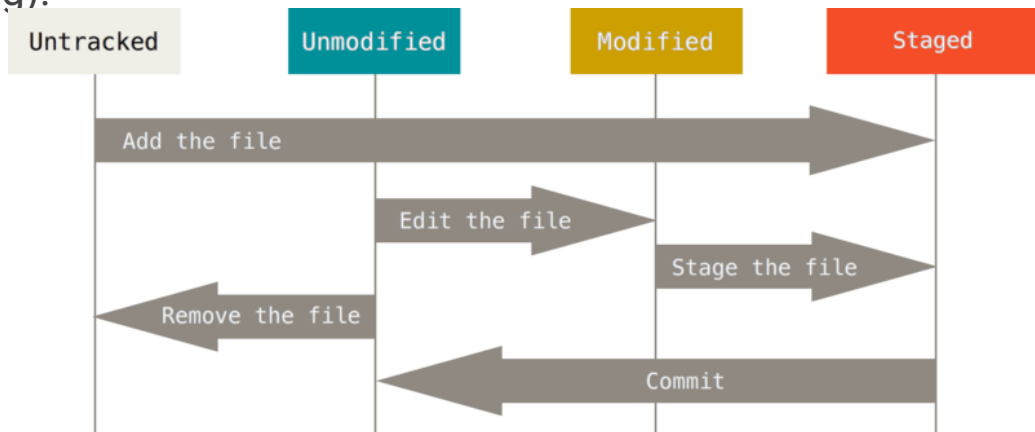


- Para agregar los archivos de un proyecto al control de versiones hay que darle seguimiento y hacer una confirmación inicial.
- La secuencia de comandos es:
`$ git add .`
`$ git commit -m "versión inicial del proyecto"`
- En este momento tienes un repositorio Git con archivos versionados y un commit inicial.

Estados

En el repositorio cada archivo puede tener 2 estados: rastreados (tracked) y sin rastrear (untracked).

Los rastreados son aquellos archivos que estaban presentes en la última instantánea del proyecto y pueden estar en estados sin modificar (unmodified), modificados (modified) o en preparación (staging).



Revisando el estado de tus archivos

- Git incorpora la herramienta `git status` para mostrar el estado de cada archivo.
- Si ejecutas el comando desde la raíz del proyecto, podrás ver información sobre la rama (branch) actual y los archivos en sus distintos estados.

```
$ git status
```

```
On branch master
```

```
nothing to commit, working directory clean
```

- El mensaje “working directory clean” significa que el directorio de trabajo está limpio, o sea, los archivos rastreados no han sido modificados y no se encuentran archivos no rastreados.

Primer proyecto Git



Actividad

Preparación

- ❑ Crear una nueva carpeta.
- ❑ Entrar a la carpeta y crear el archivo README.md vacío.
- ❑ Abrir la consola o terminal en la ruta generada.

Flujo Git

- ➔ `git init` para inicializar el proyecto Git
- ➔ `git add` para agregar todo el contenido de la carpeta.
- ➔ `git commit -m "primera versión de README"` para confirmar la versión.

```
$ git commit -m "primera version de README"
[master (root-commit) e6132da] primera version de README
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 README.md
```

Add

Agregar los cambios

- Para comenzar a rastrear un archivo se debe usar el comando `git add <NOMBRE>`
- Con este comando se agrega el archivo de nombre `<NOMBRE>` al área de preparación (stage).
- En el caso de utilizar `."` como nombre, se refiere a la carpeta actual donde se ejecuta el comando `$ git add`
- Si revisamos nuevamente el estado del proyecto, el archivo aparece como rastreado y está preparado para ser confirmado.

```
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   README
```

Commit

Confirmar los cambios

- Con el área de preparación lista, confirmamos los cambios.
- Cualquier archivo que no esté en el área de preparación no será confirmado.
- Por ejemplo, si modificas un archivo y no ejecutas git add, el archivo no será incluido en el commit.

```
git commit -m "mensaje descriptivo de los cambios incluidos"
```

Al ejecutar este comando has creado un commit que entrega información de salida con el nombre de la rama actual, el checksum del commit, cuántos archivos cambiaron y cantidad de líneas agregadas y eliminadas.

```
$ git commit -m "primera version de README"
[master (root-commit) e6132da] primera version de
README
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 README.md
```

Workflow



/* Revisemos algunos Comandos */

Cambiar el nombre de los archivos

Para cambiar el nombre de un archivo usando el comando `git mv <NOMBRE_ANTIGUO> <NOMBRE_NUEVO>`
Luego de ejecutar cualquier cambio, es una buena práctica verificar el resultado ejecutando el comando `git status`.

```
$ git mv README.md README.MD
$ git status

On branch master

Changes to be committed:

  (use "git restore --staged <file>..." to
  unstage)

        renamed:    README.md -> README.MD
```

El nombre del archivo ha sido modificado y queda en el área de preparación.

Para confirmar el cambio debemos ejecutar el comando `commit` indicando un mensaje descriptivo:

```
git commit -m "extensión de README
actualizada"
```


Eliminar archivos

Para eliminar archivos de Git debes eliminarlos de los archivos rastreados y luego confirmar. Para esto se ocupa el comando `git rm` que además elimina el archivo del espacio de trabajo, para que no aparezca la próxima vez como un archivo no rastreado.

```
$ git rm README.MD
rm 'README.MD'
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    deleted:    README.MD
```

El archivo ha sido eliminado y queda en el área de preparación.

Para confirmar la eliminación debemos ejecutar el comando `commit` indicando un mensaje descriptivo:

```
git commit -m "eliminado el archivo
README.MD"
```

Restaurar archivos

Sin confirmar

```
$ git rm README.MD
```

```
rm 'README.MD'
```

```
$ git status
```

```
On branch master
```

```
Changes to be committed:
```

```
  (use "git reset HEAD  
<file>..." to unstage)
```

```
    deleted:    README.MD
```

Luego de eliminar el archivo README.MD y antes de confirmar los cambios, se puede deshacer el cambio utilizando el comando:

```
git checkout HEAD <NOMBRE>
```

El comando `git checkout` permite acceder a una versión en particular. El argumento HEAD se refiere a la última versión guardada y <NOMBRE> corresponde al nombre del archivo, en este caso, sería README.MD

```
git checkout HEAD README.MD
```

Restaurar archivos

Confirmado

Si los cambios ya fueron confirmados usando el comando `git commit`, los archivos se encuentran en estado “no modificados” (unmodified).

```
$ git rm README.MD  
  
rm 'README.MD'  
  
$ git commit -m "eliminado el archivo  
porque está duplicado"
```

El comando `git reset` asigna el HEAD al estado especificado, y el argumento `HEAD~1` indica el estado penúltimo estado.

En la salida del comando indica cuál commit es el nuevo HEAD del proyecto.

```
$ git reset --hard HEAD~1  
HEAD is now at efff04b agregado  
projects
```

Ignorar archivos

.gitignore

- Todos los proyectos tienen archivos que se generan automáticamente y que no queremos agregar a Git.
- El archivo `.gitignore` que se encuentra en la raíz del proyecto permite indicar los archivos que queremos ignorar.
- Se pueden definir los archivos que queremos ignorar indicando, por ejemplo:

`*.[oa]` : ignora todos los archivos que terminen en “.o” o “.a”

`/build` : ignora la carpeta build y todo su contenido

`.DS_Store` : ignora el archivo de nombre .DS_Store

```
$ cat .gitignore  
  
*.[oa]  
  
build/  
  
.DS_Store
```

Resumen de comandos básicos

git <COMANDO>	Características
init	<ul style="list-style-type: none">● Inicializa el proyecto Git.● Crear la carpeta .git dentro de la raíz del proyecto.● Solo se utiliza la primera vez.
add <NOMBRE>	<ul style="list-style-type: none">● Agrega el archivo de nombre <NOMBRE> al área de preparación (stage).
commit -m “<MENSAJE>”	<ul style="list-style-type: none">● Confirma los cambios, guardando los archivos que están en el área de preparación (stage) en un commit que tiene un identificador único (checksum), un mensaje, el autor de los cambios y los archivos modificados con sus estadísticas.

Resumen de comandos básicos

git <COMANDO>	Características
status	Muestra las rutas de los archivos que tienen diferencias con el HEAD actual y rutas de archivos no rastreados (que no están siendo ignoradas por gitignore). Los primeros se confirman con <code>git commit</code> , y los otros agregandolos con <code>git add</code> antes de ejecutar <code>git commit</code> .
checkout <NOMBRE>	Sirve para restaurar un commit o cambiarse de ramas indicando el nombre <NOMBRE> de la rama.
rm <NOMBRE>	Elimina el archivo de nombre <NOMBRE> del control de versiones.
help	Documentación del uso de los comandos git disponible en forma local.
config	Es la herramienta que permite obtener y asignar variables de configuración guardadas en el archivo <code>.gitconfig</code>

¿Qué ocurre si no se hace git add a un archivo modificado?





Próxima sesión...

Creación, manejo y mezcla de ramas.

{desafío}
latam_

*Academia de
talentos digitales*

