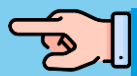
Fundamentos de GIT y GitHub

Fundamentos de GIT (Parte II)

{desafío} latam_

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

{desafío} latam_

¿Qué aprenderás en esta sesión?

Creación, manejo y mezcla de ramas.

{desafío} latam_

Proyecto 1

{desafío} latam_

Activación de conceptos

Proyecto 1

Crearemos un proyecto y lo utilizaremos como base practicando nuestros conocimientos de Git.

Creación del proyecto inicial:

- ❏ Crear una carpeta llamada proyecto1
- ❏ Dentro de la carpeta proyecto1 crear el archivo README.md con el siguiente contenido:

```
# Proyecto1
Proyecto1 es un proyecto de versionamiento con Git

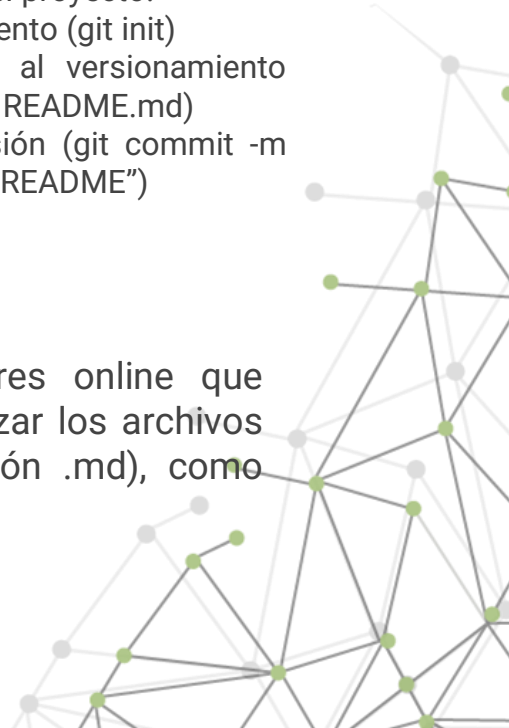
## Utilización
El flujo básico de para versionar archivos con Git incluye inicializar el
proyecto usando
```bash
git init
```

y luego agregando los archivos al área de preparación y confirmar la
versión con lo comandos
```bash
git add .
git commit -m "mensaje"
```
```

Versionamiento inicial del proyecto:

- ❏ Iniciar versionamiento (git init)
- ❏ Agregar archivos al versionamiento (git add . / git add README.md)
- ❏ Confirmar la versión (git commit -m “versión inicial de README”)

Tip: Existen editores online que permiten previsualizar los archivos Markdown (extensión .md), como dillinger.io.

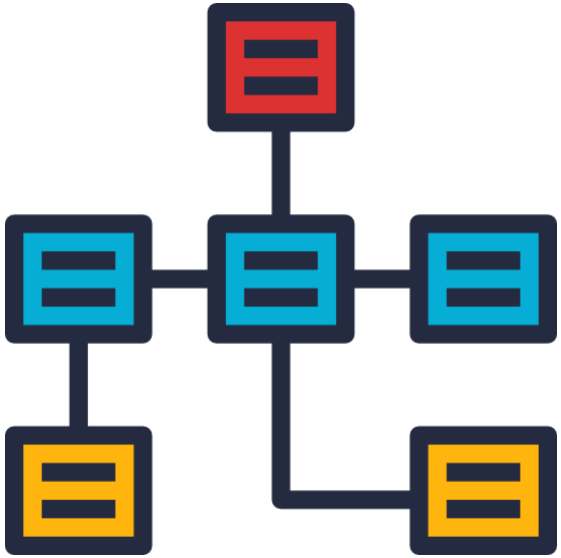


/* Ramas (branches) */

Ramas

¿Qué son?

- Las ramas son una división del estado del código.
- Permite crear nuevos caminos a favor de la evolución del código.
- En Git, las ramas son parte diaria del desarrollo, son una guía instantánea para los cambios realizados.
- Por ejemplo, quieres arreglar un error, por lo que generas una nueva rama donde están los cambios que realizaste, al realizar esta acción va a resultar más complicado que algún error o fallo del código inestable se incorpore al código base principal, dando la oportunidad de limpiar tu historial antes de fusionarlo todo con la rama principal, mejorando tu eficiencia de trabajo.



Crear una rama

Paso a paso

Cuando queremos agregar una nueva funcionalidad a nuestro proyecto, la forma adecuada de hacerlo es crear una rama de la nueva funcionalidad, por ejemplo, aplicar un descuento:

- Para crear una rama:

```
git branch aplicar_descuento
```

- Para posicionarse en la rama:

```
git checkout aplicar_descuento
```

Otra alternativa es utilizar el comando `git checkout` con el argumento -b que permite crear la rama aplicar_descuento y posicionarse en ella:

```
git checkout -b aplicar_descuento
```

Actividad

Actividad

Usando el proyecto creado en la clase anterior:

Paso	Acción	Comando
1	Crear y cambiarse a branch develop.	git checkout -b develop
2	Agregar el apartado de licencia y el link al final del archivo README.md	## Licencia [MIT](https://choosealicense.com/licenses/mit/)
3	Verificar el estado de los archivos.	git status
4	Agregar al área de presentación (stage).	git add
5	Confirmar los cambios.	git commit -m “Se agrega el tipo de licencia”.
6	Posicionarse en la rama master.	git checkout master.

¿Cuál es el contenido del archivo en la rama master?

¿Existe el link que agregamos al archivo README.md?

Análisis

git log

Usando el comando git log en ambas ramas podemos ver la historia de los commits para cada una de ellas, y podemos utilizar git branch para listar las ramas disponibles y saber la actual.

```
$ git branch
  develop
* master

$ git log
commit 725a1f7a2c8ec058164e6f8ce0ac8c9819240035 (HEAD ->
master)
Author: Awesome developer <Awesome@developer.com>
Date:   Thu Nov 10 12:25:08 2022 -0300

versión oficial de README
(END)
```

```
$ git branch
* develop
  master

$ git log
commit 54892304cc8dcffc56fd915bad546ee5bb093ef5 (HEAD -> develop)
Author: Awesome developer <Awesome@developer.com>
Date:   Thu Nov 10 12:26:09 2022 -0300

    Se agrega el tipo de licencia

commit 725a1f7a2c8ec058164e6f8ce0ac8c9819240035 (master)
Author: Awesome developer <Awesome@developer.com>
Date:   Thu Nov 10 12:25:08 2022 -0300

  versión oficial de README
(END)
```

Análisis

Al momento de realizar el commit en develop y luego volver a master, nuestra rama develop se encuentra un commit delante de este Por lo tanto, el link que agregamos y que está en el último commit de develop no se encuentra en master.

Tenemos 2 opciones:

1. Descartar los cambios, por ejemplo, eliminando la rama develop.

```
$ git branch -D develop
Deleted branch develop (was 5489230.
```

1. Integrar los cambios de la rama develop en la rama master.

```
$  git merge develop
Updating 725a1f7..5489230
Fast-forward
 README.md | 5 +++++
 1 file changed, 5 insertions(+)
```

/* Uniones */

Unión (merge)

¿Qué es?

```
$ git status
On branch master
nothing to commit, working tree clean

$ git merge develop
Updating 725a1f7..5489230
Fast-forward
 README.md | 5 +++++
 1 file changed, 1 insertion(+)
```

- Un merge (o unión) es el proceso que unifica el trabajo hecho en 2 ramas diferentes.
- En el ejemplo, se utiliza la estrategia de unión Fast-forward.

Estrategia de unión (merge)

Fast-forward

Es la estrategía de unión por defecto, cuando la rama que se va a integrar es descendiente directa de nuestra rama principal.

Este es nuestro caso, donde el último commit de la rama master es ancestro directo del commit en la rama develop.

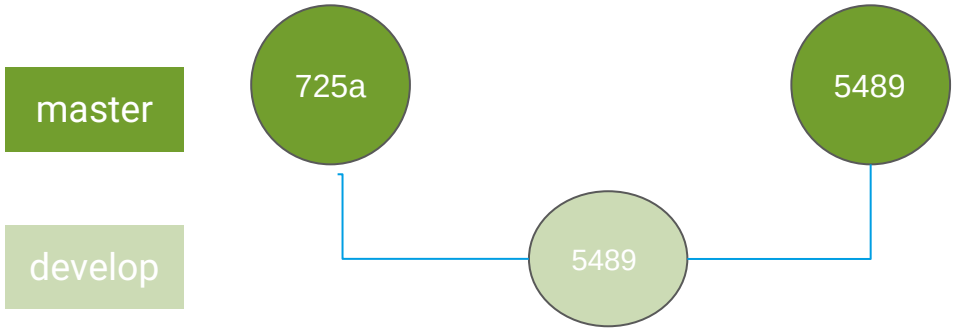
```
$ git log --graph

* commit 54892304cc8dcffc56fd915bad546ee5bb093ef5
(HEAD -> master, develop)
| Author: Awesome developer
<Awesome@developer.com>
| Date:   Thu Nov 10 12:26:09 2022 -0300
|
|     Se agrega el tipo de licencia
|
* commit 725a1f7a2c8ec058164e6f8ce0ac8c9819240035
  Author: Awesome developer
<Awesome@developer.com>
  Date:   Thu Nov 10 12:25:08 2022 -0300

      versión oficial de README
(END)
```

Estrategia de unión (merge)

Fast-forward



La rama secundaria develop es descendiente directa de la rama principal y no hay inconsistencias ni objeciones en la unión.

Estrategia de unión (merge)

Recursive

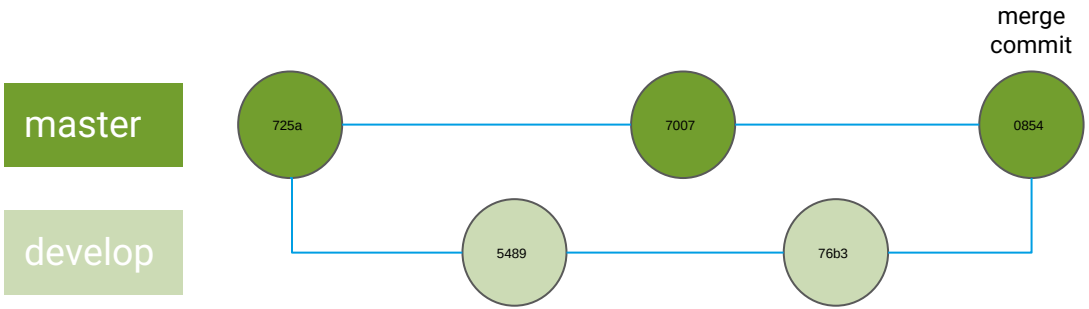
Cuando la rama principal deja de ser ancestro directo de la rama secundaria, se utiliza la estrategia *recursive,* que genera un nuevo commit indicando los 2 ancestros comunes, en de la rama principal y la rama secundaria.

Este commit recibe el nombre de ***merge commit***.

```
Auto-merging README.md
Merge made by the 'recursive'
strategy.
 README.md | 1 +
 1 file changed, 1 insertion(+)
```

Estrategia de unión

Recursive



- La rama secundaria develop NO es descendiente directa de la rama principal y no hay inconsistencias ni objeciones en la unión.
- Para hacer el merge se crea un *merge commit*.

Demostración

Estrategia de merge - Recursive

Paso	Acción	Comando
1	Posicionado en la rama master, actualizar el nombre del proyecto en la primera línea del archivo README.md.	## Proyecto Git
2	Confirmar los cambios	git add . git commit -m “Se actualiza el nombre del proyecto”
3	Cambiarse a la branch develop	git checkout develop
4	Agregar el apartado de “Comandos útiles” al final del archivo README.md	## Comandos útiles.
5	Verificar el estado de los archivos y guardar los cambios	git status git add . git commit -m se agrega apartado de comandos útiles
6	Volver a la rama master	git checkout master
7	Mezclar la rama develop en la rama master	git merge develop

Estrategia de merge - Recursive

Mensaje de confirmación

```
Merge branch 'develop'
# Please enter a commit message to explain why
this merge is necessary,
# especially if it merges an updated upstream into
a topic branch.
#
# Lines starting with '#' will be ignored, and an
empty message aborts
# the commit.
```

- Se abre el editor de textos por defecto y permite indicar el mensaje que se agregará a la información del *merge commit*.
- Este mensaje se puede proporcionar usando el argumento -m con el mensaje, igual que al crear un commit.

Estrategia de unión - Recursive

git log --graph

Podemos ver que en la historia de la branch master se ha creado un merge commit y que proporciona información para identificar unívocamente cada confirmación o commit.

```
$ git log --graph

*   commit 0854cc599ec1922894eadab01b422ef250578d8c (HEAD ->
master)
|\  Merge: 7d07e50 76b3de0
| | Author: Awesome developer <Awesome@developer.com>
| | Date:   Thu Nov 10 13:46:52 2022 -0300
| |
| |     Merge branch 'develop'
| |
| * commit 76b3de00251a28a39def628917f6e3c83ac57021 (develop)
| | Author: Awesome developer <Awesome@developer.com>
| | Date:   Thu Nov 10 13:40:35 2022 -0300
| |
| |     se agrega apartado de comandos útiles
| |
* | commit 7d07e50de74958e6b3029e7e7305b6aa6dcecff5
|/  Author: Awesome developer <Awesome@developer.com>
|   Date:   Thu Nov 10 13:35:35 2022 -0300
|
|       se actualiza el nombre del proyecto
|
* commit 54892304cc8dcffc56fd915bad546ee5bb093ef5
| Author: Awesome developer <Awesome@developer.com>
| Date:   Thu Nov 10 12:26:09 2022 -0300
|
|     Se agrega el tipo de licencia
|
* commit 725a1f7a2c8ec058164e6f8ce0ac8c9819240035
  Author: Awesome developer <Awesome@developer.com>
  Date:   Thu Nov 10 12:25:08 2022 -0300

versión oficial de README
(END)
```

{desafío} latam_

/* Conflictos */

Estrategia de unión

Paso	Acción	Comando
1	Posicionado en la rama master, agregar el comando “git status” y su descripción al archivo README.md	```bash git 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).
2	Confirmar los cambios	git add . git commit -m “Se agrega comando git status”
3	Cambiarse a la branch develop	git checkout develop

Estrategia de unión

Paso	Acción	Comando
4	Agregar el comando “git config” y su descripción al archivo README.md	```bash git config ``` Es la herramienta que permite obtener y asignar variables de configuración guardadas en el archivo .gitconfig
5	Verificar el estado de los archivos	git status
6	Confirmar los cambios	git add . git commit -m “Se agrega el comando git config”
7	Volver a la rama master	git checkout master
8	Hacer el merge	git merge develop

Análisis

Conflictos en la unión

¿Qué ocurre al modificar la misma línea de un archivo en dos ramas distintas?

```
$ git merge develop
Auto-merging README.md
CONFLICT (content): Merge conflict in
README.md
Automatic merge failed; fix conflicts and
then commit the result.
```

Si hay modificaciones dispares en una misma porción de un mismo archivo en 2 ramas distintas que se están mezclando, Git no podrá mezclarlas directamente y hará una pausa en el proceso esperando que tú resuelvas el conflicto.

Resolución de conflictos

Verificando el estado del merge

Todos los archivos con conflicto se marcan como “unmerged”.

```
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)
        both modified:   README.md
```

Se espera que los archivos en estado unmerged sean editados manualmente para resolver los conflictos.

Resolución de conflictos

¿Cómo resolvemos un conflicto?

Git agrega unos marcadores especiales para demarcar el conflicto, donde nos dice que la versión en HEAD contiene lo indicado en el bloque superior y que la versión develop tiene lo indicado en el bloque inferior, donde ambos bloques son separados por el marcador `=======`

```
## Comandos útiles
<<<<<<< HEAD
```bash
git 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).
=======
```bash
git config
```
Es la herramienta que permite obtener y asignar
variables de configuración guardadas en el archivo
.gitconfig
>>>>>>> develop
```

Resolución de conflictos

Supongamos que queremos mantener los 2 bloques que contienen los cambios, por lo que debemos eliminar completamente las líneas con los marcadores <<<<<<< , =======, >>>>>>> y mantener los 2 comandos agregados.

````
## Comandos útiles
```bash
git 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).

```bash
git config
```
Es la herramienta que permite obtener y asignar variables de
configuración guardadas en el archivo .gitconfig
````

Resolución de conflictos

Confirmando la resolución de los conflictos

Luego de resolver todos los conflictos, el archivo queda en el área de preparación (stage):

```
$ git add .
$ git status
On branch master
All conflicts fixed but you are still merging.
  (use "git commit" to conclude merge)

Changes to be committed:
        modified:   README.md
```

Para su posterior confirmación:

```
$ git commit -m "resolución de conflicto para la sección de comandos útiles"
[master af2ed8b] resolución de conflicto para la sección de comandos útiles
```

Resolución de conflictos

Verificando el historial

Al verificar el log encontramos el merge commit indicando los ancestros involucrados.

```
$ git log -- graph

*   commit af2ed8b654d19a7adf918eba0812c6451be83b01 (HEAD ->
master)
|\  Merge: ff05e5d ee5f30e
| | Author: Awesome developer <Awesome@developer.com>
| | Date:   Thu Nov 10 15:44:28 2022 -0300
| |
| |     resolución de conflicto para la sección de comandos
útiles
| |
| * commit ee5f30eb3312f3abaf0073a151efde7b7c5c5201 (develop)
| | Author: Awesome developer <Awesome@developer.com>
| | Date:   Thu Nov 10 15:33:28 2022 -0300
| |
| |     se agrega el comando git config
| |
* | commit ff05e5d9a65e8768c5fb6998462fb77dca56c00d
| | Author: Awesome developer <Awesome@developer.com>
| | Date:   Thu Nov 10 15:32:14 2022 -0300
| |
| |     Se agrega comando git status
| |
* | commit 0854cc599ec1922894eadab01b422ef250578d8c
|\| Merge: 7d07e50 76b3de0
| | Author: Awesome developer <Awesome@developer.com>
| | Date:   Thu Nov 10 13:46:52 2022 -0300
| |
| |     Merge branch 'develop'
```

Tags

Etiqueta

- Una etiqueta (*tag*) es una referencia que apunta a un commit concreto en el historial de Git y se usa generalmente para la publicación de una versión, por ejemplo, V2.0.3.
- En palabras simples, una etiqueta es una rama que no cambia.
- Para su creación se usa el comando `git tag`.

```
$ git tag <NOMBRE_DEL_TAG>
```

- Donde el <NOMBRE_DEL_TAG> debe ser reemplazado con un identificador semántico del estado del repositorio en el momento de creación del *tag*.

```
$ git tag v1.0
$ git tag
v1.0
```

¿Cómo podemos evitar los conflictos en la unión de ramas?

Próxima sesión...

Git y GitHub para versionar proyectos

{desafío}
latam_

{desafío}
latam_

Academia de
talentos digitales