

Taller de Git





tp_final.tex



tp_final_2.tex



tp_final_
este_va.tex



tp_final_
posta.tex



tp_final_
reentrega.tex



Trabajando en grupo

- Enviar cambios por mail, o
- Sincronizar cambios por Dropbox, o
- Sincronizar cambios por Google Docs.

Trabajando en grupo

- Enviar cambios por mail, o
- Sincronizar cambios por Dropbox, o
- Sincronizar cambios por Google Docs.



¿Qué es un Sistema de Control de Versiones?

Los **Sistemas de Control de Versiones** son programas que permiten **manejar los cambios** en el código fuente de un proyecto a lo largo del tiempo.

Llevan un **seguimiento** de las modificaciones que hacemos, y en caso de que nos equivoquemos, es posible volver atrás y comparar el código actual con versiones anteriores para ayudar a arreglar el error.

También permiten que distintas personas modifiquen el código a la vez y **compartan los cambios**, tratando de prevenir conflictos, y en caso de que los hubiera, ayudando a identificarlos y resolverlos.

¿Qué es un Sistema de Control de Versiones?

Los **Sistemas de Control de Versiones** son programas que permiten **manejar los cambios** en el código fuente de un proyecto a lo largo del tiempo.

Llevan un **seguimiento** de las modificaciones que hacemos, y en caso de que nos equivoquemos, es posible volver atrás y comparar el código actual con versiones anteriores para ayudar a arreglar el error.

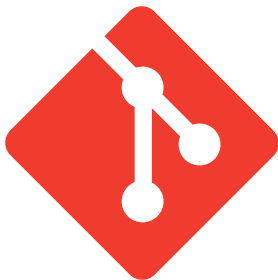
También permiten que distintas personas modifiquen el código a la vez y **compartan los cambios**, tratando de prevenir conflictos, y en caso de que los hubiera, ayudando a identificarlos y resolverlos.

Es decir, permiten...

- Arreglar *accidentes* y volver a versiones anteriores del código.
- Compartir código con otras personas.

¿Qué es Git?

Git es un Sistema de Control de Versiones **distribuido y de código abierto**. Además fue diseñado con énfasis en la **performance** (para manejar proyectos muy grandes), **seguridad** y **flexibilidad**. Provee un amplio conjunto de comandos que permiten realizar operaciones de alto y bajo nivel.



Tu identidad

Es importante establecer nuestro **nombre y email**, ya que estos van a ir asociados con los cambios que hagamos:

```
git config --global user.name "Guybrush Threepwood"  
git config --global user.email guybrush@example.com
```


Configuraciones iniciales

Clave SSH

Para poder trabajar cómodamente con repositorios Git que estén en Internet (GitHub, Bitbucket, GitLab, etc.), podemos configurar una **clave SSH** que nos identifique con el servidor que estemos usando.

Configuraciones iniciales

Clave SSH

Para poder trabajar cómodamente con repositorios Git que estén en Internet (GitHub, Bitbucket, GitLab, etc.), podemos configurar una **clave SSH** que nos identifique con el servidor que estemos usando.

Creando una clave nueva

Abrimos la terminal y ejecutamos `ssh-keygen` . Le damos *Enter* a todo.

Configuraciones iniciales

Clave SSH

Para poder trabajar cómodamente con repositorios Git que estén en Internet (GitHub, Bitbucket, GitLab, etc.), podemos configurar una **clave SSH** que nos identifique con el servidor que estemos usando.

Creando una clave nueva

Abrimos la terminal y ejecutamos `ssh-keygen`. Le damos *Enter* a todo.

Subiendo la clave a GitLab

- En la terminal, ejecutamos `cd ~/.ssh`, y luego `cat id_rsa.pub`. Copiamos todo lo que aparezca.
- Abrimos GitLab, vamos a Profile Settings, SSH Keys.
- Pegamos lo que habíamos copiado en el campo *Key*, y le ponemos un *Title*, como por ejemplo *Notebook del abuelo*.
- Le damos al boton *Add key*.

Obteniendo un repositorio Git

git clone

Obteniendo un repositorio Git

git clone

Para obtener una copia local de un repositorio existente en algún servidor, utilizamos el comando `git clone [URL]` (sin los corchetes).

Obteniendo un repositorio Git

git clone

Para obtener una copia local de un repositorio existente en algún servidor, utilizamos el comando `git clone [URL]` (sin los corchetes).

Ejercicio

Clonar el repositorio que tiene URL:

`git@gitlab.com:talleres-comcom/taller-git-ejercicio.git`

```
git add
```

git add

Una vez que tenemos cambios hechos, tenemos que marcarlos como preparados antes de confirmarlos. En la jerga de Git, decimos que pasamos los cambios a *staged*.

- 1 Creamos/modificamos el archivo en cuestión.
- 2 Ejecutamos `git add [nombre del archivo]`.

git add

Una vez que tenemos cambios hechos, tenemos que marcarlos como preparados antes de confirmarlos. En la jerga de Git, decimos que pasamos los cambios a *staged*.

- 1 Creamos/modificamos el archivo en cuestión.
- 2 Ejecutamos `git add [nombre del archivo]`.

Ejercicio

Adentro del repositorio que *clonaron* recién, crear un archivo y marcarlo como *staged* usando el comando `git add`.

```
git commit
```

git commit

Una vez que tenemos ciertos cambios en *staged*, podemos confirmarlos ejecutando `git commit -m [mensaje]`.

Donde [mensaje] es una breve descripción de los cambios que acabamos de confirmar.

git commit

Una vez que tenemos ciertos cambios en *staged*, podemos confirmarlos ejecutando `git commit -m [mensaje]`.

Donde [mensaje] es una breve descripción de los cambios que acabamos de confirmar.

Ejercicio

Confirmar los cambios que pasaron a *staged* en la diapo anterior.

¡No seas vago con los mensajes!



	COMMENT	DATE
○	CREATED MAIN LOOP & TIMING CONTROL	14 HOURS AGO
○	ENABLED CONFIG FILE PARSING	9 HOURS AGO
○	MISC BUGFIXES	5 HOURS AGO
○	CODE ADDITIONS/EDITS	4 HOURS AGO
○	MORE CODE	4 HOURS AGO
○	HERE HAVE CODE	4 HOURS AGO
○	AAAAA	3 HOURS AGO
○	ADKFJSLKDFJSDKLFJ	3 HOURS AGO
○	MY HANDS ARE TYPING WORDS	2 HOURS AGO
○	HAAAAAAAAAANDS	2 HOURS AGO

AS A PROJECT DRAGS ON, MY GIT COMMIT MESSAGES GET LESS AND LESS INFORMATIVE.

Fuente: <https://xkcd.com/1296/>

¿Está preparado, confirmado o ninguna de las dos?

git status

¡No es lo mismo!

Las modificaciones que hacemos pueden estar en **4 estados** distintos:

¿Está preparado, confirmado o ninguna de las dos?

git status

¡No es lo mismo!

Las modificaciones que hacemos pueden estar en **4 estados** distintos:

- **Sin seguimiento (untracked)**: archivos que nunca fueron agregados al repositorio, por ejemplo archivos nuevos.

Output de ejemplo

```
Untracked files:
  README
```

¿Está preparado, confirmado o ninguna de las dos?

git status

¡No es lo mismo!

Las modificaciones que hacemos pueden estar en **4 estados** distintos:

- **Modificado (modified)**: las modificaciones todavía no están marcadas como *staged*.

Output de ejemplo

```
Changes not staged for commit:
  modified:   README
```


¿Está preparado, confirmado o ninguna de las dos?

git status

¡No es lo mismo!

Las modificaciones que hacemos pueden estar en **4 estados** distintos:

- **Preparado (staged)**: las modificaciones están en *staged* e irán en la próxima *confirmación de cambios (commit)*.

Output de ejemplo

```
Changes to be committed:  
  new file:   README
```

¿Está preparado, confirmado o ninguna de las dos?

git status

¡No es lo mismo!

Las modificaciones que hacemos pueden estar en **4 estados** distintos:

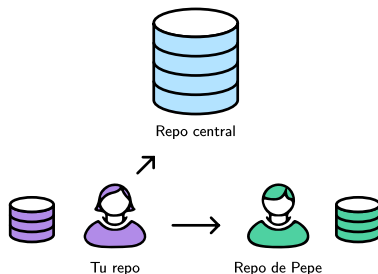
- **Confirmado (committed)**: las modificaciones están guardadas con un *mensaje* que explica los cambios realizados.

Output de ejemplo

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

Colaborando con otras personas

Los repositorios remotos son *copias* de nuestro proyecto a las cuales accedemos a través de Internet. Puede haber varios, cada uno de los cuales puede ser de solo lectura o de lectura/escritura, según los permisos que tengamos.



Colaborar con otros implica gestionar estos repositorios remotos, y mandar (**push**) y recibir (**pull**) datos de ellos cuando necesites compartir cambios.

```
git push
```

git push

Para enviar los cambios **desde nuestro repositorio local a algún repositorio remoto**, ejecutamos: `git push [remoto] [branch]`.

Por ahora, hasta la siguiente clase, vamos a usarlo como:
`git push origin master`.

```
git pull
```



git pull

Para traer cambios **desde un repositorio remoto a nuestro repositorio local**, ejecutamos: `git pull [remoto] [branch]`.

Igual que antes, hasta la siguiente clase, vamos a usarlo como:
`git pull origin master`.




Y qué pasa si... ¡BOOM!

A veces hay conflictos

- Supongamos que dos personas ( y ) están trabajando en un mismo proyecto. Es decir, ambos tienen una *copia* en su máquina.





Y qué pasa si... ¡BOOM!

A veces hay conflictos

- Supongamos que dos personas ( y ) están trabajando en un mismo proyecto. Es decir, ambos tienen una *copia* en su máquina.
- Ahora imaginemos, por ejemplo, que  modifica la línea 23 del archivo README y confirma los cambios.





Y qué pasa si... ¡BOOM!

A veces hay conflictos

- Supongamos que dos personas ( y ) están trabajando en un mismo proyecto. Es decir, ambos tienen una *copia* en su máquina.
- Ahora imaginemos, por ejemplo, que  modifica la línea 23 del archivo README y confirma los cambios.
- Sin saberlo,  también modifica la línea 23 del archivo README, pero pone algo distinto y confirma dichos cambios.





Y qué pasa si... ¡BOOM!

A veces hay conflictos

- Supongamos que dos personas ( y ) están trabajando en un mismo proyecto. Es decir, ambos tienen una *copia* en su máquina.
- Ahora imaginemos, por ejemplo, que  modifica la línea 23 del archivo README y confirma los cambios.
- Sin saberlo,  también modifica la línea 23 del archivo README, pero pone algo distinto y confirma dichos cambios.
- ¿Qué va a pasar cuando quieran compartir lo que hicieron?





Y qué pasa si... ¡BOOM!

A veces hay conflictos

- Supongamos que dos personas ( y ) están trabajando en un mismo proyecto. Es decir, ambos tienen una *copia* en su máquina.
- Ahora imaginemos, por ejemplo, que  modifica la línea 23 del archivo README y confirma los cambios.
- Sin saberlo,  también modifica la línea 23 del archivo README, pero pone algo distinto y confirma dichos cambios.
- ¿Qué va a pasar cuando quieran compartir lo que hicieron?
Va a haber un conflicto, ya que dos personas modificaron de forma distinta la misma línea.
- ¿Qué va a hacer Git?

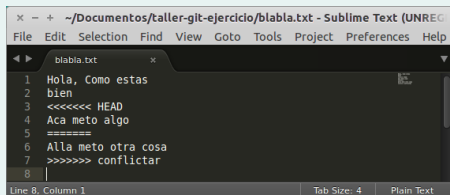
Y qué pasa si... ¡BOOM!

A veces hay conflictos

- Supongamos que dos personas ( y ) están trabajando en un mismo proyecto. Es decir, ambos tienen una *copia* en su máquina.
- Ahora imaginemos, por ejemplo, que  modifica la línea 23 del archivo README y confirma los cambios.
- Sin saberlo,  también modifica la línea 23 del archivo README, pero pone algo distinto y confirma dichos cambios.
- ¿Qué va a pasar cuando quieran compartir lo que hicieron?
Va a haber un conflicto, ya que dos personas modificaron de forma distinta la misma línea.
- ¿Qué va a hacer Git?
Se va a quejar. A alguno de los dos le va a tocar **incorporar a mano los cambios del otro**.

Apagando el incendio

¿Cómo se ve un conflicto?



```
blabla.txt
1 Hola, Como estas
2 bien
3 <<<<<< HEAD
4 Aca meto algo
5 =====
6 Alla meto otra cosa
7 >>>>>> conflictar
8 |
```

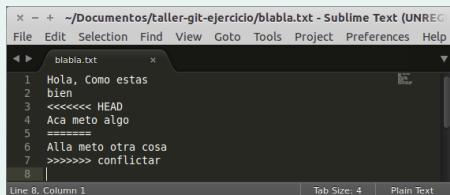
Line 8, Column 1 Tab Size: 4 Plain Text

¿Qué hago?

- Decido cómo tiene que quedar el archivo final

Apagando el incendio

¿Cómo se ve un conflicto?



```
blabla.txt
1 Hola, Como estas
2 bien
3 <<<<<< HEAD
4 Aca meto algo
5 =====
6 Alla meto otra cosa
7 >>>>>> conflictar
8 |
```

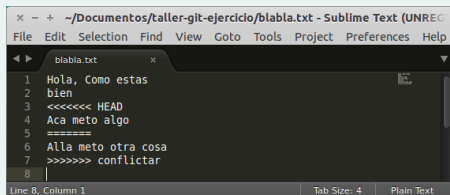
Line 8, Column 1 Tab Size: 4 Plain Text

¿Qué hago?

- Decido cómo tiene que quedar el archivo final
- Hago add

Apagando el incendio

¿Cómo se ve un conflicto?



```
blabla.txt
1 Hola, Como estas
2 bien
3 <<<<<< HEAD
4 Aca meto algo
5 =====
6 Alla meto otra cosa
7 >>>>>> conflictar
8 |
```

Line 8, Column 1 Tab Size: 4 Plain Text

¿Qué hago?

- Decido cómo tiene que quedar el archivo final
- Hago add
- Despues commit normalmente, con un mensaje como 'Merge'

Creando un repositorio vacío

```
git init
```

Creando un repositorio vacío

git init

Crea un repositorio local vacío. Un lienzo en blanco, por así decirlo.

- ➊ Nos paramos en el directorio que queremos convertir en un repositorio.
- ➋ Ejecutamos `git init`.

Esto crea un subdirectorio `.git` que tiene todos los archivos necesarios de Git.

Vinculando un repositorio remoto

git remote

Vinculando un repositorio remoto

git remote

Ver los repositorios remotos asociados

Ejecutamos `git remote -v`.

Vinculando un repositorio remoto

git remote

Ver los repositorios remotos asociados

Ejecutamos `git remote -v`.

Agregar un repositorio remoto

Ejecutamos `git remote add [nombre que queramos] [URL]`.