

El control de versiones con Git

Taller

Denisse Fierro Arcos,
Linda Cabrera Orellana

27 abril, 2022



Nuestro Código de Conducta

Queremos crear un lugar inclusivo y acogedor para que todos puedan aprender. Por lo que te pedimos:

1. Tratar a todos con respeto.
2. No interrumpas a los demás.
3. Póngase en contacto si se siente incómodo.

Antes de iniciar

1. Consulta el sitio web de nuestro taller en <https://lidefi87.github.io/2022-04-26-control-version-Git/>.

2. Completa la encuesta pre-taller

3. Nuestro horario (UTC-5)

17:00 - 18:50 - Lecciones 1 a 7

18:50 - 19:00 - Breve descanso

19:00 - 20:00 - Lecciones 8 a 14

4. Puedes encontrar las lecciones aquí: <https://swcarpentry.github.io/git-novice/>



¿Qué es el control de versiones y por qué se debe usar?

Control de versiones

Control de versiones

¿Qué es el control de versiones?

El control de versiones es un sistema que almacena y organiza los cambios realizados a uno o varios archivos para permitir acceder a distintas versiones de los mismos posteriormente, a controlarlos y a fusionar nuestros archivos.

¿Por qué se debe usar?

- Permite revertir archivos o el proyecto entero a versiones anteriores.
- Permite comparar versiones, ver cambios.
- Permite trabajar de forma colectiva en grupos de desarrollo.
- Permite recuperar los archivos en caso de pérdida o de algún error catastrófico.

Introducción a Git

¿Qué es Git?

Git es un sistema de control de versiones distribuido que se diferencia del resto en el modo en que modela sus datos. Al ser distribuido cada usuario tiene su propio repositorio y los distintos repositorios pueden intercambiar y mezclar revisiones entre ellos.

Las premisas principales del diseño fueron:

- Completamente Distribuido
- Soporte para desarrollos no lineales (infinitas ramas paralelas)
- Seguridad, ya que todas las estructuras internas de datos irán cifradas con el código SHA1 (Secure Hash Algorithm)



Configurando Git

🔗 ¿Cómo me preparo para configurar Git?

Es necesario configurar algunas cosas cuando trabajamos en una computadora por primera vez:

- Nombre y correo electrónico,
- Editor de texto preferido,
- Ajustes globalmente (es decir, para cada proyecto).

Sintaxis de los comandos

Los comandos de Git se escriben como `git verb`, donde `verb` es lo que queremos hacer.

git config

Este comando se utiliza para ver y modificar la configuración de git.

```
$ git config --global user.name "nombre usuario"
$ git config --global user.email "usuario@flisol.loja"

$ git config --list
$ git config -h
$ git config -help
```

Estableciendo el editor de texto

Editor	Comando de configuración
Atom	<code>\$ git config --global core.editor "atom --wait"</code>
nano	<code>\$ git config --global core.editor "nano -w"</code>
TextWrangler (Mac)	<code>\$ git config --global core.editor "edit -w"</code>
Sublime Text (Mac)	<code>\$ git config --global core.editor "subl -n -w"</code>
Sublime Text (Win, 64-bit)	<code>\$ git config --global core.editor "'c:/program files/sublime text 3/sublime_text.exe' -w"</code>
Notepad++ (Win, 64-bit)	<code>\$ git config --global core.editor "'c:/program files/Notepad++/notepad++.exe' -multiInst -notabbar -nosession -noPlugin"</code>
Scratch (Linux)	<code>\$ git config --global core.editor "scratch-text-editor"</code>
emacs	<code>\$ git config --global core.editor "emacs"</code>
vim	<code>\$ git config --global core.editor "vim"</code>
Visual Studio Code	<code>\$ git config --global core.editor "code --wait"</code>



Demo en vivo



Creando un repositorio

Creando un repositorio en Git

git init

Crea un repositorio local en el directorio de trabajo. Básicamente una carpeta oculta llamada `.git` que contiene toda la información que necesita para funcionar.

```
$ cd
$ cd C:/linda/RLadies
$ mkdir flisolLOJA
$ cd flisolLOJA
$ git init
```

¡Mucho Cuidado!

Para eliminar un directorio puedes utilizar el siguiente código:

```
$ rm -rf flisolLoja/.git
$ pwd
```



Demo en vivo



Rastreando cambios

Registrando cambios en Git

git status

Podemos ver en qué estado se encuentran los archivos de nuestro repositorio. Compara los archivos actuales con las versiones del último `commit` e indica:

- Que archivos fueron modificados, creados o eliminados.
- Cuales de esos cambios están agregados para el próximo `commit`.
- Comandos recomendados a utilizar para modificar el estado presentado.

```
$ git status
```

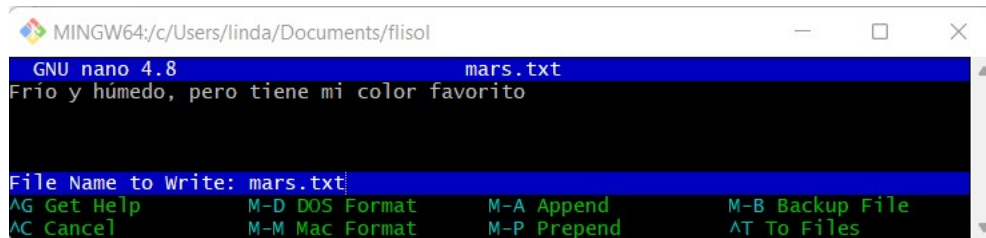
Crear un archivo *.txt* en Nano

1. Crear un archivo llamado `mars.txt`

```
$ nano marte.txt
```

2. Ingresar el texto en el documento

```
Frío y húmedo, pero tiene mi color favorito
```



```
MINGW64:/c:/Users/linda/Documents/flisol
GNU nano 4.8 mars.txt
Frío y húmedo, pero tiene mi color favorito
File Name to Write: mars.txt
AG Get Help  M-D DOS Format  M-A Append  M-B Backup File
AC Cancel    M-M Mac Format   M-P Prepend  AT To Files
```

3. Algunos comandos

Comando	Descripción
Ctrl + O	Guardar
Ctrl + X	Salir del editor y volver a la terminal
Ctrl + G	Ayuda

4. Visualizar texto

```
$ ls
$ cat marte.txt

$ git status
```

Registrando cambios en Git

git add

Este comando permite elegir qué archivos (modificados) queremos agregar a la próxima versión que guardemos en el repositorio.

```
$ git status  
$ git add marte.txt  
$ git status
```

git commit -m "mensaje"

El comando commit crea una instantánea del proyecto con los archivos y cambios agregados a la *"staging area"*, acompañados con un mensaje.

```
$ git commit -m "Inician las notas de Marte"  
$ git status
```

Registrando cambios en Git

git log

Muestra la historia de commits en el repositorio. A partir de diversas opciones permite configurar completamente la información que proporciona, pudiendo mostrar:

- Los cambios realizados a los archivos por `commit`.
- Filtrar `commits` por autor, fecha, etc.
- Graficar la línea temporal de commits.
- Mostrar información reducida o ampliada.
- Combinar las opciones anteriores y más.

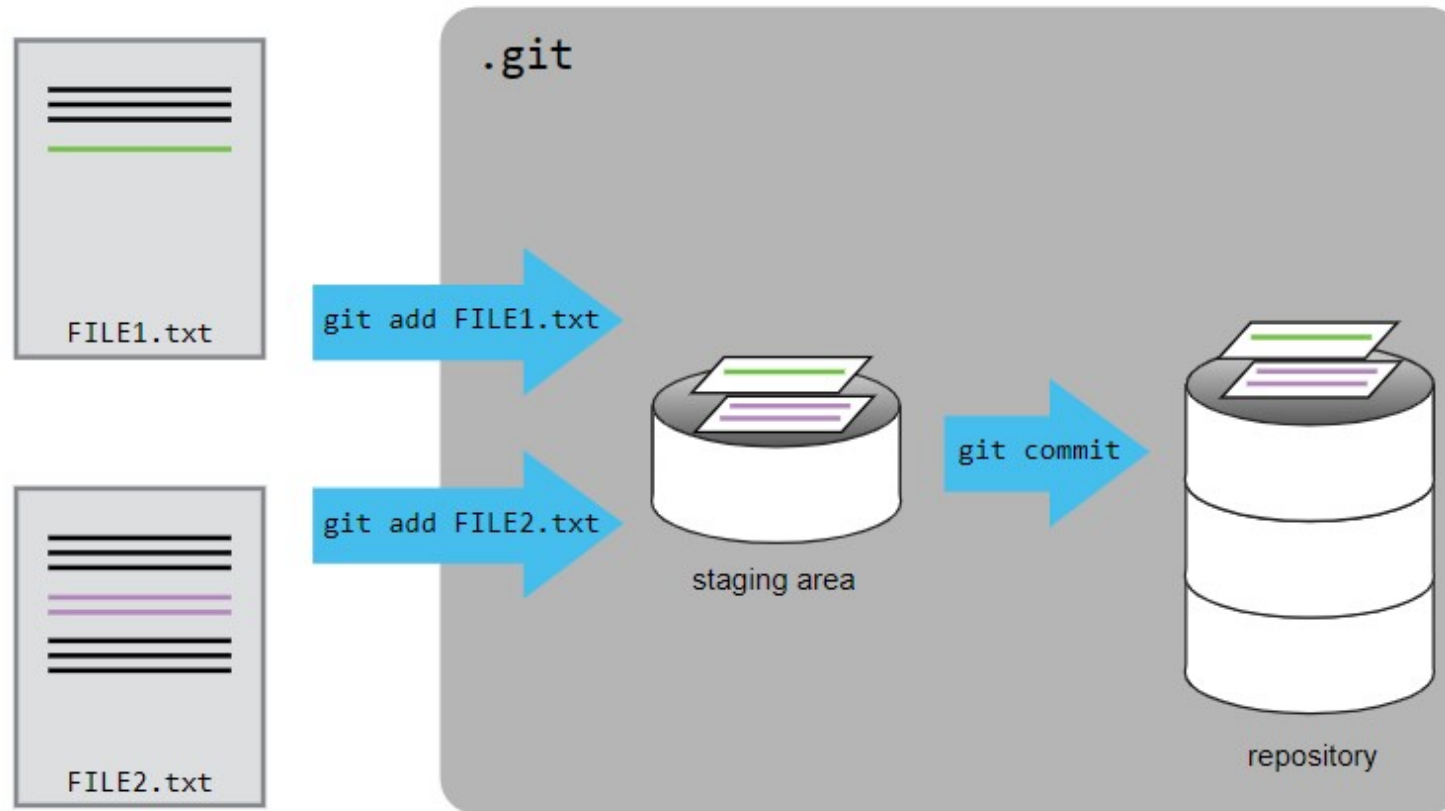
```
$ git log
```

git diff

Al igual que `git commit`, presenta los archivos que cambiaron en el directorio respecto al último `commit`, pero además muestra cuáles fueron esos cambios imprimiendo las dos versiones de los archivos línea por línea.

```
$ git diff
```

🔗 Registrando cambios en Git





Demo en vivo



Explorando el "History"

¿Cómo identifico versiones anteriores de archivos?

 HEAD

 git checkout

El comando permite cambiar el directorio de trabajo al estado de cualquier anterior. Esta acción es segura, ya que es de solo-lectura, se pueden modificar archivos y experimentar, e incluso hacer nuevos commits, pero estos no sobrescribirán ni afectarán a la rama de trabajo.

```
$ git checkout HEAD mars.txt
$ cat mars.txt

$ git checkout f22b25e mars.txt
$ cat mars.txt

$ git checkout -f master mars.txt
```




Demo en vivo



Ignorando cosas

💡 ¿Cómo puedo indicarle a Git que ignore los archivos que no quiero rastrear?

.gitignore

Archivo que contiene listado de archivos que indica a Git que los ignore.

```
$ nano .gitignore
$ cat .gitignore

$ git add .gitignore
$ git commit -m "Add the ignore file"
$ git status
```



Demo en vivo



Repositorios remotos en GitHub

Conectando GitHub y Git

Hay dos maneras para conectar con GitHub

1. Crear un token de acceso personal (PAT)

Instrucciones: <https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/creating-a-personal-access-token>

2. Configurar las claves SSH

Instrucciones: <https://swcarpentry.github.io/git-novice/07-github/index.html>

GitHub ya no permite que los usuarios usen solo contraseñas.

Conectando GitHub y Git

Después de crear una clave PAT o SSH, puede conectar su repositorio local a la nube

Conecta repositorios remotos a tu computadora

```
$ git remote add BRANCH_NAME LINK_GITHUB
```

Enviar (push) cambios a GitHub

```
$ git push REMOTE_BRANCH LOCAL_BRANCH
```

Recibir (pull) cambios de GitHub

```
$ git pull REMOTE_BRANCH LOCAL_BRANCH
```

Copiar un repositorio remoto localmente

Copie (clone) un repositorio de Github a su computadora

```
$ git clone LINK_GITHUB
```

Enviar (push) cambios a GitHub

```
$ git push REMOTE_BRANCH LOCAL_BRANCH
```

Recibir (pull) cambios de GitHub

```
$ git pull REMOTE_BRANCH LOCAL_BRANCH
```


Conectando GitHub y Git

¿Qué pasa si no quiero usar GitHub?

Algunas alternativas disponibles incluyen:

1. GitLab: <https://gitlab.com/>
2. Gitbucket: <https://bitbucket.org/>
3. SourceForge: <https://sourceforge.net/>
4. Gitea: <https://gitea.io/en-us/>

¿Necesito codificar para usar Git y GitHub?

De ningún modo. Hay varias GUI disponibles ahora:

1. El sitio web de Git ofrece una extensa lista de opciones: <https://git-scm.com/downloads/guis>
2. También puede conectarse a GitHub desde:
 - RStudio
 - Código visual
 - Átomo

Algunas referencias extras

1. GIT CHEAT SHEET

2. A Quick Introduction to Version Control with Git and GitHub:
<https://doi.org/10.1371/journal.pcbi.1004668>

3. Ten Simple Rules for Taking Advantage of Git and GitHub:
<https://doi.org/10.1371/journal.pcbi.1004947>

4. When it comes to reproducible science, Git is code for success. And the key to its popularity is the online repository and social network, GitHub: <https://www.natureindex.com/news-blog/when-it-comes-to-reproducible-science-git-is-code-for-success>

¡Hemos terminado!

¡Gracias por acompañarnos y feliz Git!

Síguenos

@RLadiesGye 

R-Ladies Guayaquil 

@rladiesgps 

R-Ladies Galapagos Islands 