Git

(+ mini instructivo de entrega de TPs)

Algoritmos y Programación II

Curso: Méndez - Pandolfo

Valentina Laura Correa - 2c2023

FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Índice

Introducción

El objetivo de este documento es empezar a conocer a quien va a ser nuestro mejor amigo a la hora de escribir código, y de qué forma utilizarlo para realizar las entregas de los trabajos prácticos de la materia.

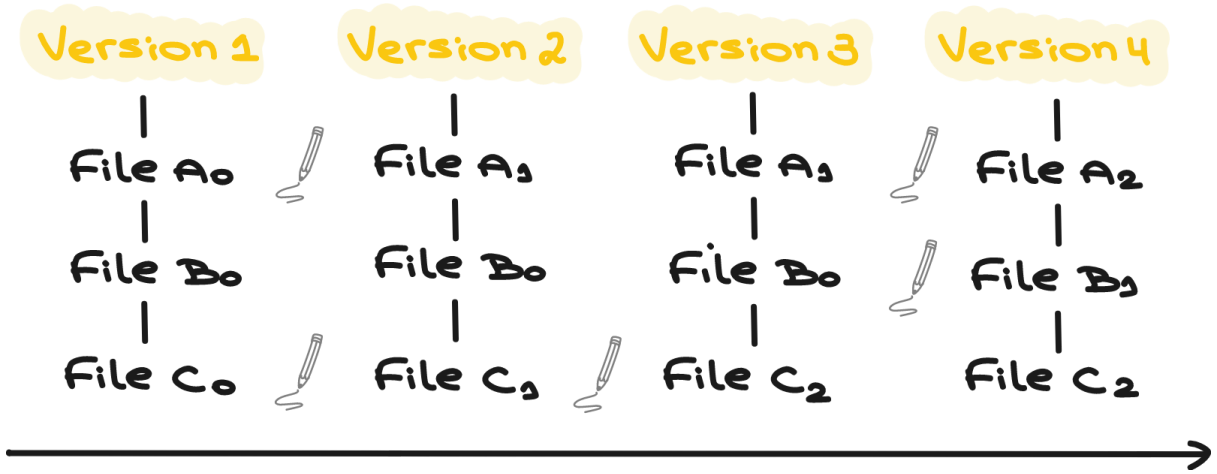
¿Empezamos?

Git

¿Qué es Git?

Git es un sistema de control de versiones, el cual tiene como objetivo facilitar el desarrollo y mantenimiento de software. Éste va guardando un registro de todos los cambios que se van realizando a lo largo del tiempo en nuestro código. Permite la restauración a versiones pasadas, distinguir que programador hizo determinados cambios y detectar posibles conflictos entre ellos, entre otras cosas.

Cada vez que se guarda el estado del proyecto, Git "le saca una foto" a lo que hay en ese preciso momento, y guarda una referencia a esa foto (snapshot).



Configuración

Antes de poder empezar a trabajar con Git, necesitamos tener algunas configuraciones previas (también recordá hacerte una cuenta en GitHub si todavía no la tenés).

En caso de que no lo tengamos descargado, vamos a necesitar correr:

```
$ sudo apt-get install git
```

en la terminal. Podemos verificar que se haya instalado correctamente mirando la salida de:

```
$ git --version
```

Por otro lado, también tenemos que indicar con qué nombre vamos a querer que identifiquen nuestros commits (vamos a ver qué es esto más adelante). Para eso hacemos:

```
$ git config --global user.name "Valen Correa"
```

Podemos verificar que se guardó correctamente corriendo:

```
$ git config user.name
```

Finalmente, realizamos también la asociación de nuestro email:

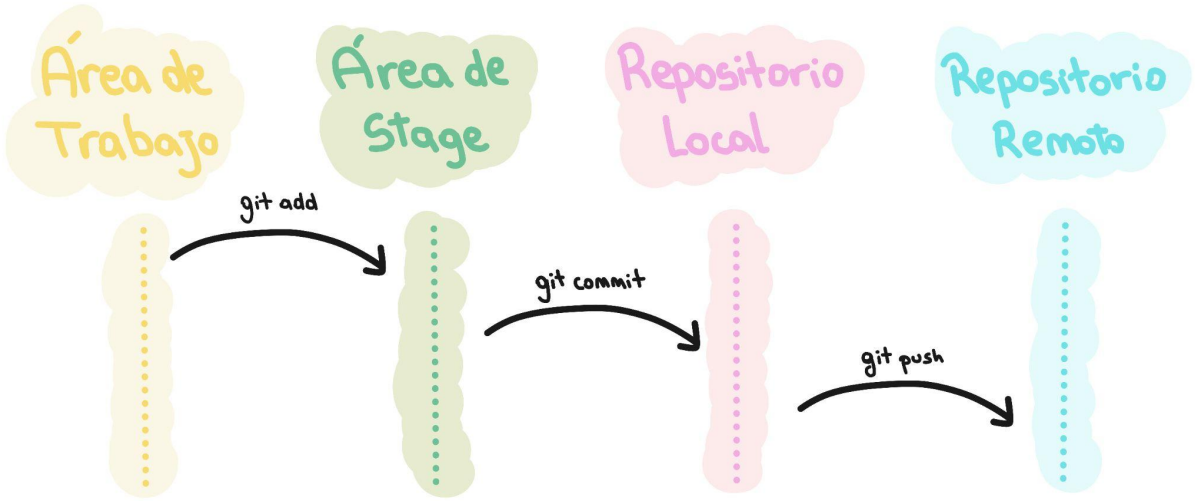
```
$ git config --global user.email "valen@gmail.com"
```

en donde nuevamente podemos validarlo con:

```
$ git config --global user.email
```

Áreas

A medida que vamos editando nuestros archivos, se espera que estos vayan pasando por distintas etapas:



Área de trabajo

También se la conoce como untracked área. Todos los cambios que se vayan haciendo en los archivos se van a ver acá usando el comando git status.
De esa forma, si nosotros lo corremos sin hacer ninguna modificación a nuestros archivos, vamos a ver algo como:

```
valentinacorrea@ubuntu ~/git/personal/repoTest
 % git status
En la rama main
Tu rama está actualizada con 'origin/main'.

nada para hacer commit, el árbol de trabajo está limpio.
```

pero si creamos y modificamos por ejemplo un archivoPrueba.txt:

```
valentinacorrea@ubuntu ~/git/personal/repoTest
 % touch archivoPrueba.txt | echo
"Holisssss!! :D" > archivoPrueba.txt
valentinacorrea@ubuntu ~/git/personal/repoTest
 % git status
En la rama main
Tu rama está actualizada con 'origin/main'.

Archivos sin seguimiento:
  (usa "git add <archivo>..." para incluirlo a lo que se será
confirmado)
        archivoPrueba.txt

no hay nada agregado al commit pero hay archivos sin
seguimiento presentes (usa "git add" para hacerles
seguimiento)
```

Área de stage

Cuando guardamos algo en el área de stage, lo que estamos haciendo es diciéndole a git que ya queremos que comience a trackear esos cambios, para que de esa forma se incluyan en el próximo commit.
Como nos decía la terminal, esto se puede hacer ya sea con git add archivoPrueba.txt, o bien git add . (este último va a incluir todos los archivos modificados que se encuentren en el área de trabajo).

```
valentinacorrea@ubuntu ~/git/personal/repoTest
 % git add archivoPrueba.txt
valentinacorrea@ubuntu ~/git/personal/repoTest
 % git status
En la rama main
Tu rama está actualizada con 'origin/main'.

Cambios a ser confirmados:
  (usa "git restore --staged <archivo>..." para sacar del área de
stage)
        nuevos archivos: archivoPrueba.txt
```

Repositorio local

Es en donde Git va a guardar toda la metadata de nuestro proyecto.
En el momento en el que se realiza un commit, se toman todos los archivos que se encontraban en el área de stage y se los guarda en un snapshot.

```
valentinacorrea@ubuntu ~/git/personal/repoTest
 % git commit -m "agrego archivo prueba"
[main 111xxx] agrego archivo prueba
 1 file changed, 1 insertion(+)
 create mode 222xxx archivoPrueba.txt
```

Éstos a su vez, van a tener un id que los define unívocamente (en el ejemplo 333xxx y 444xxx) el cual nos sirve en caso de que en otro momento necesitemos volver a esta versión.
Podemos acceder tanto a estos como al historial de commits a través de git log.

```
commit 333xxx (HEAD -> main)
Author: Valu <vcorrea@fi.uba.ar>
Date:   Thu Aug 17 16:43:22 2023 -0300

    agrego archivo prueba

commit 444xxx (origin/main, origin/HEAD)
Author: ...
```

Repositorio remoto

Acá se van a guardar las versiones de tu proyecto que estén subidas a la red. Además, el gestionar los repositorios remotos enviando y trayendo nuevos cambios, nos permite poder colaborar en el mismo proyecto con distintas personas.
Para subir nuestros cambios al repositorio remoto basta con hacer un push.

```
valentinacorrea@ubuntu ~/git/personal/repoTest
 % git push origin main
Enumerando objetos: 4, listo.
Contando objetos: 100% (4/4), listo.
Compresión delta usando hasta 12 hilos
Comprimiendo objetos: 100% (2/2), listo.
Escribiendo objetos: 100% (3/3), 1001 bytes | 1001.00 KiB/s, listo.
Total 3 (delta 0), reusado 0 (delta 0)
To github.com:valencorrea/repoTest.git
   555xxx  main -> main
```

Cheat sheet

git clone <url>
Apunta a un repositorio existente y copia su contenido en un nuevo directorio, en otra ubicación. Establece una conexión remota "origin" que apunta al repositorio original.

git checkout <branch-name>
Este comando permite desplazarte entre las ramas creadas previamente.

git checkout -b <new-branch-name>
Crea una nueva rama a partir de donde estemos parados en ese momento, y se mueve hacia ella.

git status
Nos sirve para comprobar cuáles archivos ya se incluyeron en el área de stash y cuáles no.

git pull origin <branch-name>
Extrae el contenido del repositorio remoto y con este actualiza el local.

git add <file>
Agrega el archivo indicado en el comando al área de stage.

git add .
Agrega todos los archivos al área de stage.

git log
Nos muestra el historial de commits de la rama actual hasta el momento.

git commit -m "commit message"
Añade a un commit todos los archivos que se encuentren en el área de stage.

git revert <commit-id>
Vuelve a la versión especificada. Esto lo hace a través de un nuevo commit, lo que evita perder el historial. Los commit ids los podemos ver a partir de git log.

git merge <branch-name>
Une ramas. En particular, <branch-name> será mergeada a la rama en donde estemos parados en ese momento.

git push origin <branch-name>
Actualiza el repositorio remoto con los commits nuevos que se encuentren en el repositorio local.

Entrega de TPs

Github

Cada trabajo pedido va a tener su propio repositorio. Dentro de este va a estar el enunciado junto a todos los archivos necesarios para que podamos empezar a desarrollarlo.

Para poder comenzar a trabajar, necesitamos avisarle a github que queremos crear un repositorio personal a partir del template expuesto por la cátedra. Cuando lo hagas, recordá crearlo privado y con el nombre del tp a entregar.

Una vez creado nuestro repositorio remoto, nos resta hacer un git clone para poder tenerlo localmente. Esta acción copia el contenido del repositorio indicado en un nuevo directorio, y a su vez crea una conexión remota llamada origin que apunta al repositorio original.

Por default, el repo se va a crear con la rama main, pero esta no es en la cual vamos a trabajar. Lo que haremos será crear una nueva llamada entrega a partir de la mencionada anteriormente. Haciendo un checkout -b creamos esa nueva rama y a su vez nos movemos hacia ella. Es acá en donde vamos a tener el desarrollo de nuestro código.

Ya tenemos creado nuestro repositorio de trabajo. ¡Genial! Supongamos ahora que ya estuvimos implementando un montón de cosas, pero decidimos seguir después. ¿Cómo dejamos guardados los cambios que hicimos? Si usamos el comando add, recordemos que lo que estamos haciendo es enviarlos al área de stage. De esta forma le indicamos a Git que las modificaciones que acabamos de hacer las queremos incluir en nuestra siguiente confirmación.

Una vez que decidimos que no vamos a cambiar más nada por el momento, lo único que nos resta hacer es un commit. Su mensaje asociado debería ser una breve descripción que nos ayude a acordarnos cuáles fueron las modificaciones que incluye esa versión.

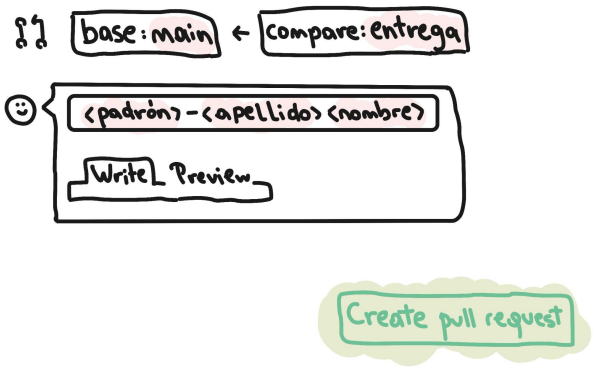
Pero... ¿Y ahora cómo lo subimos? Acordate que luego de hacer el commit, podemos subirlo a nuestro repositorio remoto haciendo un push hacia la rama en donde estuvimos trabajando.

Ahora si... después de hacer todos los commits necesarios para dejar nuestro trabajo 10 puntos, ya estamos listos para que alguien de la cátedra lo vea. ¡Pero espera! No te vayas que todavía nos queda una parte muy importante...

Pull requests

Para dar como registrada la entrega, es necesario hacer una pull request a la rama de trabajo principal. Como se mencionó anteriormente, si dejamos la configuración default, esta será la rama main.

Seleccionamos esta como base, y en compare agregamos la rama de trabajo en donde está todo el código de la entrega final. No te olvides de agregar en el nombre de la pull request tu nombre y padrón.



Es acá en donde nosotros vamos a poder dejarte comentarios sobre el trabajo.

Github actions

Github actions es una herramienta de integración continua. Esta nos trae como beneficio la automatización de tareas tales como, por ejemplo, la compilación y corrida de pruebas sobre nuestro código.

Este flujo de corridas a su vez lo podemos configurar para que se dispare a partir de un determinado evento. En particular, agregamos el archivo *.github/workflows* para que cada vez que realices un push hacia la rama main o bien crees una pull request hacia esta, se dispare la corrida de las pruebas de dicha entrega. De esta forma, vas a poder ir teniendo un seguimiento continuo y preciso del impacto de los commits.

(Recordá que para que la entrega esté aprobada tiene que pasar la totalidad de las pruebas!)

Si te quedaste con alguna duda también podes mirar este instructivo, o bien dejarnos tu pregunta en nuestro canal de discord.

Bibliografía

- https://www.atlassian.com/git
- https://docs.github.com/en/get-started/quickstart/set-up-git
- https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions
- https://git-scm.com/book