

Conceptos

Header: Se refiere a un apuntador, nos indica en donde nos encontramos (versión de commit de la rama actual).

Master: Se refiere a la rama principal de desarrollo.

Branch: Se refiere a una rama de desarrollo diferente de la rama Master. Una rama siempre se creará a partir del header actual.

Comandos de inicialización y configuración básica

Estos comandos son configuraciones iniciales que debemos usar en Git, es importante memorizarlos.

```
git init = Iniciar el repositorio en la carpeta seleccionada
git config --global user.name "nombre_ej": permite configurar el nombre del usuario.
git config --global user.email "correo_ej": permite configurar el correo del usuario.
```

Estos comandos son los básicos para empezar a trabajar nuestro git.

```
git add . : Nos permite añadir todos los cambios a la sección de staging.
git commit -m "mensaje_ejemplo": nos permite crear un nuevo commit con los últimos cambios.
git commit -am "mensaje_ejemplo": nos permite hacer un add y un commit al mismo tiempo.
```

Comandos para crear ramas de desarrollo

```
git branch (nombre deseado para branch): nos permite crear una nueva rama de desarrollo.
git branch -d (nombre del branch): nos permite eliminar una rama creada anteriormente.
git merge (nombre de la rama para el merge): Nos permite unir los cambios de una rama con otra.
```

Comandos para gestionar Git

```
gitk: abre un software gráfico que nos da un log e igualmente una representación gráfica en arbol de nuestro git.
git status: Nos permite saber el status actual de nuestro repositorio.
git log: nos tener un listado de todos nuestros commits.
```

`git log --oneline`: nos permite tener un listado de nuestros commits pero resumidos, ideal si ya tenemos muchos commits.

`Git checkout (hash del commit/branch)`: nos permite mover el Header de la branch actual hacia el último commit realizado de dicha rama, nos permite movernos entre versiones de nuestros o commits o bien a otra rama de desarrollo.

Alias: Ésto nos permite crear variables reutilizables con códigos de git, así no tendremos que escribir códigos super largos cada vez, veamos un ejemplo...

```
alias arbol ="nuestro código"
```

Ahora cuando escribamos la palabra "arbol" se ejecutará el código que hayamos puesto entre las comillas, sin tantas complicaciones.

Eliminación de archivos

`Git reset` y `git rm` son comandos con utilidades muy diferentes, pero se pueden confundir muy fácilmente.

`git rm --cached``: Elimina los archivos de nuestro repositorio local y del área de staging, pero los mantiene en nuestro disco duro. Básicamente le dice a Git que deje de trackear el historial de cambios de estos archivos, por lo que pasaran a un estado ``untracked``.

`git rm --force``: Elimina los archivos de Git y del disco duro. Git siempre guarda todo, por lo que podemos acceder al registro de la existencia de los archivos, de modo que podremos recuperarlos si es necesario (pero debemos usar comandos más avanzados).

Creación de repositorios remotos





Llaves publicas y privadas

Crear llaves localmente (localmente)

Lo primero que necesitamos para crear un repositorio remoto es crear unas llaves privadas esto es relativamente sencillo aunque hay que ser específicos con el código. El código para crear una llave ssh es con git es:

```
ssh-keygen -t rsa -b 4096 -C "correo_GitHub"
```

Una vez hecho eso, Git nos recomendará la carpeta donde guardar, así que solamente damos enter. Le podemos poner una contraseña a esa llave ssh, ésto es seguridad adicional, por practicidad no la pondremos, pero es recomendable sí ponerla, por buenas prácticas más que nada. La carpeta designada contendrá archivos como los que se muestran a continuación.

<input type="checkbox"/> Nombre	Fecha de modificación	Tipo	Tamaño
 id_rsa	18/02/2023 07:31 p. m.	Archivo	4 KB
 id_rsa.pub	18/02/2023 07:31 p. m.	Microsoft Publisher Document	1 KB
 known_hosts	18/02/2023 07:45 p. m.	Archivo	1 KB
 known_hosts.old	18/02/2023 07:45 p. m.	Archivo OLD	1 KB

id_rsa: Llave privada > La llave que no debemos compartir y debemos cuidar demasiado.

id_rsa.pub: llave pública > la que usaremos publicamente y sí podemos compartir, se usará para nuestras cuentas de GitHub u otras.

Necesitamos comprobar el proceso y agregarlo a Windows:

```
eval $(ssh-agent -s)
ssh-add ~/.ssh/id_rsa
```

El simbolo "~" es un shortcut absoluto, que buscará dentro de la carpeta home de nuestro sistema.

El comando ssh-add añadirá nuestra llava privada, si tiene otro nombre hay que sustituirlo.

Para añadir el proyecto remoto usamos el siguiente código:

```
git remote add "nombre_ej" git@github.com:"nuestro proyecto"
```

Es importante notar que en "nombre_ej" podemos poner cualquier nombre que queramos, en este caso se llamará teamProject pero puede ser cualquier nombre, como podemos ver a continuación.

Ejemplo (con el proyecto actual):

```
git remote add teamProject git@github.com:Baalgarthem/proyectoCompilador.git
```

Si en alguna ocasión necesitamos cambiar la dirección de nuestro proyecto remoto podemos usar el siguiente comando:

```
git remote set-url origin git://new.url.here
```

Donde "origin" sería el nombre que hayamos elegido, en este caso fue *teamProject*.

Si deseamos en cambio, no cambiar la dirección sino eliminar nuestro vinculo remoto, podemos usar el siguiente comando:

```
git remote remove origin
```

Conectar repositorio

Ahora el siguiente paso importante para configurar nuestro repositorio con GitHub es ir a la página oficial:

<https://github.com/settings/keys>

Y hacer lo que se muestra en la siguiente imagen:

Cuando seleccionemos "New SSH key" metemos el contenido de nuestra llave publica. En la carpeta que creamos abrimos nuestra

llava la que tiene el formato "pub" de publico. Es recomendable abrir el archivo con el bloc de notas, posterior a eso se copiará el contenido de ese archivo y se pegará en el recuadro de SSH keys, tal cual se muestra en la imagen de arriba.

Una vez hecho lo anterior y para comprobar que todo está correcto usamos el siguiente comando

git remote -v

Si hicimos todo correctamente, ya deberíamos poder hacer comandos remotos, tenemos dos direcciones remotas, una para hacer fetch y otra para hacer pull. Si nos aparecen como en la siguiente imagen, significa que ya terminamos de configurar el ámbito remoto.

```
6de27be se añade un documento de proyecto compilador, para estructurar y documentar el proyecto
Alush@Reign MINGW64 /d/Academico/Lenguajes y automatas 11/Proyecto Compilador (ejercicios)
$ git remote add teamProject git@github.com:Baalgarthem/proyectoCompilador.git
Alush@Reign MINGW64 /d/Academico/Lenguajes y automatas 11/Proyecto Compilador (ejercicios)
$ git remote -v
teamProject      git@github.com:Baalgarthem/proyectoCompilador.git (fetch)
teamProject      git@github.com:Baalgarthem/proyectoCompilador.git (push)
Alush@Reign MINGW64 /d/Academico/Lenguajes y automatas 11/Proyecto Compilador (ejercicios)
$
```

Si ejecutamos

git remote

veremos solamente la conexión remota (hacia nuestro GitHub) que hayamos configurado, tal cual se muestra a continuación:

```
Alush@Reign MINGW64 /d/Academico/Lenguajes y automatas 11/Proyecto Compilador (ejercicios)
$ git remote
teamProject
```

Comandos necesarios para trabajar con repositorios remotos.

Para recapitular, entendamos que los comandos add, commit, trabajan sobre nuestros archivos locales. O sea que no afectarán al repositorio remoto de ninguna forma.

El comando para enviar archivos al repositorio remoto es "push". Mientras que el comando para jalar o bien descargar todos los cambios más recientes del repositorio es "pull". Siempre es recomendable que hagamos un pull (antes de comenzar a trabajar) para descargar los ultimos cambios antes de hacer un push, esto más que nada para no provocar cambios o inconsistencias en el código (cuando se trabaja en equipos grandes, si son menos de cinco personas esto podría omitirse o bien si una sola persona se

encargará de una rama, entonces no hay necesidad de cumplir esta regla sin embargo es una buena práctica y es recomendable acostumbrarse a ello)

```
git push teamProject master  
git pull teamProject master
```

Recordemos que teamProject es solamente el nombre que elegimos, casi siempre se suele llamar origin, pero podemos poner cualquier nombre que deseemos.

Añadir colaboradores en en Github

Una vez que un miembro del equipo ya ha creado el repositorio, los demás miembros del equipo seguramente estarán emocionados por comenzar a usar la herramienta git. Cada miembro del equipo necesitaría una cuenta ya sea en GitHub, o en GitLab, o cualquier otro sistema de su preferencia, pero en esencia el funcionamiento es igual en todos lados. Así que en éste caso estoy usando GitHub.

Search or jump to... / Pull requests Issues Codespaces Marketplace Explore

Baalgarthem / proyectoCompilador Public

<> Code Issues Pull requests Actions Projects Wiki Security Insights Settings

master 3 branches 0 tags

Go to file Add file <> Code

Baalgarthem Se marca el inicio del ejercicio 2 54be8b8 last week 11 commits

#Diagramas	cambios de ejemplo de prácticas	last week
#Documentación	Punto de respaldo del proyecto	last week
#Ejercicios en JavaCC	Se marca el inicio del ejercicio 2	last week
.gitignore	Se añade el primer ejercicio de los ejercicios en Java CC, en donde e...	last week

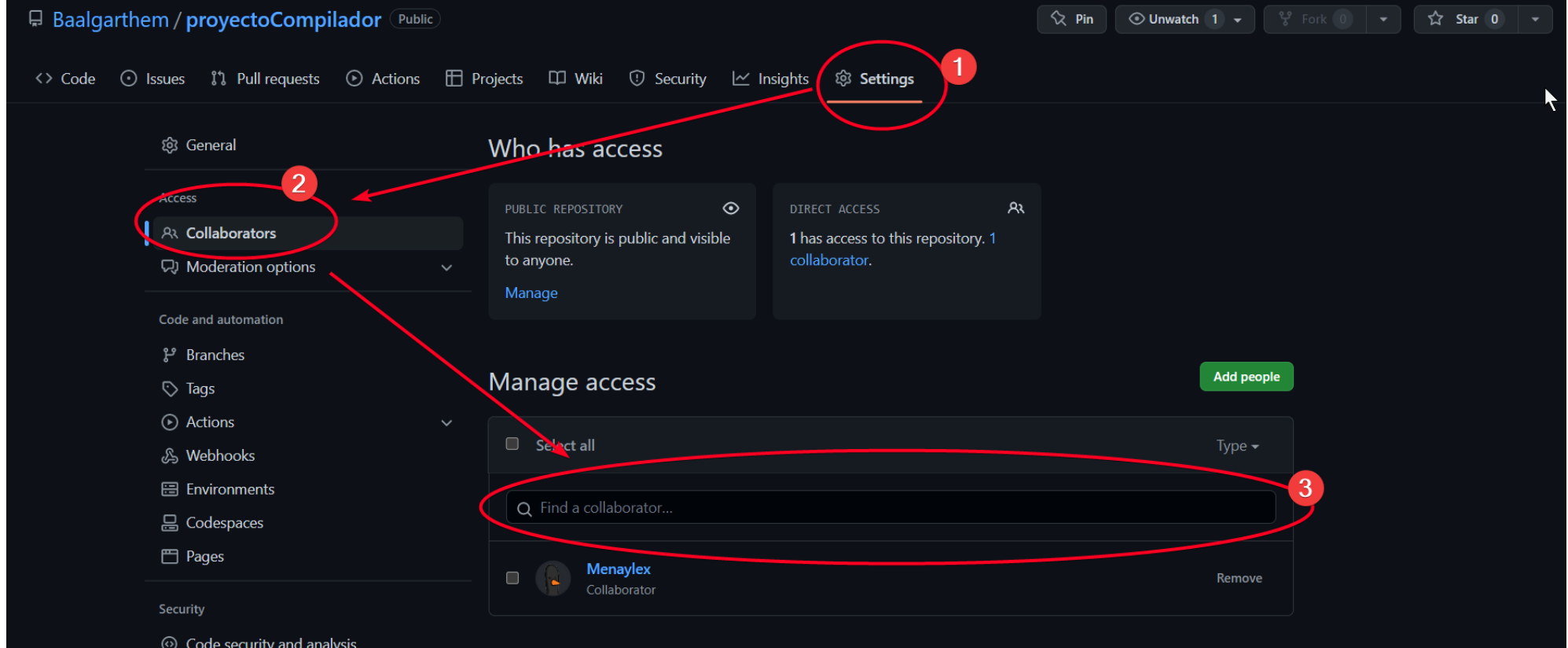
Help people interested in this repository understand your project by adding a README. Add a README

About
Proyecto compilado
automatas II | TECN
0 stars
1 watching
0 forks

Releases
No releases published
Create a new release

Packages
No packages published
Publish your first package

Para colaborar de forma remota en GitHub lo que se necesita es (desde las configuraciones de nuestro proyecto) añadir un colaborador, para esto el colaborador debe darnos su nombre de usuario en GitHub y debemos añadirlo como colaborador al proyecto. Una vez hecho eso esta persona ya tendría permisos de push.



Para comenzar a trabajar se necesita crear una nueva carpeta e iniciar git bash en ella. **PERO (IMPORTANTE LEER)** no debe hacer un git init, porque ésta persona no va a iniciar un repositorio, sino que va a jalar (pull) un repositorio ya existente. Si se inicia un repositorio y luego se hace un clone provocará una inconsistencia en git que nos dirá "estas creando un repositorio dentro de otro repositorio". Para evitar ésto, simplemente no hagamos un git init. Eso se utiliza solamente cuando nosotros vamos a ser los masters.

En resumen:

- Para añadir un colaborador, la forma más fácil es con su nombre de usuario publico en github, desde las configuraciones (de nuestro repositorio al que lo queremos añadir) y buscando su nombre de usuario.
- Un colaborador no debe usar el comando "git init"
- Una vez que el colaborador ha sido invitado (y aceptó) a participar en el proyecto debe hacer un pull por primera vez.
- Finalmente, debe hacer su primer push y checar el log para comprobar que sus cambios se han realizado con éxito.

Trabajar con ramas en Git y GitHub

Para trabajar con ramas tanto en git como en github necesitamos un conjunto de codigos que se diferencia entre, codigos locales y codigos remotos. Los locales trabajan unicamente sobre nuestro equipo, donde tengamos nuestros archivos de trabajo, mientras que los comandos remotos afectarán al repositorio online.

Para saber cuales ramas existen en nuestro repositorio local podemos escribir este comando en nuestro git bash.

```
git branch
```

Eso nos mostrará un listado de todas las ramas que tenemos actualmente, si en cambio queremos crear una nueva rama es bastante sencillo

```
git branch nombre_rama
```

Solamente tendríamos que reutilizar el comando branch y añadirle el nombre de la rama que queremos crear. ¿Pero y qué pasa si queremos borrar la rama porque ya no la vemos necesaria? eso es simple también. Hacemos el mismo comando pero añadiendo una flag -d (de delete) en caso de que haya complicaciones git nos podría recomendar usar una forma forzada de eliminación -D. Veamos los códigos.

```
git branch -d nombre_rama : este codigo borrará una rama  
git branch -D nombre_rama : este codigo también borrará una rama (pero de forma forzada)
```

Cabe recalcar, que al ser una bandera, no importa si la ponemos despues o antes del nombre, lo importante es ponerla. También, podemos añadir muchas banderas a diferentes comandos en git. Si esto se combina con el comando "alias" puede ser una forma increíblemente rapida y efectiva de trabajo. En todo caso, continuemos.

Si queremos renombrar una rama podemos usar el siguiente comando:

```
git branch -m nombrePrevio nombreNuevo
```

Como limpiar archivos innecesarios de nuestro Git

En el flujo de trabajo, quizás hayamos creado archivos de más, o simplemente queramos eliminar cualquier cosa que no estemos usando en nuestro proyecto, para ello podemos usar los siguientes comandos

```
**git clean --dry-run** Para visualizar qué archivos se van a borrar. Archivos, no carpetas  
**git clean -f** Para borrar los archivos que visualizaste: Ignorará lo que esté dentro del archivo .gitignore  
**git clean -df** Borra archivos y carpetas
```

—Ing. Arturo Ramirez, TECNM - ALVARADO , 2023