CLASE 11 - GIT

Material de lectura

Aprende Git:
https://www.atlassian.com/es/git/tutorials/what-is-version-control

Lista de comandos
https://gist.github.com/dasdo/9ff71c5c0efa037441b6

Como usar Git desde Visual Studio Code
https://www.digitalocean.com/community/tutorials/how-to-use-git-integration-in-visual-studio-code-es

Como volver a un estado (commit) anterior
https://devconnected.com/how-to-git-reset-to-head/

---

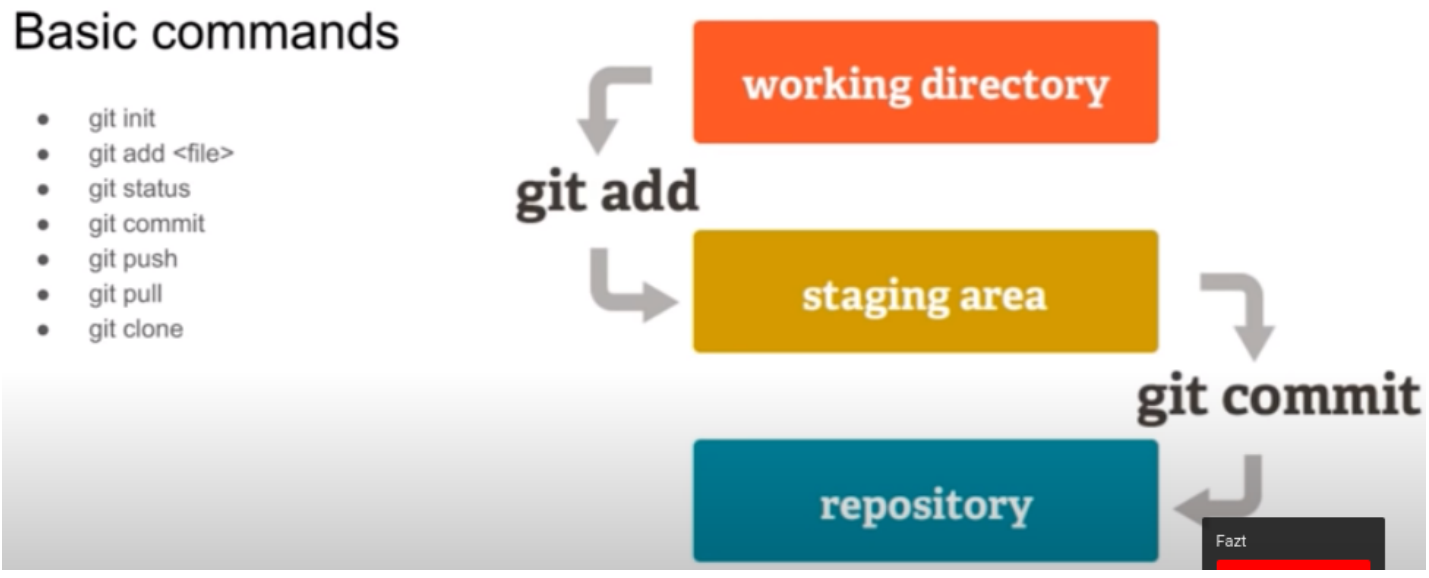
0) Nociones básicas de git:

(Ver más detalles en las diapositivas de la clase y el material del aula virtual)

- Es un sistema de **control de versiones (el mas popular)**
- Controla / administra distintas versiones de un programa (o cualquier proyecto)
- A medida que se van produciendo cambios, git puede llevar un registro de las versiones intermedias.
- OpenSource, creado por Linus Tovals, para controlar los cambios que se producen en el kernel de Linux
- Permite trabajar sobre un mismo programa entre varios desarolladores, que comparten un repositorio de código y trabajan sobre una copia local del mismo.

Lleva un registro de cada linea de código escrita, y es posible volver a versiones anteriores (revertir cambios).

- Trabaja con repositorios LOCALES y REMOTOS.



1) Uso de comandos consola

Comandos básicos del Sistema Operativo:

$ **ls** -> archivos en el directorio
$ **ls -lh** -> mas detallado
$ **ls -lha** -> incluye archivos ocultos
$ **cd** -> cambiar a un directorio/carpeta
$ **md / mkdir** -> Crear directorio/carpeta
$ **mv** -> Renombrar archivo
$ **rm** -> Borrar archivo (-r si es una carpeta)

$ **clear / cls** -> Limpia la pantalla de la consola en Windows / Linux
$ **touch** -> Crea archivos (vacios)
$ **pwd**

---

2) Crear cuenta:

Para crear una cuenta en github, nos dirigimos a la url https://github.com/ y seguimos los pasos. Desde la opcion "Signup" realizamos el proceso. Vamos a necesitar una dirección de correo válida, y seleccionar un nombre de usuario y contraseña.

---

3 Instalar git

Desde la url https://git-scm.com/downloads elegimos la versión correspondiente a nuestro SO, descargamos e instalamos.

---

4) configuracion

Los siguientes comandos nos permiten realizar la configuración inicial de git:

git --version (consultamos la version instalada de git)

git config --global user.name "ArielCodo"
git config --global user.name ariel.palazzesi@bue.edu.ar
git config --global core.editor "code --wait"

git config --global -e (ver configuracion, la va a mostrar en VSCode por lo que configuramos antes)
git config -h (ayuda de la opcion "config")

---

5) Creamos un directorio

Utilizando los comandos vistos, creamos un directorio de trabajo y nos posicionamos en él.

---

6) Inicializamos el directorio creado.

git init (ver el .git)

Los nombres comúnmente elegidos en lugar de '**master**' son 'principal', 'troncal' ydesarrollo'. Se puede cambiar el nombre de la rama recién creada mediante este comando:

git branch -m <nombre>

Actualmente se está realizando una campaña para utilizar como rama principal "**main**" en lugar de "**master**". Se puede utilizar, en rigor, cualquier nombre que el usuario desee. Por cuestiones de forma y compatibilidad, lo ideal seria utilizar "main" o "master".

---

7) Comenzamos a usar git

Abrimos el editor de texto, abrimos la carpeta y creamos el archivo archivo1.txt

git status

Con git add podemos agregar archivos al staging area:
git add archivo1.txt (otras formas: archivo add *.txt , archivo add archivo1.txt archivo2.txt , git add .)

git status (deberíamos ver que ya tenemos archivos listos para ser confirmados)

Creamos archivo2.txt usando touch u otro mecanismo. Luego, si hacemos

git status

aparece el nuevo archivo para ser agregado. Lo agregamos y vemos con git status que esta listo para ser agregado nuevamente.

git add archivo2.txt
git status

Si modificamos algun archivo, por ejemplo el archivo2.txt y hacemos git status, vemos también que esta listo para ser agregado nuevamente.

git status (aparece archivo2.txt en verde y rojo)
git add archivo2.txt
git status

---

8) Commit

Con commit pasamos los archivos que estan en el staging area a nuestro repositorio local:

git commit -m "Commit inicial"
git status

Cada vez que realizamos cambios importantes, repetimos el proceso de agregar los archivos al staging arear, y realizamos un commit nuevo. Por ejemplo, si modificamos archivo2.txt, podríamos hacer:

git status
git add archivo2.txt
git status

git commit (Abre el editor de texto para poner el mensaje)
git status

Con **git log** puedo ver el resultado del (los) commit(s) realizado(s).

Nota:
Con **git commit -am "comentario"** se obtiene el mismo resultado que usando los dos comandos **git add .** y **git commit -m "comentario"** uno despues del otro.

---

9) Eliminando archivos

Es posible eliminar archivos con de una carpeta con rm. En ese caso, deberiamos sacarlo (si lo habiamos agregado) del staging area. Veamos los pasos necesarios:

ls
rm archivo2.txt

Nota: Nos adelantamos un poco, pero con **git reset --hard HEAD** podríamos volver al estado del último commit y recuperar el estado de los archivos previos al cambio....

git status (aparece archivo2.txt en rojo)
git add archivo2.txt
git status

git commit -m "Eliminando Archivo2.txt"
git status

El proceso de borrar y hacer el git add se puede hacer con
git rm archivo1.txt (Borra y agrega al staging area)

---

10) Mover o cambiar nombres de archivos

mv archivo1.txt archivo.txt (cambia el nombre)
ls

git status (Hay dos cambios, uno nuevo, uno eliminado)
git add archivo1.txt archivo.txt
git status (Hay un archivo renombrado)
git commit -m "Renombramos archivos"

El proceso de cambiar de nombre y hacer el git add se puede hacer con
git rm archivo1.txt (Borra y agrega al staging area)

git mv archivo.txt archivo1.txt
git status

git commit -m "Renombramos archivos otra vez"

---

11) Quitar cambios de la staging area

git restore --staged archivo1.txt
git status

(archivo1.txt no esta en el staging area)

---

12) Recuperar archivos / Volver a un estado anterior

Por supuesto, es posible volver a un estado anterior. Con

git restore archivo1.txt
recuperamos un archivo desde el ultimo commit.

Si tenia cambios, archivo1.txt ha sido recuperado de la versión guardada en el ultimo commit

Con

git reset --hard HEAD

Restauramos el estado del contenido de la carpeta a como estaba en el momento de hacer el ULTIMO commit. Y con

git reset --hard HEAD<numero de commits hacia atrás>

recuperamos el contenido que teníamos hace un número determinado de commits. Podemos usar

git log o **git log -oneline** para buscar el número de commit apropiado.

Mucha más info en https://devconnected.com/how-to-git-reset-to-head/

---

13) Ignorar archivos

Podemos crear un archivo llamado .gitignore y colocar dentro una lista de los archivos que no queremos incluir en nuestros add y commit.

Creo un archivo .env con alguna configuracion
git status
ls
ls -a

Creo el archivo .gitignore y agrego a .env dentro.
git status (.env ya no aparece, pero aparece gitignore)

lo agregamos y comiteamos:
git add .gitignore
git commit -m "Agregamos git ignore"
git status (.env ya no aparece, y .gitignore tampoco)

---

14) Alternativa a git status:

hacemos algunas modificaciones en los archivos.
clear
git status

git status -s

M roja: modificado
M verde: agregado al staging area
?? rojo: Archivo es nuevo, no se ha agregado.

git add archivo2.txt
git status -s

A verde: Estamos agregando a archivo2.txt

Rojo: sin incluir en el staging area
verde: listo para commit

---

15) Git diff

Modificamos y guardamos archivo2.txt
git status

git diff
- rojo: lineas eliminadas
+verde: lineas agregadas
(ver caso del salto de linea que no se ve en la primer linea)
@@ -1 +1,3 @@ (quitamos en linea 1, agregamos entre 1 y 3)

Si es necesario, se sale de la pantalla con las diferencias usando la tecla "q"

git add archivo2.txt
git commit -m "cambios...."
git diff (ya no hay cambios)

git diff --staged (muestra cambios con el staged area)

---

16) Ver historial del repositorio

Si necesitamos ver todos los commits que hemos realizado en nuestro repositorio, usamos

git log (largo, mucha información que a veces no es necesaria)

se sale con q. En caso de querer ver solo la información indispensable, usamos

git log -oneline (resumen, de una linea por commit)

Resulta muy importante, como se puede ver, utilizar descripciones pertinentes a la hora de hacer cada commit.

---

17) Ramas

Veamos como crear y manejar ramas:

git status
git restore --staged archivo2.txt

Verificamos en que rama estamos con
git branch (estamos en la rama master / main)

git checkout -b ramab (creamos la rama ramab)
git branch (estamos en la rama ramab)

Agregamos algo en el archivo2.txt

git status
git add archivo2.txt
git commit -m "actualizado archivo 2"
git log -oneline (vemos que estamos en head ramab)

cat archivo2.txt (muestra el contenido de archivo2.txt de rama b)

Le hacemos algun cambio al archivo archivo2.txt, lo stageamos y comitamos:
git add archivo2.txt
git commit -m "Artualizamos archivo2.txt otra vez"

git checkout master (Volvemos a rama master)
cat archivo2.txt (muestra el contenido de archivo2.txt de rama master)

Tenemos varias versiones del codigo existiendo a la vez.

---

18) Fusionar ramab con master

Debemos estar en master!

git merge ramab
Aca puede ser necesario agregar y comitear.
Y si sale bien, se vera que archivo2.txt esta actualizado.
Si hay conflictos, los abre en el editor CODE para solucionarlos manualmente.

---

19) Subir a repositorio remoto en github

Hacemos un repositorio llamado, por ejemplo "mi web", público, en hithub.

Copiamos con
git remote add origin https://github.com/ArielCodo/miweb.git
(reemplazando ArielCodo por el usuario que corresponda)

y la pegamos en la terminal. Le estamos indicando de donde y donde vamos a subir los cambios.

Copiamos "git push -u origin main" y la pegamos en la terminal
git push -u origin master

Estamos creando una rama llamada master en el servidor remoto

Nos pide el usuario: (ArielCodo en mi caso)
y una contraseña QUE NO ES LA DE GITHUB. Hay que hacerla en github,
desde el usuario -> Settings -> developer setings -> Personal tokens
Genero un token y copio. La pego en la terminal.
Todo el codigo sube, y ya puedo verlo en **github**

Podemos crear y subir mas ramas:
git checkout -b ramac
git push -u origin ramac

Si voy a **github**, tengo la nueva rama.

Las actualizaciones siguientes simplemente se hacen con

git push

---

20) Descargar modificaciones desde Github:

Simplemente,

git pull

---

21) Cómo enviar cambios a GitHub si el repositorio no es tuyo (vía fork)

Si el repositorio que vas a modificar no es tuyo y pretendes continuar el desarrollo,
agregando cambios que querrás enviar a GitHub, tendrías que clonar el repositorio de otra manera.

Primero tienes que crear un "fork". Esto es, un repositorio que es copia de otro que ya está publicado en GitHub. Lo bueno de un fork es que el repositorio será exactamente igual, pero estará en tu cuenta de GitHub, con tu usuario, por lo que tendrás permisos para hacer lo que tú quieras con él.

Hay un botón para hacer el fork en la parte de arriba de la página del repositorio.
Una vez hecho el fork, el repositorio ya te pertenece. Entonces podrás clonar (el fork, no el repositorio original) y realizar los cambios que quieras, que podrás subir perfectamente a ti propio repositorio (tu fork).

IMPORTANTE: Si nos aparece el error **remote: Permission to yyyyy.git denied to xxxxx. remote: Permission to ArielCodo/prueba1.git denied to ar13lp.**
Hay que cambiar el usuario com
git config credential.username "ArielCodo"
