**GIT Y GITHUB**

7. Introducción a la terminal y línea de comandos

• Muestra la ruta de la carpeta en la que estoy:

**pwd**

• Navegar a una carpeta **NOTA**: Ningún nombre de la ruta del repositorio lleva espacios:

**cd <**carpeta**>** ó **<**carpeta**>/<**carpeta**>**

• Cuando termina en **~** está en el home del usuario, y cuando termina en **/** en la raíz del disco.

• Ir al home desde la raíz del disco (/):

**cd ~**

• Ir a la raíz del disco desde el home (~):

**cd /**

• Ver el grupo de archivos de la carpeta en la que estoy excluyendo los ocultos:

**ls**

• Ver una lista de los archivos incluyendo los ocultos:

**ls -al**

• Ir viendo los comandos que se han escrito:

**Flecha arriba**

• Ver una lista de los archivos excepto los ocultos:

**ls -l**

• Ver el grupo de archivos de la carpeta en la que estoy incluyendo los ocultos:

**ls -a**

• Limpiar consola:

**Ctrl** + **L** ó **clear**

• Regresar a la carpeta anterior:

**cd ..**

• Autocompletar una ruta:

**cd <**primera letra**>** + **Tab**

• Regresar al home del usuario

**cd**

• Crear carpeta:

**mkdir <**nombre para la carpeta**>**

• Crear archivo:

**touch <**file.ext**>**

**•** Ver el contenido de un texto plano:

**cat <**file.ext**>**

• Ver historial de comandos:

**history**

• Repetir comando del historial de comandos:

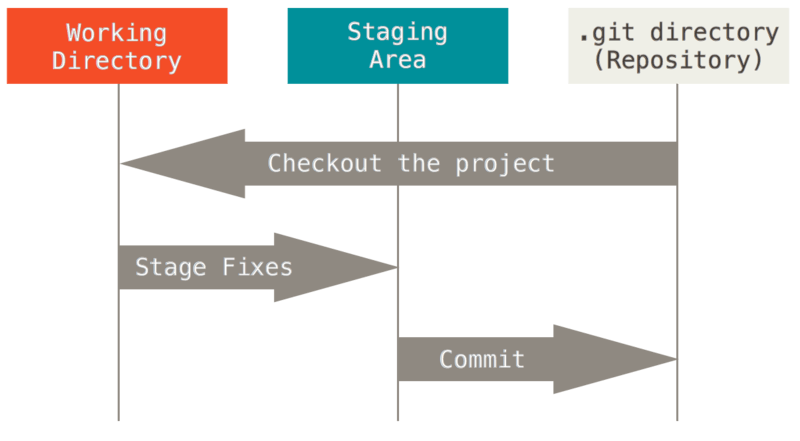
!**<**# del comando en la lista**>**

• Eliminar archivo de la carpeta, **NOTA**: CUIDADO POR QUE SI SE HACE MAL SE PUEDE BORRAR EL DISCO DURO:

**rm <**file.ext**>**

• Saber todas las opciones de un comando:

**<**comando**> --help**

8. ¿Qué es el staging y los repositorios? Ciclo básico de trabajo en Git

10. Crea un repositorio de Git y haz tu primer commit

• Crear repositorio:

**git init**

• Abrir VSC:

**code**

• Ver estatus del proyecto:

**git status**

• Agregar archivo al staging:

**git add <**file.ext**>**

• Agregar TODOS los archivos modificados al staging, **NOTA**: . porque ahí es la carpeta actual donde está todo:

**git add .**

• Sacar archivo y sus cambios de staging:

**git reset HEAD <**file.ext**>** (Soportado en todas las versions de git)

ó

**git restore --staged <**file.ext**>** (Version más moderna y descriptiva, soportado desde git 2.23)

• Sacar archivo y sus cambios de staging, pero no del disco duro:

**git rm --cached <**file.ext**>**

• Borrar archivo del staging y del disco duro, **NOTA**: Aunque se pueda recuperar con comandos más avanzados, **ES PELIGROSO**:

**git rm --force <**file.ext**>**

• Ver todas las configuraciones de git:

**git config**

• Ver configuración default de git:

**git config --list**

• Salir del list de la configuración:

**q**

• Ver donde están guardadas las configuraciones:

**git config --list --show-origin**

• Cambiar la configuración global del nombre de todos los usuarios, **NOTA**: Dos guiones es para usar palabras, uno, solo para letras, ejemplo: ls, al, etc.:

**git config --global user.name “**usuario**”**

• Cambiar la configuración global del correo de todos los usuarios:

**git config --global user.email “**correo@correo.com**”**

• Mandar archivo/s al repositorio, **NOTA**: Es buena práctica siempre poner un mensaje con una breve descripción del archivo o de qué se cambió:

**git commit -m “**mensaje**”**

• Ver el contenido de un archivo .txt desde Git Bash:

**cat <**archivo**.txt>**

• Abrir VSC para editar un archivo:

**code <**file.ext**>**

• Ver el historial de commits de un archivo:

**git log <**file.ext**>**

• Ver todo el historial de commits

**git log**

• Salir del historial de commits del archivo:

**q**

11. Analizar cambios en los archivos de tu proyecto con Git

• Ver los cambios que se han hecho en un archivo, **NOTA**: Muy útil para ver que puede estar mal en un código que funcionaba y ahora no:

**git show <**file.ext**>**

• **Vim**: Es un editor de texto. El texto de los #’s son comentarios, nada de eso ira a Git.

• Escribir en vim:

**Esc** + **I** ó solo escribir

• Salir de vim, **NOTA**: Esto guarda el archivo y fuerza él envió del commit:

**Esc** + **Shift** + **Z** + **Z**

• Ver las diferencias entre dos commits, **NOTA**: El número se copia con Ctrl + Ins y se pega con Shift + Ins:

**git diff <**# del commit anterior**> <**# del commit reciente**>**

• Ver cambios de archivos entre lo que está en staging y lo que está en el disco duro:

**git diff**

12. Volver en el tiempo en nuestro repositorio utilizando reset y checkout

• Regresar un archivo a una versión anterior, **NOTA**: Con **--hard** se regresa a la versión anterior en todos lados, **ES PELIGROSO**, porque borra todo lo que se halla hecho hasta ese momento. Con **--soft**  solo lo que este en staging vuelve a la versión anterior, el working directory no cambiara, o sea, solo mueve el puntero de la rama (HEAD) al commit anterior:

**git reset <**# del commit al que se quiere regresar**> --hard**

**git reset <**# del commit al que se quiere regresar**> --soft**

• Ver cambios específicos en los archivos:

**git log --stat**

• Ver una versión anterior de un archivo, **NOTA**: Al checar se verá que el archivo fue modificado, **ESTO ES MUY PELIGROSO**, si se hace commit se borraran todos los cambios antes de checar la versión:

**git checkout <**# del commit de la versión anterior**> <**file.ext**>**

ó

**git switch <**# del commit de la versión anterior**> <**file.ext**>**

• Regresar a la versión actual trayendo la versión de main:

**git checkout main <**file.ext**>**

15. Introducción a las ramas o branches de Git

• Mandar un commit directo sin hacer add, solo con los cambios de archivos que ya halla agregado antes con **git add** al staging, no funciona para archivos nuevos:

**git commit -am “**mensaje**”**

• Crear una rama:

**git branch <**rama**>**

• Moverse a una rama:

**git checkout <**rama**>**

• Eliminar rama **definitivamente**:

**git branch -D <**rama**>**

• **HEAD**: Es el apuntador de en donde se está trabajando.

16. Fusión de ramas con Git merge

• Si se tienen cambios y se cambia de rama antes de agregarlos al staging, **SE PUEDEN PERDER LOS ARCHIVOS**.

• El merge debe ser desde main o desde la rama que se quiera que ahora sea la principal.

• Ver que ramas hay:

**git branch**

• Hacer **merge**, **NOTA**: merge puede llevar -m “mensaje” al final

**git merge <**rama**>**

18. Uso de GitHub

• Agregar origen remoto de nuestros archivos, **NOTA**: Para esto primero se crea un repositorio en GitHub con el nombre del proyecto y se copia su dirección https:

**git remote add origin <**dirección https del repositorio en GitHub**>**

• Ver si hay repositorio remoto, si lo hay saldrá la palabra “origin”:

**git remote**

• Ver la dirección específica del repositorio, **NOTA**: fetch es traer y push enviar:

**git remote -v**

**1/3**: Antes de hacer push primero se debe hacer pull…

• Traer del repositorio remoto a la rama main:

**git pull origin main**

**2/3**: … luego…

• Forzar a que los historiales remoto y local se fusionen:

**git pull origin main --allow-unrelated-histories**

**3/4**: … ahora hacer commit de los cambios traídos…

**4/4**: … y finalmente…

• Enviar al repositorio remoto la rama main:

**git push origin main**

20. Configura tus llaves SSH en local

• Asegurarse que el correo local sea el mismo que el remoto.

• Generar llaves SSH, **NOTA**: Se hace desde el home (~), es recomendable agregar un passphrase:

**ssh-keygen -t ed25519 -b 4096 -C “**correro@correo.com**”**

• Revisar que el servidor de llaves SSH este activado, **NOTA**: Si sale “Agent pid <numero>” si funciona:

**eval $(ssh-agent -s)**

• Atajo para ir a una carpeta desde la raíz del disco (/):

**cd ~/<**carpeta**>** ó **<** carpeta **>/<** carpeta **>**

• Ir a la carpeta de las llaves desde (/):

**cd ~/.ssh**

• Agregar llave **PRIVADA** a GitHub, **NOTA**: El comando dice; “agregar llave ssh que está en el home, en la carpeta **.ssh** y la llave se llama **id\_** **ed25519**”. Se sabe que está en la carpeta **.ssh** al checar ahí con **ls -al**:

**ssh-add ~/.ssh/id\_** **ed25519**

21. Conexión a GitHub con SSH

• Agregar llave **publica** en GitHub, **NOTA**: La llave se llama **id\_** **ed25519.pub**, está en **C:\Users\usuario\.ssh** y se abre con **VSC**:

**Settings** > **SSH and GPG keys** > **New SSH key**

• Cambiar url del repositorio local por el SSH remoto:

- Copiar url del repositorio en GitHub de **Code** > **Clone with SSH**

- **git remote set-url origin** **<**git@github.com:usuario/proyecto.git**>**

• Enviar cambios al remoto:

1. Traer la última versión del remoto con **git pull**

2. Confirmar los cambios

3. Traer otra vez y fusionar con main con **git pull origin main**

4. Checar todo con **git status** y **git diff**

5. Hacer commit con **git commit -am “**mensaje**”**

6. Volver a traer y fusionar con main con **git pull origin main**

7. **git push origin main**

22. Tags y versiones en Git y GitHub

• Ver todo el historial de commits:

**git log --all**

• Ver el grafo del funcionamiento de las ramas:

**git log --all --graph**

• Ver el grafo del historial de commits comprimido:

**git log --all --graph --decorate --oneline**

• Poner alias a un comando, **NOTA**: Se ejecuta escribiendo el alias:

**alias <**alias**>=”**comando**”**

• Poner tag a un commit:

**git tag -a <**versión, ejem: v0.1**> -m “**mensaje**” <**# de commit**>**

• Ver lista de todos los tags:

**git tag**

• Ver a que commits pertenecen los tags:

**git show-ref --tags**

• Enviar los tags al remoto:

1. Traer la última versión del remoto con **git pull origin main**

2. **git push origin --tags**

• Borrar tag, **NOTA**: Aun así, si se actualiza el remoto, no se borrará:

**git tag -d <**tag**>**

• Borrar tag del remoto definidamente:

**git push origin :refs/tags/<**tag**>**

23. Manejo de ramas en GitHub

• Ver las ramas y su historia:

**git show-branch**

• Ver las ramas y su historia más detallado:

**git show-branch --all**

• Ver ayuda visual de git:

**gitk**

• Mandar una rama al remoto:

1. Traer la última versión del remoto con **git pull origin main**

2. Moverse a la rama que se quiere enviar con **git checkout <**rama**>**

3. **git push origin <**rama**>**

24. Configurar múltiples colaboradores en un repositorio de GitHub

• Clonar repositorio remoto:

1. Crear carpeta y entrar a ella con **mkdir <**carpeta**>** y **cd <**carpeta**>**

2. **git clone <**url SSH/HTTPS**>**

25. Flujo de trabajo profesional: Haciendo merge de ramas de desarrollo a main

• Agregar archivo en carpeta:

**git add <**carpeta/file.ext**>**

26. Flujo de trabajo profesional con Pull requests

• En un entorno profesional normalmente se bloquea la rama **main**, y para enviar código a dicha rama pasa por un **code review** y luego de su aprobación se unen códigos con los llamados **merge request**.

• Funciona así:

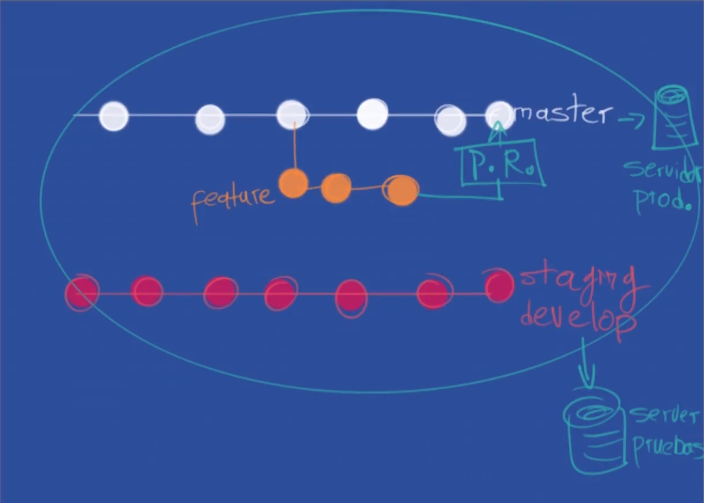
1. Se crea una rama para una nueva característica y se desarrolla en el entorno local.

2. Se hace un **pull request** al **staging develop** (servidores de pruebas) para su revisión y aprobación por otros miembros del equipo.

3. Al ser aprobados se autoejecuta el **merge** al **staging develop**. En teoría este y **main** siempre deben estar actualizados.

4. Una vez testeados los cambios, se hace un **pull request** a **main** para fusionarlos.

**NOTA**: Pull request es una característica exclusiva de GitHub aunque en otras plataformas como GitLab se llaman merge request.



27. Utilizando Pull Requests en GitHub

• Los **pull request** no existen en **Git**, solo los **merge**, un **pull request** es como una pausa justo antes de fusionarlo, es como trabajar en un **staging** del lado del servidor que te permite agregar cambios.

28. Creando un Fork, contribuyendo a un repositorio

• Clonar repositorio remoto del cual no se es parte del equipo:

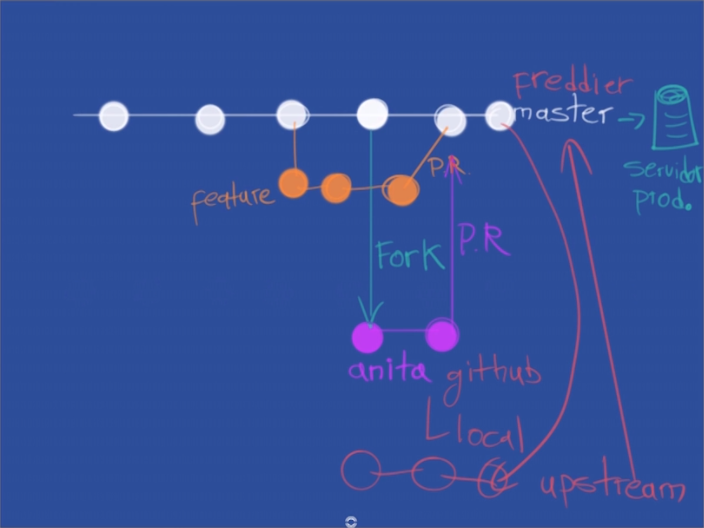
**git clone <**url HTTPS o SSH**>**

• Antes de hacer cambios siempre es buena práctica traer los cambios del remoto.

• Crear otra rama para traer los cambios más recientes del remoto, **NOTA**: **upstream** es un apuntador al main del remoto, al traerlo al local se fusionará con el local:

**git remote add upstream <**url HTTPS o SSH**>**

**git pull upstream main**



31. Ignorar archivos en el repositorio con .gitignore

• Para ignorar archivos que no se deban enviar al remoto, se crea el archivo **.gitignore** y ahí se indica que archivos se van a ignorar.

• Ignorar todos los archivos de determinado formato:

**\*.<**extensión**>**

• Comentarios en **.gitignore**:

**# <**Comentario**>**

• Para ignorar una carpeta en **.gitignore** solo basta con escribir el nombre de la carpeta.

• Excepciones:

**!/<**carpeta**>/<**carpeta ó file.ext**>**

32. Readme.md es una excelente práctica

• Editor online de **.md**:

<https://pandao.github.io/editor.md/en.html>

• No es buena práctica incluir imágenes en GitHub porque son binarios, lo mejor es subirlas a una plataforma de imágenes y referenciarlas al código con su url, aunque no hay problema si no son muy pesadas.

34. Git Rebase: reorganizando el trabajo realizado

• **Git Rebase** es agarrar los cambios realizados en una rama y ponerlos sobre otra, esto reescribe la historia de esa rama; **es una muy mala práctica enviarla a repositorios remotos**, solo es para locales.

• Hacer rebase, **NOTA**: Primero se hace en la rama que desaparecerá (donde se hicieron los cambios) y luego a la final:

**git rebase <**rama**>**

35. Git Stash: Guardar cambios en memoria y recuperarlos después

• Estando en una rama con modificaciones se quiere ir a otra, pero aún no se quiere hacer commit de esas modificaciones, así que se guardan en memoria con:

**git stash**

• Ver en que commit están los cambios stahseados:

**git stash list**

• Aplicar los cambios de un stash donde se requieran:

**git stash pop**

• Crear una rama y guardar un stash en ella:

**git stash branch <**rama**>**

• Borrar stash, **NOTA**: Si se tienen varios en una sola rama solo se borrará el más reciente:

**git stash drop**

36. Git Clean: limpiar tu proyecto de archivos no deseados

• Para eliminar archivos no deseados primero hay que ver cuales identifica Git como tales con, **NOTA**: No detecta carpetas, solo archivos, y tampoco los archivos de .gitignore:

**git clean --dry-run**

• Borrar los archivos listados, **NOTA**: Cuidado porque los borra definitivamente:

**git clean -f**

37. Git cherry-pick: traer commits viejos al head de un branch

• Traer cambios de un commit anterior de una rama, a otra con cambios más recientes, **NOTA**: El # del commit se ve resumido con **git log --oneline** en la rama en la que esta, y el comando se ejecuta a la rama en la que ira, pero, es mala práctica, porque reconstruye la historia de commits, así que es mejor hacer un merge aunque es útil para hotfixes:

**git cherry-pick <**# comit**>**

38. Reconstruir commits en Git con amend

• Agregar cambios que faltaron al commit más reciente:

1. Se agrega el o los archivos al staging con **git add <**file.ext**>** o **git add .** respectivamente

2. **git commit --amend -m “**mensaje**”**

39. Git Reset y Reflog: **NOTA**: Úsese en caso de emergencia

• Si se pierden archivos y se rompe todo y con el comando **git log --oneline** no se ve todo el historial, se usa este comando para VER TODO y extraer el # del commit al que se quiere restaurar usando **git reset <**# del commit al que se quiere regresar**> --hard**:

**git reflog**

40. Buscar en archivos y commits de Git con Grep y log

• Ver en qué archivos esta cierta palabra:

**git grep <**palabra**>**

• Ver en qué línea de que archivos está la palabra:

**git grep -n <**palabra**>**

• Ver cuantas veces esta repetida una palabra en los archivos, **NOTA**: Para buscar código se debe encerrar entre comillas “”, incluso código HTML:

**git grep -c <**palabra**>**

• Buscar una palabra en la historia de los commits:

**git log -S “**palabra**”**

41. Comandos y recursos colaborativos en Git y GitHub

• Ver la lista de commits, por mensaje, que ha hecho cada miembro del equipo:

**git shortlog**

• Ver la cantidad de ciertos commits por miembro:

**git shortlog -sn**

• Ver la cantidad de commits por miembro, incluso los borrados:

**git shortlog -sn --all**

• Ver la cantidad de commits por miembro, excepto los merges:

**git shortlog -sn --all --no-merges**

• Aplicar un alias a la configuración global del equipo, **NOTA**: Para ejecutarlo es con **git <**alias**>**:

**git config --global alias.<**alias**> “**comando sin la palabra git**”**

• Ver quien hizo que cambios en un archivo, **NOTA**: **-c** es para verlo con mejor formato:

**git blame -c <**file.ext**>**

• Ver el manual de cómo funciona un comando:

**git <**comando**> --help**

• Ver específicamente entre que líneas quien hizo que:

**git blame <**file.ext**> -L<**# de a partir de que línea**>,<**# de hasta que línea**> - c**

• Ver las ramas remotas desde el local:

**git branch -r**

• Ver las ramas locales y remotas:

**git branch -a**