Guía rápida de Comandos Git y GitHub

Verificar instalación de GIT y Versión:

$> git -v
$> git --version

Consultar configuración:

$> git config --list
$> git config user.name
$> git config user.email

Configurar Cuenta Global:

$> git config --global user.name "Nombre Usuario" →Ejm, git config --global user.name "Wilmer Gil"

$> git config --global user.email "Email" →Ejm, git config --global user.name "wilmer.gil@udea.edu.co"

Configurar Cuenta en Cada Repositorio:

$> git config user.name "Nombre Usuario" → Ejm, git config --global user.name "Wilmer Gil"

$> git config user.email "Correo" → Ejm, git config --global user.name "wilmer.gil@udea.edu.co"

Configurar llave SSH para conexión automática (solo en PC Personal):

$> ssh-keygen -t ed25519 -C "email"

1. Indicar al PC que se desea usar esa llave para la conexión SSH:

$> eval "$(ssh-agent -S)"

$> ssh-add ~/ .ssh/id_ed25519 → Esto si es la ruta por defecto, o si fue modificada al crear la clave, indicar dicha ruta.

2. Luego Copiar la llave para poderla pegar; para ello se usan los siguientes comandos de acuerdo al SO:

Linux:

$> cat ~/ .ssh/id_ed25519.pub

Windows:

$> clip < ~/ .ssh/id_ed25519.pub

Mac:

$> pbcopy < ~/ .ssh/id_ed25519.pub

3. Ir a la cuenta de GitHub en la nube, y buscar "cuenta → configuración → SSH y GPG Keys" y pegar la cuenta en la "Authentication Key"

4. Verificar la conexión:

$> ssh -T git@github.com

Configurar el repositorio (ubicarse en la carpeta donde estará el proyecto)

$> git init

Esto creará el proyecto y lo ubicará en la rama "main".

Pasar cambios del área de trabajo al área "stagging" (listos para hacer commit) → Comando Add:

$> git add index.html → agrega el archivo al stagging

$> git add *.js → agrega todos los .js al stagging

$> git add src/component-* → agrega todos los archivos que cominzan con "component-" de la carpeta "src"

$> git add . → agrega todos los archivos del directorio actual (pero no inclute subdirectorios)

$> git add -A → agrega todos los archivos con cambios en el proyecto (incluyendo subdirectorios(

$> git add -u → agrega todos los archivos modificados o eliminados (pero no los archivos nuevos o creados)

$> git reset → devuelve los cambios de Stagging a la zona de trabajo

$> git status → muestra el estado de los diferentes archivos y el área en la que se encuentran

$> git status -s → muestra el estado o área de los archivos pero "resumido"

Pasar cambios del área de "stagging" al "Repositorio" (Commit):

$> git commit → Genera una nueva versión en el repositorio. Luego del comando, agregar la descripción en el archivo de edición que se habilita.

$> git commit -m "mensaje" → Agrega un "mensaje" al commit

$> git log → Lista los commits del proyecto

$> git show idCommit → Muestra los cambios realizados en el commid con el id respectivo. Ejm: $> git show 06a3bab35ohj38wn30898dhjb

$> git commit --amend → Actualizar o adicionar cambios al último commit, incluso permitiendo editar la descripción.

Buenas prácticas para nombrar commits:

1. No más de 50 caracteres en el título
2. Separar el título de la descripción con una línea en blanco.

Ejm:

This is a example commit

Example of a commit of a course about avanced GitHub.

3. La descripción debería responder las preguntas "¿Qué?" y "¿Por qué?"
4. Escribir los verbos en imperativo:

Ejm: Fix header transition on mobile [Ok]
Fixed header transition on mobile [Wrong]

Sincronización (pasa los cambios y registros del repositorio local al repositorio remoto):

1. Crear el proyecto o repositorio en la nube (en Github.com).
2. Conectar ambos repositorios. GitHub indica los comandos necesarios en caso que el repositorio no se haya creado aún, o si ya existe. Para el segundo sería algo cómo:

$> git remote add origin git@github.com:nombreUsuario/nombreProyecto.git

$> git branch [en caso que la rama sea MAIN] ó $> git branch nombreRama → si va a agregarse a una distinta o se está en una que no es Main.

3. Subir los archivos al repositorio

$> git push -u origin main → Sube los archivos del repositorio local al remoto, y asocia la rama "main" a la rama "main" remota.

$> git push → actualizar los archivos en el repositorio remoto, ya no es necesario conectar la rama si ya están conectadas.

4. Actualizar el repositorio local con los cambios remotos:

$> git pull → Se trae los cambios remotos al repositorio local.

5. Clonar un proyecto (cuando se desea tener acceso a un proyecto existente y tener el repositorio local)

$> git clone direcciónProyecto → La "direcciónProyecto" se toma de gutHub.com y puede ser https, ssh o CLI. Se recomienda https si no hay conexión ssh.

Ejm: $> git clone git@github.com:nombeUsuario/nombreProyecto

Al hacerlo, se crea una carpeta con "nombreProyecto" en la carpeta actual donde se ejecutó el comando.

Creación y gestión de ramas:

1. Ver las ramas existentes:

$> git branch

2. Moverse entre ramas:

$> git switch nombreRama

3. Crear una rama:

$> git switch -c nombreRama

4. Borrar una rama

$> git switch -D nombreRama → Importante estar ubicado en una rama diferente a la que se va a eliminar

Mezcla o Merge entre Ramas:

1. Navegar a la rama en la cuál se agregarán o mezclarán los cambios. Por ejemplo, fusionar Develop en Main:

$> git switch nombreRamaQueUniraCambios → Ejm: $> git switch main

2. Ejecutar comando "merge" diciendo el nombre de la rama que se fusionará con la rama actual:

$> git merge nombreRamaAFusionar → Ejem: $> git merge develop → Esto fusiona la rama Develop en la rama Main

$> git merge --squash nombreRamaAFusionar → En caso que se desee hacer un "swuash merge"

3. Resolver conflictos

$> git checkout --ours nombrearchivo → Si desean conservarse los cambios que están en la rama "padre" (en la que se va a fusionar)

$> git checkout --theirs nombrearchivo → Si desean conservarse los cambios de la rama hija (la que se va a fusionar)

Después adicionar el archivo y hacer el commit:

$> git add nombrearchivo

$> git commit

4. Verificar "merge":

$> git log → Muetsra los cambios generales realizados

$> git diff → Observar los cambios en los archivos comparándolos

Pull Request (Solicitud para integrar nuestros cambios en una rama diferente - Solicitud para hacer y aprobar un merge):

- Se trabaja en el portal Web de GitHub
- Se selecciona la rama a la que se desea integrar los cambio y se compara con la rama en la que hemos realizado los cambios. Ejm:

base: main ← Compare: develop (Busca hacer la solicitud de integrar "develop" en "main")

- Se le crea un título y una descripción en el Pull Request
- Revisar los Pull Request y si se aprueban, aceptar la opción "Merge Pull request".

Recomendaciones en la descripción de un Pull Request:

- Explicar qué se quiere lograr con los cambios
- Tener mensajes de Commit descriptivos
- Agregar enlace al ticket/issue
- Agregar capturas de pantalla
- Mantener un delta inferior a 200 nuevas líneas de código

En los Pull Request se puede hacer la "Code Review" para que alguien apruebe, comente o sugiera cambios en los Pull Request.

Se recomienda que todo el equipo realice "Code Review". Hacerlo usando plantillas escritas en lenguaje "MarkDown".

Guardar temporalmente cambios para retomarlos más adelante (Stash)

$> git stash → Guarda en una pila los cambios realizados y retoma la versión anterios hasta el último "staging" sin considerar archivos nuevos

$> git stash save "nombrePila" → Guarda en una pila los cambios realizados bajo el nombre "nombrePila" y retoma la versión anterior hasta el último "staging"

$> git stash --include-untracked → Guarda en una pila los cambios realizados y retoma la versión anterior pero incluyendo archivos recién creados

$> git stash list → Muestra la pila de cambios guardados con un índice específico, como por ejemplo: "stash@{1}"

$> git stash pop → Retoma los últimos cambios almacenados en la pila

$> git stash pop nombreíndice → Retoma los cambios almacenados en el índice especificado; Ejm: $> git stach pop stash@{1}

Ignorar Archivos para los Commits:

1. Crear el archivo gitignore en la raíz del proyecto llamado: ".gitignore"
2. Indicar qué archivos, carpetas o tipos de archivos se van a ignorar. Ejemplo:

```
# gitignore file
node_modules/          # Ignorar toda la carpeta "node_modules"
yarn-error.log         # Ignorar el archivo "yarn-error.log"

*.zip                  # Ignorar todos los archivos con terminación .zip en la carpeta actual
**.zip                 # Ignorar todos los archivos con terminación .zip en la carpeta actual y subcarpetas
node_modules/**,zip    # Ignorar todos los archivos con terminación .zip en la carpeta "node_modules" y en sus subcarpetas
```

Revertir cambios realizados en un commit específico:

$> git log --oneline → Muestra el historia de commits resumidos en una línea y con un Id para cada uno

$> git show idCommit → Muestra el detalle de los cambios en el commit con idCommit; Ejm: $> git show c033393

$> git revert idCommit → Revierte los cambios hechos en el commit con idCommit

Lo anterior crea un nuevo commit donde se revierten los cambios del commit respectivo.

Regresar a un estado anterior del repositorio:

$> git reset --soft idCommit → Devuelve el repositorio al estado del Commit con idCommit, pero conserva los cambios en Stage por si desean retomarse

$> git reset idCommit → Devuelve el repositorio al estado del Commit con idCommit, pero conserva los cambios en el área de trabajo por si desean retomarse

$> git reset --hard idCommit → Devuelve el repositorio al estado del Commit con idCommit sin conservar cambios

Sincronizar dos ramas a un punto específico (por ejemplo, actualizar la rama "Develop" con el estado actual de la rama "Main":

$> git switch develop → Ubicarme en la rama que quiero sincronizar con otra

$> git rebase main → "Rebasar" o sincronizar la rama actual (develop) con la rama main u otra rama.

Copiar un Commit de otra rama a la rama actual:

$> git switch develop → Nos ubicamos en la rama a la cuál le queremos copiar el commit de otra rama

$> git cherry-pick idCommit → Copia en la rama actual (develop) el commit que había en otra rama

Workflows Conocidos:

Featured - Based:

→ Se crean ramas por cada Feature

→ Antes de hacer el Merge con la rama principal, se revisan los cambios en la Feature a través del trabajo colaborativo (otros compañeros revisan)

GitFlow:

→ Maneja una rama principal con la versión estable del sistema (main)

→ Maneja una rama secundaria (develop) con una versión de desarrollo y en la cual se pueden crear otras ramas tipo "Featured-Based".

→ Puede hacerse el "merge" en el "main" antes de la versión estable pero en la Release. Sería un "Release Branches".

→ Si surge un error crítico en "main" (rama de producción) se resuelve con una "Hotfix branch"; se crea la rama desde main, se resuelve y luego se sincroniza con Develop.

Palabras clave para nombrar ramas en este tipo de WorkFlow:

- Feature: Hace referencia a una nueva funcionalidad
- Fix: arreglar algún error
- Release: La rama contiene una versión completa y estable del código
- HotFix: Cambio inmediato en la rama Main por ser un error en producción

Si se tiene una numeración de las tareas, features o HU, el nombre de la rama debería incluirlo.

Ejm:

- fix-213/connect-error
- feature-456/add-whatsapp-button
- 442-fix-footer-links

Trunk Based:

→ Maneja una rama principal sobre la que se agregan los cambios

→ Se crean ramas alternas para ciclos muy cortos y aspectos muy específicos, y apenas se revisa y testea se fusiona con la rama principal.

Issues (manejo de incidencias, errores, tareas y demás en un proyecto de GitHub):

- Darle seguimiento a una tarea
- Reportar un bug en la aplicación
- Compartir feedbak sobre un feature
- Hacer una pregunta sobre el funcionamiento del código.
- En GitHub (versión Web) pueden asociarse los Pull Request a los issues haciendo mención del issue correspondiente a través del número o identificador de éste, el cuál se sugiere luego de crear el Issue y poniendo la tecla "#" (numeral) en la descripción del Pull.

Documentar en Git:

- Las descripciones y demás elementos de texto pueden tener formato sugerido por el lenguaje "Markdown" (Usa etiquetas similares al html).

→ h1
→ h2
→ h3
[ ] → casilla verificación
- → Viñeta para lista
[nombreEnlace] (https://enlace.com) → Vínculos
Y tiene opciones para subrayados, negritas, ...

- Informar y registrar el proyecto en el archivo Readme.md (extensión de Markdown). Agregar allí:
  - Título
  - Descripción
  - Cómo instalar y ejecutar el proyecto
  - Cómo utilizar el proyecto