Control de Versiones Git, GitHub y Sourcetree

CUC
Colegio Universitario de Cartago

Agenda

- ¿Que es Git?
- Sistemas de control de versiones
- Instalación Git
- ¿Qué es GitHub?
- Creación y configuración de la cuenta GitHub
- Instrucciones básicas en Git
- Instalación GitHub Desktop y Sourcetree
- Practicas

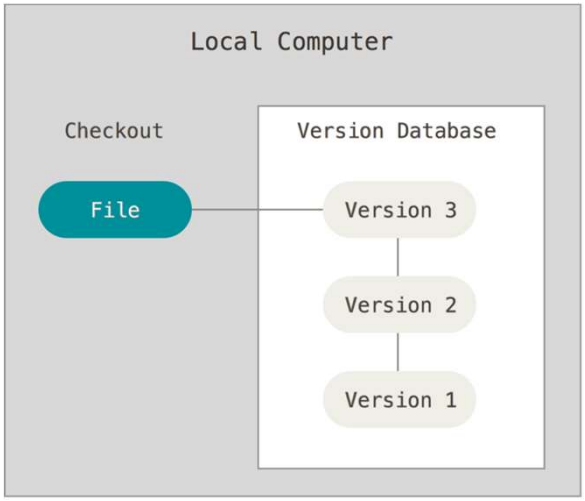
¿Que es Git?

- Git es un **sistema de control de versiones**. Fue creado por Linus Torvalds en 2005 y mantenido por Junio Hamano desde entonces.

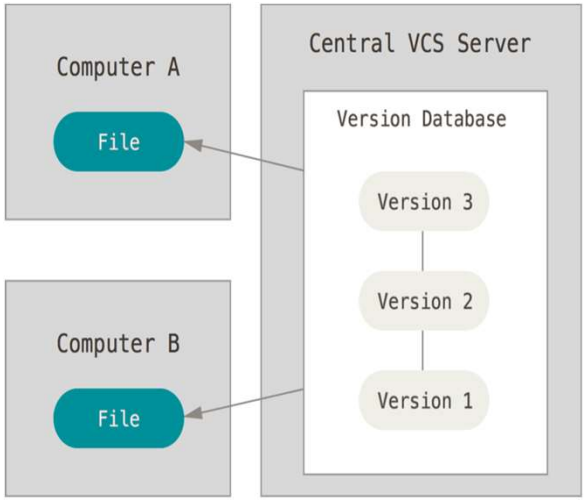
- Un sistema control de versiones (**VCS** por sus siglas en inglés) es un sistema que registra los cambios realizados en un archivo o conjunto de archivos a lo largo del tiempo, de modo que puedas recuperar versiones específicas más adelante.

- Si sos *desarrollador de software, diseñador gráfico o de web, científico de datos, analista de datos, ingeniero de datos* o cualquier perfil que utilice muchos archivos y quieres mantener cada versión de una imagen, diseño, script o archivo (es algo que sin duda vas a querer), usar un sistema de control de versiones es una decisión muy acertada. Dicho sistema **permite regresar a versiones anteriores de tus archivos**, regresar a una versión anterior del proyecto completo, comparar cambios a lo largo del tiempo, ver quién modificó por última vez algo que pueda estar causando problemas, ver quién introdujo un problema y cuándo. Usar un VCS también significa generalmente que, **si arruinas o pierdes archivos, será posible recuperarlos fácilmente**

Tipos de sistemas de control de versiones

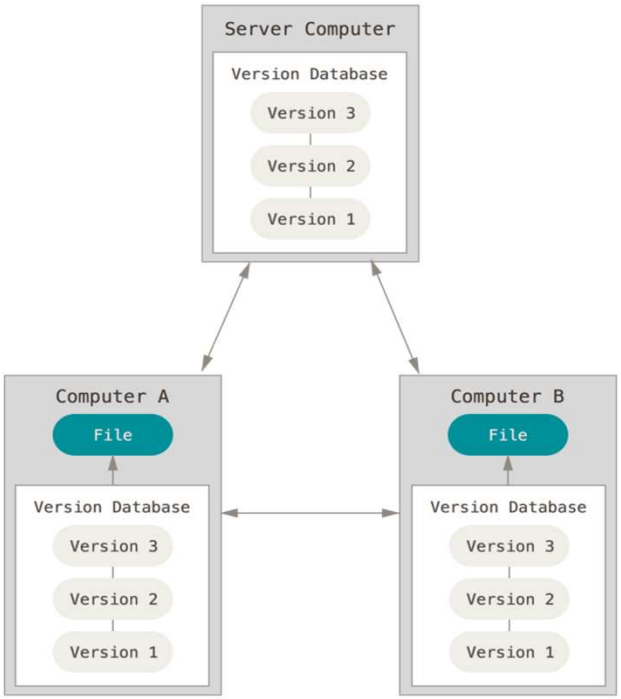


Sistemas de Control de Versiones Locales:
RCS



Sistemas de Control de Versiones Centralizados:

Subversion, TFS

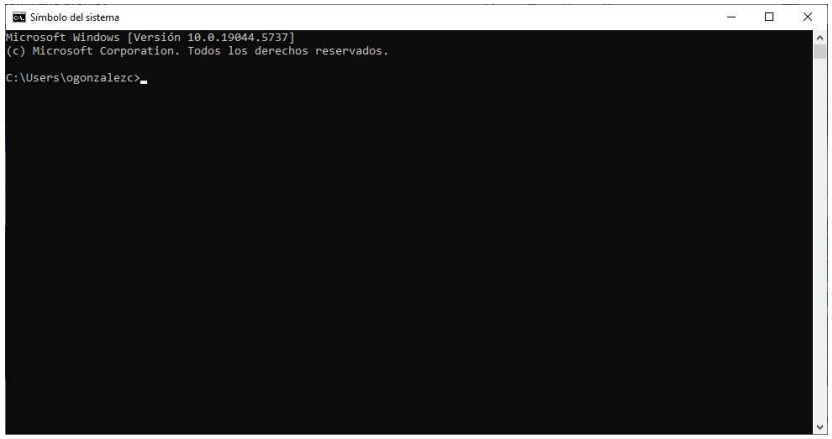


Control de versiones distribuido:
Git, Mercurial, Bazaar o Darcs

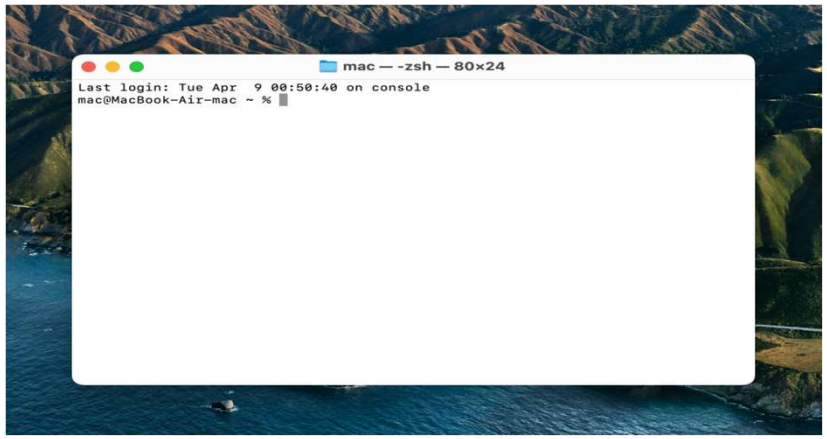
CUC
Colegio Universitario de Cartago

Línea de comandos

Existen muchas formas de usar **Git**. Por un lado, tenemos las herramientas originales de **línea de comandos**, y por otro lado tenemos una gran variedad de **interfaces de usuario** con distintas capacidades. En esta clase vamos a utilizar Git desde la línea de comandos e interfaces graficas. La línea de comandos es el único lugar en donde puedes ejecutar todos los comandos de Git - la mayoría de las interfaces gráficas de usuario solo implementan una parte de las características de Git por motivos de simplicidad. Si conoces cómo realizar algo desde la línea de comandos, seguramente podrás averiguar cómo hacer lo mismo desde una interfaz gráfica. Sin embargo, la relación opuesta no es necesariamente cierta. Así mismo, la decisión de qué cliente gráfico utilizar depende totalmente del gusto, pero todos tendrán las herramientas de línea de comandos instaladas y disponibles.



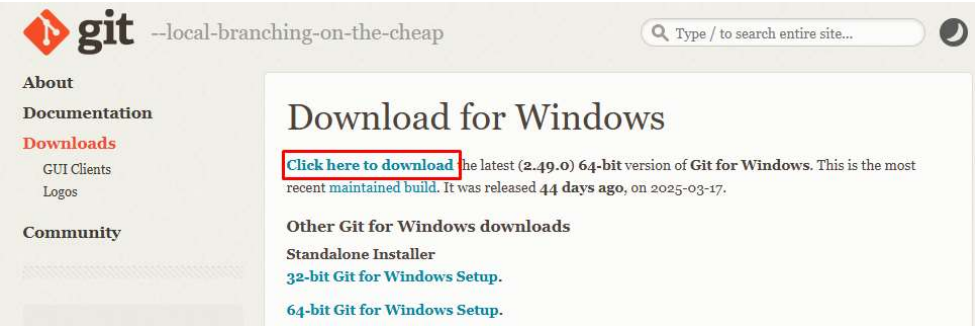
Terminal de Windows
Windows + R, escribir cmd y pulsar Enter



Terminal de Mac
CMD + Espacio para abrir Spotlight, luego escribir "Terminal" y presionar Enter

Instalación de Git

- En Windows: https://git-scm.com/downloads/win



- En Mac: https://git-scm.com/downloads/mac

- En Linux: https://git-scm.com/downloads/linux

- Opcional Instalación de terminal

Adicionalmente, si gustan pueden instalar una terminal con mas funciones como es:
tabby terminal: https://tabby.sh/ ó una terminal con IA como warp: https://www.warp.dev/

¿Que es GitHub?

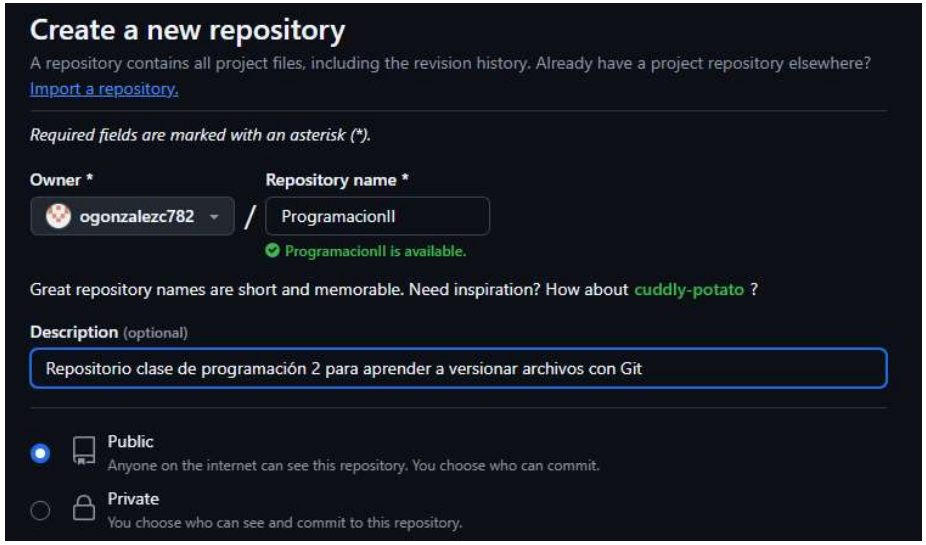
GitHub es un proveedor de alojamiento de **repositorios Git**, es el punto de encuentro para que millones de desarrolladores colaboren en el desarrollo de sus proyectos. Un gran porcentaje de los repositorios Git se almacenan en **GitHub**, y muchos proyectos de código abierto lo utilizan para hospedar su Git, realizar su seguimiento de fallos, hacer revisiones de código y otras cosas.

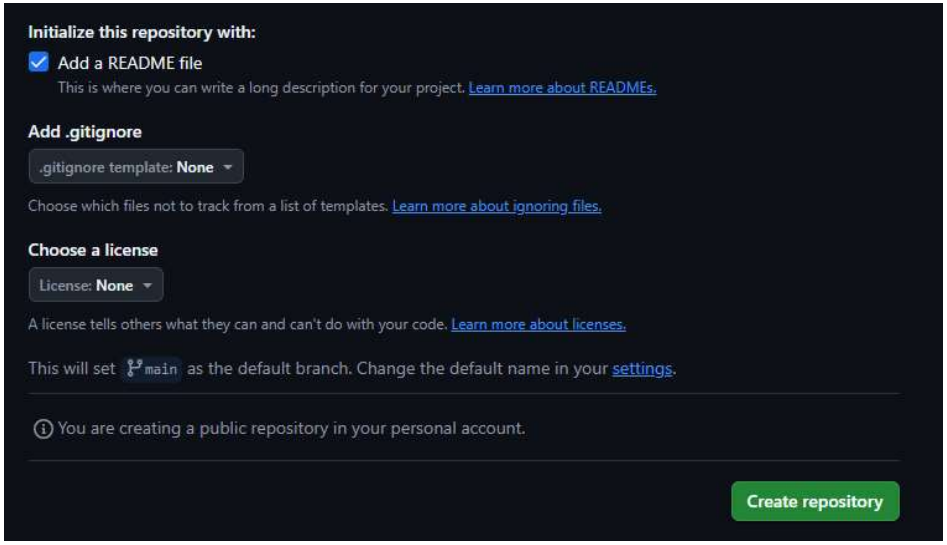
Existen otra gran variedad de repositorios que usan Git, como por ejemplo *GitLab, AWS CodeCommit, Bitbucket, Azure Repos*, entre otros. Sin embargo, nos vamos a enfocar en GitHub.

Creación y configuración de la cuenta

Lo primero es crear una cuenta de usuario gratuita https://github.com, elige un nombre de usuario que no esté ya en uso, proporciona un correo y una contraseña, y pulsa el botón verde grande "**Sign up for GitHub**".

Después creamos un repositorio llamado "**ProgramacionII",** escribimos una descripción del repositorio **"Repositorio de clase de programación 2 para aprender a versionar archivos con Git",** dejamos **"Public",** marcamos **"Add a README file"** y le hacemos clic al botón **"Create repository"**





Instrucciones básicas en Git (version/config)

Para comenzar a usar Git, primero abriremos nuestra Terminal de comandos.

Para Windows, puedes usar Git bash ó terminal de Windows de preferencia. Para Mac y Linux, puedes usar la terminal integrada.

Lo primero que debemos hacer es comprobar si Git está correctamente instalado: `git --version`
Si Git está instalado, debería mostrar algo como **git version xxxx**

Configurar Git

```
git config --global user.name "Osvaldo Gonzalez Chaves"
git config --global user.email "ogonzalezc782@gmail.com"
```

Cambia el nombre de usuario y la dirección de correo electrónico por los tuyos. Mas adelante también los usaremos para registrarnos en GitHub.

Nota: Usa `--global` para configurar el nombre de usuario y el correo electrónico para **cada repositorio** en su computadora.
Si desea configurar el nombre de usuario/correo electrónico solo para el repositorio actual, puede eliminar `--global`
Para consultar la configuración global actual de Git: `git config --list`

Ahora, creemos una nueva carpeta para nuestro proyecto:

```
mkdir progamacionII
cd progamacionII
```

mkdir **crea** un nuevo directorio
cd **cambia** el directorio de trabajo actual

Instrucciones básicas en Git (init/status)

Inicializar Git

Una vez que haya navegado a la carpeta correcta, puede inicializar Git en esa carpeta:

```
git init
Initialized empty Git repository in C:/programacionII/.git/
```

Nota: Git ahora sabe que debe vigilar la carpeta en la que lo iniciaste.
Git crea una carpeta oculta para realizar un seguimiento de los cambios.

Acabamos de crear el primer Repo Git local. Pero está vacío.
Así que agreguemos algunos archivos o creemos un nuevo archivo usando algún editor de texto. Luego guárdelo o muévalo a la carpeta que acaba de crear.

Volvamos al terminal y enumeremos los archivos en nuestro directorio de trabajo actual:

```
ls
Index.html
Practica.py
```

Luego revisamos el git `status` y mira si es parte de nuestro repositorio:

```
git status
On branch master
No commits yet
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        Practica.py
        index.html
nothing added to commit but untracked files present (use "git add" to track)
```

Instrucciones básicas en Git(staging enviroment/add)

Ahora Git es consciente de los archivos, pero no aun no lo agrega a nuestro repositorio.

Los archivos en su carpeta de repositorio de git pueden estar en uno de los 2 estados:

- **Tracked**(rastreado): archivos que Git conoce y se agregan al repositorio
- **Untracked**(sin rastrear): archivos que están en su directorio de trabajo, pero no se agregan al repositorio

Cuando agrega archivos por primera vez a un repositorio vacío, todos están **untracked**. Para que Git los rastree, debe organizarlos o agregarlos al área de preparación(**staging enviroment**).

Mientras trabaja, puede agregar, editar y eliminar archivos. Pero siempre que alcance un hito o termine una parte del trabajo, debe agregar los archivos a un **staging enviroment**.

Git: agregar nuevos archivos

Ahora tenemos archivos que están listos para ser comprometidos(**commit**) hacia repositorio en el que está trabajando(lo veremos en la siguiente diapositiva) pero primero debemos agregarlos al **staging enviroment:**

```
git add index.html
```

Para agregar mas de un archivo a nuestra carpeta de trabajo podemos usar:

```
git add --all ó git add .
```

Al usarlo --all en lugar de nombres de archivos individuales se mostrarán el `stage` de todos los cambios (nuevos, modificados y eliminados) de los archivos.

Nota: El comando abreviado para `git add --all` es `git add -A`

Instrucciones básicas en Git(commit/log)

Git Commit

Estamos listos para movernos de **stage** a **commit** para nuestro repositorio.

Agregar compromisos(**commits**) realizará un seguimiento de nuestro progreso y cambios mientras trabajamos.

Git considera cada **commit** como un "punto de guardar". Es un punto en el proyecto. Puede devolverse si encuentra un error o si desea hacer un cambio.

Cuando hacemos un **commit**, deberíamos incluir siempre un mensaje .

Se debe agregar mensajes claros a cada **commit**, asi es mas sencillo para ver lo que ha cambiado y cuándo. Ver estándares y convenciones de commits para los proyectos: https://www.conventionalcommits.org/en/v1.0.0/#summary

```
git commit -m “feat:Primer lanzamiento de Hello World y script de python”
```

Git Commit log

Para ver el historial de **commits** para un repositorio, podemos usar el comando `log`

```
git log
commit f0de3c16996fed8fcbbc691879b9781865851bca (HEAD -> master)
Author: Osvaldo Gonzalez Chaves <ogonzalezc782@gmail.com>
Date:   Wed Apr 30 00:20:20 2025 -0600

    Primer lanzamiento de Hello World y script de python
```

CUC
Colegio Universitario de Cartago

Instrucciones básicas en Git(help)

Ayuda de Git

Si tienes problemas para recordar comandos u opciones de comandos, puedes usar Git `help`

Hay un par de formas diferentes en las que puedes usar el help comando en la línea de comandos:
`git` *command* `-help`: Ver todas las opciones disponibles para el comando específico
`git help --all`: Ver todos los comandos posibles

Ejemplos:

```
git add -help
git help --all
```

Nota: Si se encuentra atascado en la vista de lista, **SHIFT + G** salta al final de la lista y luego **q** sale de la vista.

Instrucciones básicas en Git(remote/push)

Ahora, vamos a realizar el **enlace** de nuestra carpeta que esta local hacia el **repositorio remoto de Github** con este comando:

```
git remote add origin https://github.com/ogonzalezc782/ProgramacionII.git
```

Para realizar la **subida de los archivos** al repositorio remoto de GitHub los hacemos con un **push:**

```
git push --set-upstream origin master
```

Observe que la instrucción realiza lo siguiente:

- ✓ Subirá tu rama local **master** al repositorio remoto.
- ✓ Creará la rama origin/master en GitHub (si no existe).
- ✓ Establecerá la relación de seguimiento (**upstream**) para que en adelante puedas simplemente hacer git push sin parámetros adicionales.

En caso de no existir la rama **master** en GitHub, existen varias alternativas:

1. Renombrar la rama master a **main** y **hacer push:**

```
git branch -m master main
git push --force-with-lease --set-upstream origin main
```

2. Crear una rama **master** en Github.

```
git remote set-head origin master
git push --set-upstream origin master
```

Instrucciones básicas en Git(branch)

En git, un **branch(rama)** es una versión **nueva/separada** del repositorio principal(**master ó main**).

- Las ramas permiten trabajar en diferentes partes de un proyecto sin afectar la **rama principal.**
- Cuando el trabajo es completado, una rama puede ser mezclada(**merge**) con la rama principal.

Git branch
Crear rama nueva:

```
git branch feature/nueva-funcion-maximo
```

Lista las ramas

```
git branch
```

Nos **trasladamos** de espacio de trabajo de **master** a la **nueva rama**

```
git checkout feature/nueva-funcion-maximo
```

Practica

Sobre la rama **feature/nueva-funcion-maximo** realice lo siguiente:

1. Sobre el archivo Practica.py agregue una funcion en Python que retorne el máximo de 3 valores suministrados por parámetro.
2. Guarde el archivo y realice los pasos necesarios para **subir los cambios** a la rama **feature/nueva-funcion-maximo**

¿Cuales fueron las instrucciones de Git que utilizaron?

Instrucciones básicas en Git(merge)

Una vez realizados los cambios en la rama **feature/nueva-funcion-maximo**, el siguiente paso es realizar el **merge** con la rama principal master ó main.

Primero nos trasladamos a la rama master:

```
git checkout main
```

Switched to branch 'main'

Ahora, realizamos el **merge** de la rama actual **master** con **feature/nueva-funcion-maximo**

```
git merge feature/nueva-funcion-maximo
```

Después de realizar merge, existe la posibilidad de eliminar las ramas locales, ya que no serán utilizadas a menos que se requiera solucionar un conflicto con el merge.

```
git branch -d feature/nueva-funcion-maximo
```

Estrategias de merge (**Resolución de conflictos**)

- https://www.datacamp.com/es/tutorial/git-merge
- https://www.geeksforgeeks.org/git/merge-strategies-in-git/
- https://git-scm.com/docs/merge-strategies

Instrucciones básicas en Git(pull)

Cuando trabaja en equipo en un proyecto, es importante que todos se mantengan al día. Cada vez que comience a trabajar en un proyecto, debe obtener el más reciente cambio a su copia local.

Con git, puedes hacer eso con `pull`

```
git pull origin
Already up to date.
```

Así es como se mantiene el Git local actualizado desde un repositorio remoto.

Instrucciones básicas en Git(resumen de comandos para dev)

Estos son los comandos básicos si ya existe un repositorio creado en Github o cualquier repositorio.

1. Clonar el repositorio

```
git clone “ruta del repositorio”
```

2. Agregar al stage archivos nuevos o modificados

```
git add .
```

3. Comprometer los cambios

```
git commit -m “mensaje”
```

4. Subir cambios al repositorio central

```
git push
```

5. Actualizar la rama con cambios del servidor central

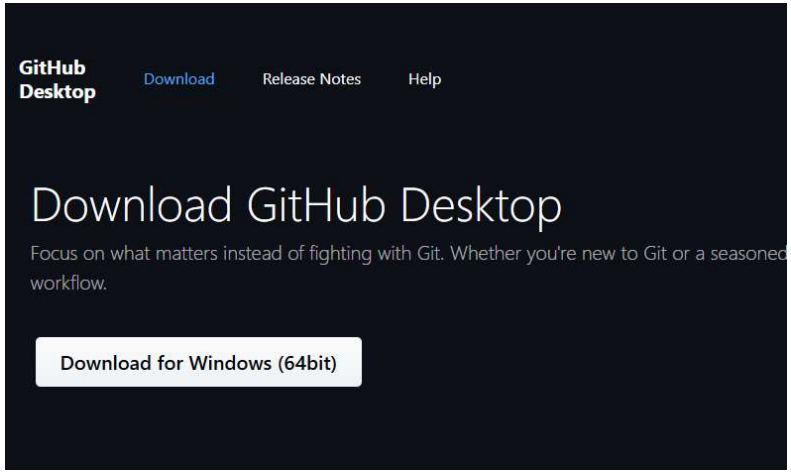
```
git pull
```

Para validar el estado de la rama usar(se puede usar en cualquier comento)

```
git status
```

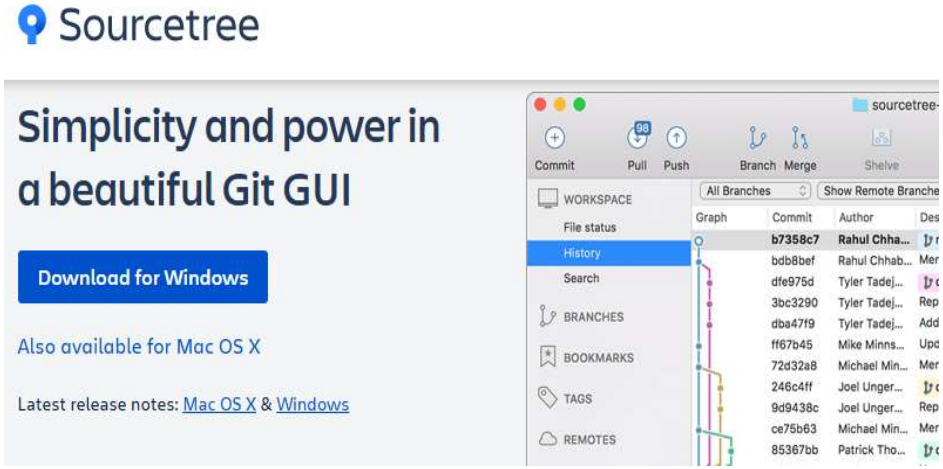
Instalación GitHub Desktop y Sourcetree

https://desktop.github.com/download/



- Interfaz sencilla y minimalista.
- Integración directa con GitHub (clonar, crear repos, hacer pull requests).
- Muestra visualmente los cambios, commits, ramas, y conflictos.
- Buena opción para principiantes o quienes no desean usar línea de comandos.

https://www.sourcetreeapp.com/



- Soporta repositorios locales y remotos(GitHub, Bitbucket, GitLab etc).
- Visualización muy completa del historial de ramas (árbol de commits).
- Manejo detallado de staging, resolución de conflictos, stashing, rebase, cherry-pick.
- Herramienta más avanzada que GitHub Desktop, pensada para usuarios intermedios/avanzados.