Tutorial Git y GitHub

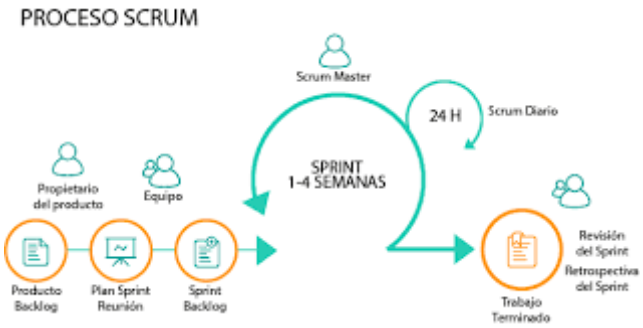
Unidad 2

Contenido

Metodología Ágil

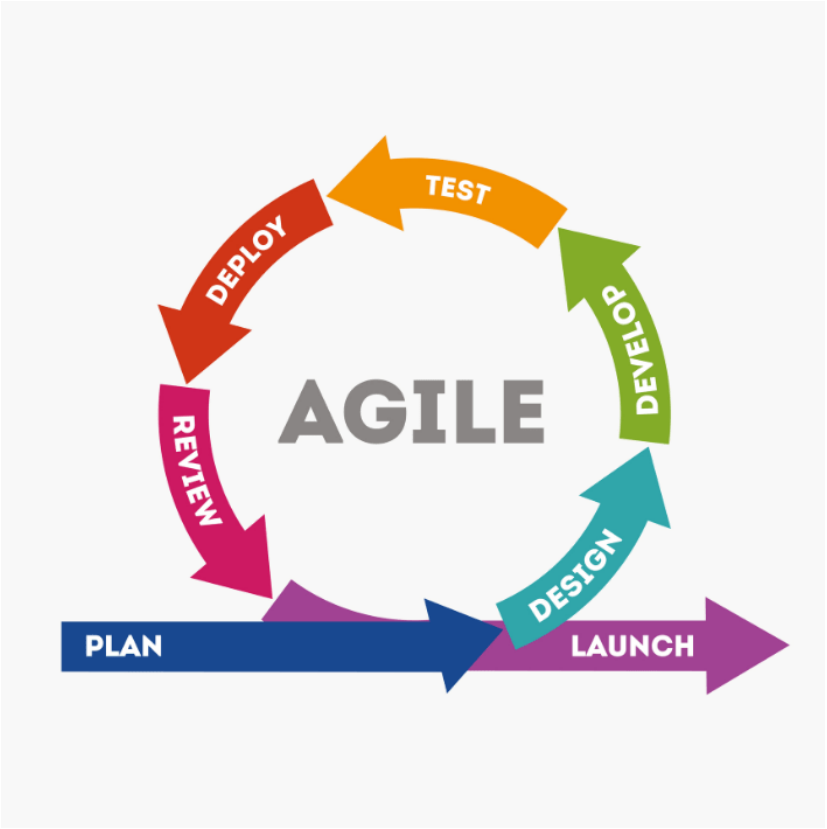
Ágil

Agile es un enfoque iterativo para la gestión de proyectos y el desarrollo de software que ayuda a los equipos a entregar valor a sus clientes más rápido y con menos dolores de cabeza. Las metodologías ágiles son inmensamente populares en la industria del software, ya que permiten a los equipos ser inherentemente flexibles, bien organizados y capaces de responder al cambio.



Desarrollo de Software

El desarrollo de software se refiere al diseño, la documentación, la programación, las pruebas y el mantenimiento continuo de un producto de software. La combinación de estos pasos se utiliza para crear una canalización de flujo de trabajo, una secuencia de pasos que, cuando se siguen, producen entregas de software de alta calidad. Esta canalización se conoce como el ciclo de vida del desarrollo de software.



Pasos en el ciclo de vida de desarrollo de software.

Descubrir

Los proyectos se visualizan, diseñan y priorizan. Una metodología como ágil ayuda a guiar el proceso de trabajo del proyecto.

Plan

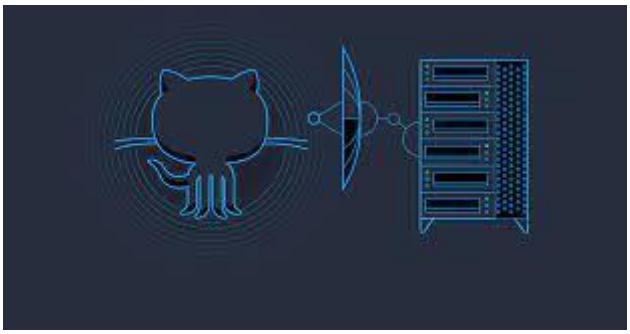
Se identifican las partes interesadas, se establecen los presupuestos y se solicita la infraestructura.

Construir y probar

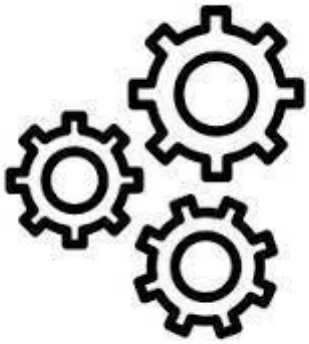
Los equipos de desarrollo trabajan para crear software listo para la producción que cumpla con los requisitos y los comentarios

Desplegar

Con el código escrito, probado y fusionado, es hora de enviarlo.



Funcionar



Se requiere soporte y mantenimiento para los proyectos de software activos

Observar

La gestión de incidentes es cuando Desarrollo y Operaciones responden a eventos no planificados y restauran servicios utilizando métodos confiables para priorizar incidentes y llegar a una resolución rápida.

Control de Versiones

El control de versiones, también conocido como "control de código fuente", es la práctica de rastrear y gestionar los cambios en el código de software. Los sistemas de control de versiones son herramientas de software que ayudan a los equipos de software a gestionar los cambios en el código fuente a lo largo del tiempo.

El software de control de versiones realiza un seguimiento de todas las modificaciones en el código en un tipo especial de base de datos. Si se comete un error, los desarrolladores pueden ir hacia atrás en el tiempo y comparar las versiones anteriores del código para ayudar a resolver el error, al tiempo que se minimizan las interrupciones para todos los miembros del equipo.

Ventajas

- ✓ Un completo historial de cambios a largo plazo de todos los archivos.
- ✓ Creación de ramas y fusiones.
- ✓ Trazabilidad. Ser capaz de trazar cada cambio que se hace en el software y conectarlo con un software de gestión de proyectos.

Git

Hoy en día, Git es, con diferencia, el sistema de control de versiones moderno más utilizado del mundo. Git es un proyecto de código abierto maduro y con un mantenimiento activo que desarrolló originalmente Linus Torvalds, el famoso creador del kernel del sistema operativo Linux, en 2005.

Git, que presenta una arquitectura distribuida, es un ejemplo de DVCS (sistema de control de versiones distribuido, por sus siglas en inglés).

Además de contar con una arquitectura distribuida, Git se ha diseñado teniendo en cuenta el rendimiento, la seguridad y la flexibilidad.

Rendimiento

Las características básicas de rendimiento de Git son muy sólidas en comparación con muchas otras alternativas. La confirmación de nuevos cambios, la ramificación, la fusión y la comparación de versiones anteriores se han optimizado en favor del rendimiento.

Seguridad

Git se ha diseñado con la principal prioridad de conservar la integridad del código fuente gestionado. El contenido de los archivos y las verdaderas relaciones entre estos y los directorios, las versiones, las etiquetas y las confirmaciones, todos ellos objetos del repositorio de Git, están protegidos con un algoritmo de hash criptográficamente seguro llamado "SHA1".

Flexibilidad

Uno de los objetivos clave de Git en cuanto al diseño es la flexibilidad. Git es flexible en varios aspectos: en la capacidad para varios tipos de flujos de trabajo de desarrollo no lineal, en su eficiencia en proyectos tanto grandes como pequeños y en su compatibilidad con numerosos sistemas y protocolos.

Instalación y Configuración de git en Windows

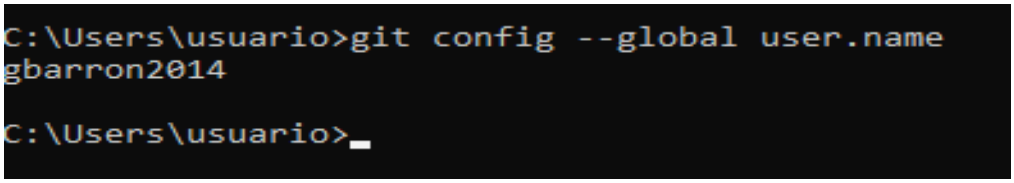
Paso 1. Descárgate el instalador de Git para Windows más reciente.

Paso 2. Cuando hayas iniciado correctamente el instalador, deberías ver la pantalla del asistente de configuración de Git. Selecciona las opciones Next (Siguiente) y Finish (Finalizar) para completar la instalación. Las opciones predeterminadas son las más lógicas en la mayoría de los casos.

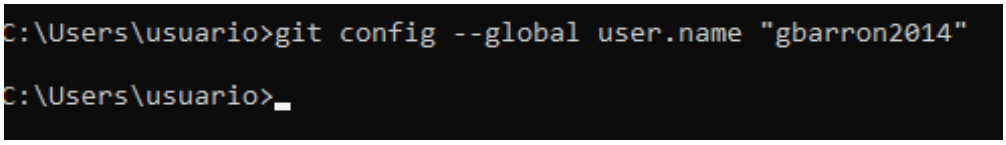
Paso 3. Abre el símbolo del sistema (o Git Bash si durante la instalación seleccionaste no usar Git desde el símbolo del sistema de Windows).

Paso 4. Ejecuta los siguientes comandos para configurar tu nombre de usuario y correo electrónico de Git (sustituye el nombre que aparece en el código por el tuyo). Esta información se asociará a todas las confirmaciones que crees:

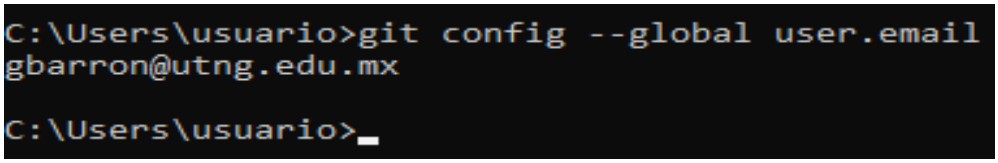
a. Conocer el usuario Git



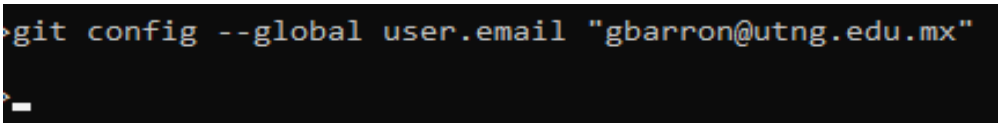
b. Modificar el usuario



c. Conocer el email

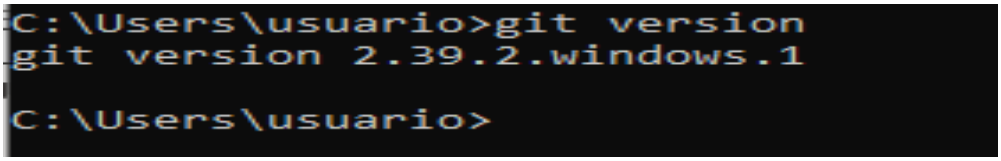


d. Modificar el email



e.

Paso 5. Conocer la versión



Repositorio

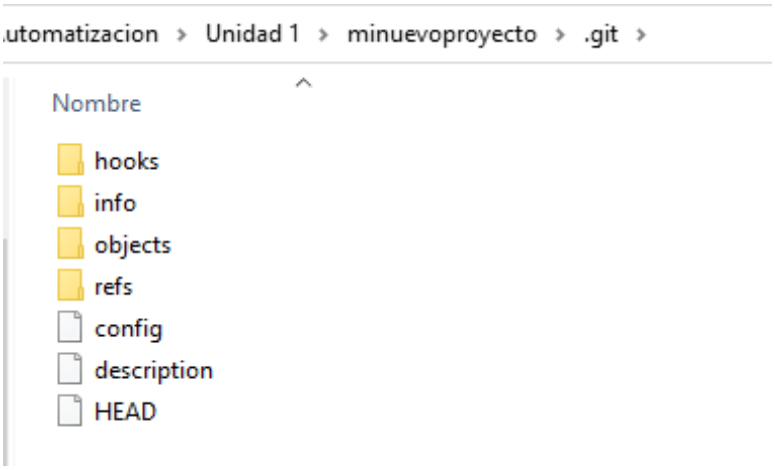
Un repositorio de Git es un almacenamiento virtual de tu proyecto. Te permite guardar versiones del código a las que puedes acceder cuando lo necesites.

Instrucciones

a. Crear una nueva carpeta llamada **minuevoproyecto**
b. Cambiarse al repositorio **minuevoproyecto**
c. Teclear el comando **git init**

```
C:\materias\Automatizacion\Unidad 1\minuevoproyecto>git init
Initialized empty Git repository in C:/materias/Automatizacion/Unidad 1/minuevoproyecto/.git/
```

d. Se crea una carpeta oculta donde se almacenarán todos los cambios hechos al proyecto.



Práctica 01 Creando un repositorio vacío

✓ Crea un repositorio vacío en cualquier lugar de tu computadora.

e. Teclear el comando **git status**

```
C:\materias\Automatizacion\Unidad 1\minuevoproyecto>git status
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)
```

Práctica 02 Crear varios archivos dentro del directorio de proyecto

- ✓ Crear algunos archivos dentro de tu directorio de trabajo y checar el estado del directorio de trabajo

Área de preparación

En términos técnicos, el área de preparación es el término medio entre lo que ha hecho con sus archivos (también conocido como el directorio de trabajo) y lo que ha confirmado por última vez (la confirmación HEAD). Como su nombre lo indica, el área de preparación le brinda espacio para preparar (escenificar) los cambios que se reflejarán en la próxima confirmación.

f. Teclear el comando **git add README.md** y verificar el status

```
C:\materias\Automatizacion\Unidad 1\minuevoproyecto>git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   README.md
```

Nota: para que quitar el archivo fuera del Stage Area teclea el siguiente comando: **git rm –cached README.md**

Commit

El comando **git commit** guarda una instantánea de los cambios por etapas actuales en el proyecto. Las instantáneas comprometidas son versiones "seguras" de un proyecto que Git nunca modificará a menos que se lo solicite específicamente.

Para guardar los cambios ejecuta el siguiente comando: **git commit -m “Version 1”**

```
C:\materias\Automatizacion\Unidad 1\minuevoproyecto>git commit -m "Versión Inicial"
[master (root-commit) 0fc54f1] Versión Inicial
 1 file changed, 1 insertion(+)
 create mode 100644 README.md
```

Ignorando archivos

Git ve cada archivo de tu copia de trabajo de una de las siguientes maneras:

- ✓ **Con seguimiento**: Un archivo que se ha preparado o confirmado previamente.
- ✓ **Sin seguimiento**: Un archivo que no se ha preparado o confirmado.
- ✓ **Ignorado**: Un archivo que se le ha indicado explícitamente a Git que ignore.

Crear el archivo .gitignore en directorio raíz del proyecto

COPY CON .gitignore

Editar el archivo y al finalizar CTRL + Z (Windows)

git add .gitignore

git commit

Para una mayor referencia de patrones véase Tutorial .gitignore

Checando Logs e Hostoria

El propósito de todo sistema de control de versiones es registrar los cambios hechos en el código. Esto permite volver al historial del proyecto para ver quiénes son los autores, averiguar dónde se introdujeron los errores y revertir los cambios problemáticos. Pero tener todo este historial no sirve de nada si no sabes cómo navegar por él. Ahí es donde entra en juego el comando git log.

Teclear el siguiente comando **git log**

```
C:\materias\Automatizacion\Unidad 1\minuevoproyecto>git log
commit 44e716c40ff5df4976a992320efac5b9b68d2d0b (HEAD -> master)
Author: gbarron2014 <gbarron@utng.edu.mx>
Date:   Mon May 22 13:43:15 2023 -0600

    Version Inicial

commit 0fc54f177a6d960f0c6f72900bd30c827adad71e
Author: gbarron2014 <gbarron@utng.edu.mx>
Date:   Mon May 22 13:16:51 2023 -0600

    Versión Inicial
```

Versión reducida **git log –oneline**

```
C:\materias\Automatizacion\Unidad 1\minuevoproyecto>git log --oneline
44e716c (HEAD -> master) Version Inicial
0fc54f1 Versión Inicial
```

Mostrar un commit especifico **git show 44e716**

```
C:\materias\Automatizacion\Unidad 1\minuevoproyecto>git show 44e716
commit 44e716c40ff5df4976a992320efac5b9b68d2d0b (HEAD -> master)
Author: gbarron2014 <gbarron@utng.edu.mx>
Date:   Mon May 22 13:43:15 2023 -0600

    Version Inicial

diff --git a/.gitignore b/.gitignore
new file mode 100644
index 0000000..b43bf86
--- /dev/null
+++ b/.gitignore
@@ -0,0 +1 @@
+README.md
```

Revisando los cambios actuales

git diff es un comando de Git de usos múltiples que, cuando se ejecuta, ejecuta una función diff en las fuentes de datos de Git. Estas fuentes de datos pueden ser compromisos, ramas, archivos y más

Modificar el archivo **README.md** agregando una o más líneas

Ejecutar el comando **git diff**

```
C:\materias\Automatizacion\Unidad 1\minuevoproyecto>git diff
diff --git a/README.md b/README.md
index 845358e..67ff798 100644
--- a/README.md
+++ b/README.md
@@ -1 +1,2 @@
-Repositorio de trabajo
\ No newline at end of file
+Repositorio de trabajo
+Otro repositorio
\ No newline at end of file
```

Verificar Instantanea del proyecto

git checkout

```
C:\materias\Automatizacion\Unidad 1\minuevoproyecto>git checkout
M       README.md
```

Cambie de rama o restaure los archivos del árbol de trabajo

git checkout master

```
C:\materias\Automatizacion\Unidad 1\minuevoproyecto>git checkout master
Already on 'master'
A       Nueva carpeta/Nuevo documento de texto.txt
```

Revertir cambios

git revert <Nombre de commit>

Práctica 01 Comandos Git a nivel local

Realizar el ejercicio mediante línea de comandos en Windows, Linux o Mac

- Configurar los atributos globales de tú usuario y email de Git.

```
C:\Windows\System32>git config --global user.name 1220100075

C:\Windows\System32>git config --global user.email 1220100075@alumnos.utng.edu.mx
```

- Crear una carpeta llamada **gitexample** e inicializar el repositorio con el comando correspondiente.

```
C:\Windows\System32>mkdir gitexample

C:\Windows\System32>cd gitexample
```

- Comprueba el estado del **repositorio** a través del comando correspondiente.

```
C:\Windows\System32\gitexample>git init
Initialized empty Git repository in C:/Windows/System32/gitexample/.git/
```

- Crea los siguientes archivos:
  - **indice.txt** con el siguiente contenido:
    Este es un ejercicio práctico de Git.
    Archivo de contenido
  - **archivo1.txt** con el siguiente contenido:
    Este es un ejercicio práctico de Git.
    Archivo 1
  - **configuracion.conf** con el siguiente contenido:
    Este es un ejercicio práctico de Git.
    Archivo de configuración

- Añadir los archivos al Stanging Area mediante el comando correspondiente.

```
C:\Windows\System32\gitexample>git add archivo1.txt

C:\Windows\System32\gitexample>git add configuration.conf
fatal: pathspec 'configuration.conf' did not match any files

C:\Windows\System32\gitexample>git add configuration.conf.txt

C:\Windows\System32\gitexample>git add indice.txt

C:\Windows\System32\gitexample>
```

- Comprueba el estado del repositorio.

```
C:\Windows\System32\gitexample>git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   archivo1.txt
        new file:   configuration.conf.txt
        new file:   indice.txt
```

- Hacer un **commit** de los cambios con el mensaje "Introduccion a git".

```
C:\Windows\System32\gitexample>git commit -m "Introduccion a git"
[master (root-commit) 2628f92] Introduccion a git
 3 files changed, 6 insertions(+)
 create mode 100644 archivo1.txt
 create mode 100644 configuration.conf.txt
 create mode 100644 indice.txt
```

- Mostrar los cambios de la última versión del repositorio con el comando correspondiente.

```
C:\Windows\System32\gitexample>git log
commit 2628f929e134805bd4801d15c5be77092723b273 (HEAD -> master)
Author: 1220100075 <1220100075@alumnos.utng.edu.mx>
Date:   Mon May 29 17:34:20 2023 -0600

    Introduccion a git
```

- Cambiar el mensaje del último **commit** por "*Añadido Introduccion a Git*"

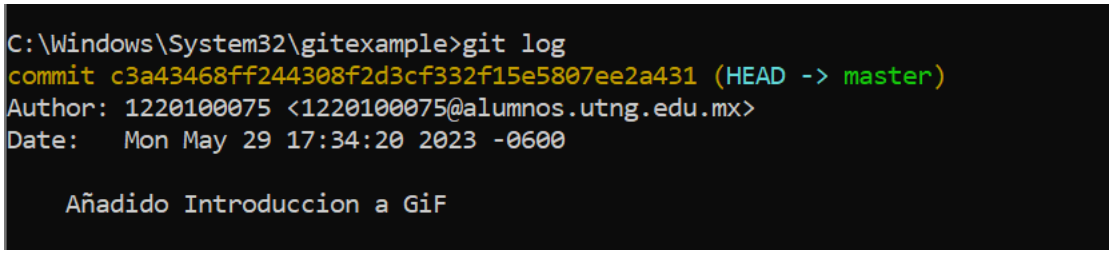
```
C:\Windows\System32\gitexample>git commit --amend -m "Añadido Introduccion a GiF"
[master c3a4346] Añadido Introduccion a GiF
 Date: Mon May 29 17:34:20 2023 -0600
 3 files changed, 6 insertions(+)
 create mode 100644 archivo1.txt
 create mode 100644 configuration.conf.txt
 create mode 100644 indice.txt
```

- Volver a mostrar los últimos cambios del repositorio.

```
C:\Windows\System32\gitexample>git log
commit c3a43468ff244308f2d3cf332f15e5807ee2a431 (HEAD -> master)
Author: 1220100075 <1220100075@alumnos.utng.edu.mx>
Date:   Mon May 29 17:34:20 2023 -0600

    Añadido Introduccion a GiF
```

- Mostrar la historia del último commit.

- 

Notas:

Una vez terminado el ejercicio crea un archivo en formato PDF donde ejecutas los comandos y las salidas correspondientes y enviarlo a la repositorio Git

Nombrar el archivo unidad2-[apellidos]-[nombre].pdf

Flujo de trabajo GitFlow

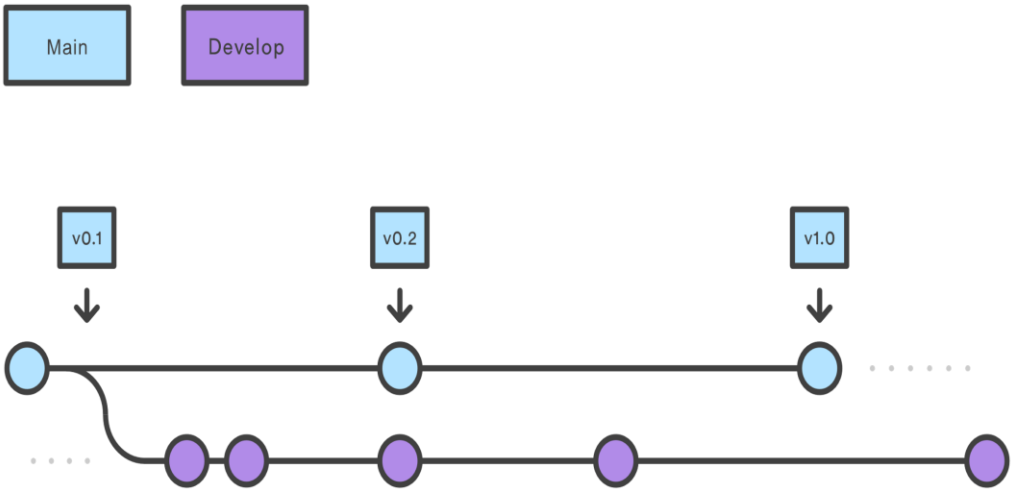
Gitflow es un flujo de trabajo de Git heredado que originalmente era una estrategia disruptiva y novedosa para administrar ramas de Git.

Gitflow es un modelo alternativo de bifurcación de Git que implica el uso de bifurcaciones de características y múltiples bifurcaciones primarias. Fue publicado por primera vez y popularizado por Vincent Driessen en nvie .

Instalar GitFlow de acuerdo al sistema operativo que tengas

¿Cómo funciona?

En lugar de una sola rama **main**, este flujo de trabajo utiliza dos ramas para registrar el historial del proyecto. La rama **main** almacena el historial oficial de versiones y la rama **develop** que sirve para integración de funcionalidades. También es conveniente etiquetar todas las confirmaciones en la rama **main** con un número de versión.



Instrucciones

Paso 1. Abrir la consola del sistema operativo, y crear un directorio de trabajo mediante el siguiente comando:

mkdir testflow

Paso 2. Inicializar el directorio de trabajo mediante el siguiente comando

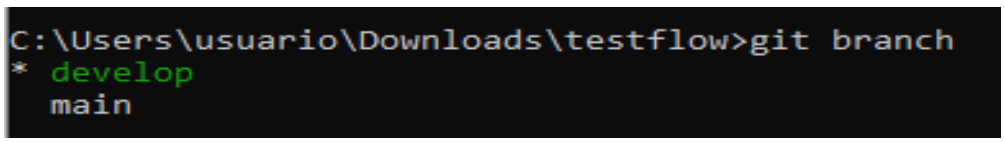
git flow init

Se te pregunta por las dos ramas default:

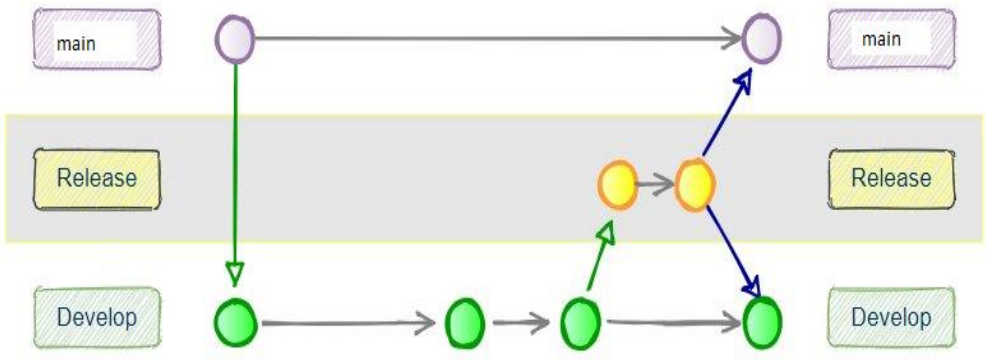
Branch name for production releases: [master] **main**
Branch name for "next release" development: [develop]
Feature branches? [feature/]
Bugfix branches? [bugfix/]
Release branches? [release/]
Hotfix branches? [hotfix/]
Support branches? [support/]
Version tag prefix? [] 1.0
Hooks and filters directory? [C:/Users/usuario/Downloads/testflow/.git/hooks]

Paso 3. Teclear el comando para conocer los branchs de los directorios de trabajo.

git branch



Observa en la siguiente imagen qué tenemos dos ramas por omisión las cuales son main y develop, el cual todo se trabajará sobre la rama develop.



Paso 4. Crear una rama para la nueva funcionalidad llamada **create-form-login**, por tanto teclear el siguiente comando.

git flow feature start fun-login create-form-login

Paso 5. Checar las ramas qué están sobre el proyecto mediante el siguiente comando

git branch -a

```
C:\Users\usuario\Downloads\testflow>git branch -a
  develop
* feature/create-form-login
  main
```

Paso 6. Crear dos archivos llamados **index.html** y l**ogin.html** y agregar contenido

Paso 7. Agregar los dos archivos al Staging Area mediante el comando: **git add .**

Paso 8. Dar commit a través del comando: **git commit -m "Funcionalidad de login"**

Paso 10. Finalizar la funcionalidad los cambios

git flow feature finish create-form-login

```
C:\Users\usuario\Downloads\testflow>git flow feature finish create-form-login
Switched to branch 'develop'
Updating 743601c..e515fa4
Fast-forward
 index.html | 2 ++
 1 file changed, 2 insertions(+)
 create mode 100644 index.html
Deleted branch feature/create-form-login (was e515fa4).

Summary of actions:
- The feature branch 'feature/create-form-login' was merged into 'develop'
- Feature branch 'feature/create-form-login' has been locally deleted
- You are now on branch 'develop'
```

Paso 11. Checar las ramas, historia de los commits y el estado del directorio de trabajo

Crear una liberación

Paso 12. Cambiar al branch main: git checkout main

Paso 13. Listar el contenido de la rama: dir

Paso 14. Cambiar al branch develop: git branch develop

Paso 15. Listar el contenido de la rama: dir

Paso 16. Verificar liberación con el siguiente comando: **git tag -l**

Paso 17. Liberación de la rama develop a la rama main: **git flow release start '1.0'**

Paso 18. Modificar el archivo **index.html** que represente una modificación o arreglo a la funcionalidad

Hello World!!!

Paso 19. Agregar el archivo index.html con el comando

git add .

git commit -m "Se ha arregla index"

Paso 20. Terminar la liberación

git flow release finish '1.0'

Práctica 02 Comandos Git Flow a nivel local

Realizar el ejercicio mediante línea de comandos en Windows, Linux o Mac

- Crear una carpeta llamada **gitpracticaflow** e inicializar el repositorio con el comando correspondiente tomando en cuenta qué existen dos ramas: **main y develop**
- Comprueba el estado del repositorio a través del comando correspondiente.
- Crear una nueva funcionalidad llamada **fun-login** utilizando el patrón MVC y los comandos correspondientes de **gitflow**
    - Crea tres carpetas llamadas: **model, views y controlle**r.
    - Agregar un archivo para cada carpeta
        - Model: **ModelLoginDTO.java**
        - Views: **viewLogin.html**
        - Controller: **ControllerLogin.java**
    - Finaliza la funcionalidad
- Verificar las diferencias entre las ramas **main** y **develop**
- Crear una modificación a la funcionalidad fun-login
    - **ModelLoginDTO.java** agregar la instrucción public class modelLoginDTO {}
    - **viewsLogin.html** agregar la instrucción <h1> Estoy entre ramas </h1>
    - **ControlleLogin.java** agregar la instrucción public class controlleLogin.java{}
    - Guardar los cambios al Stanging Area
    - Verificar la historia de los commits
    - Verificar las diferencias entre las ramas main y develop a través del comando
- Iniciar una liberación con la versión 1.0 como prefijo
- Iniciar la liberación con el sufijo "**.1**".
- Finalizar la liberación
- Verificar las diferencias entre las ramas main y develop a través del comando

Notas:

Cada comando tecleado en la consola y su salida deberá ser capturado

Una vez terminado el ejercicio crea un archivo en formato **PDF** y enviarlo a la repositorio Git

Nombrar el archivo: **unidad2practica2-[apellidos]-[nombre].pdf**

Repositorios

Un repositorio contiene todos los archivos de tu proyecto y el historial de revisiones de cada uno de ellos. Puedes debatir y administrar el trabajo de tu proyecto dentro del repositorio.

Comando Git remote

El comando git remote es una pieza del sistema más amplio que es responsable de sincronizar los cambios. Los registros registrados mediante el comando git remote se utilizan junto con los comandos git fetch, git pushy git pull . Todos estos comandos tienen sus propias responsabilidades de sincronización.

Git remote

El comando git remote permite crear, ver y eliminar conexiones a otros repositorios. Las conexiones remotas son más como marcadores que como enlaces directos a otros repositorios. En lugar de proporcionar acceso en tiempo real a otro repositorio, sirven como nombres convenientes que se pueden usar para hacer referencia a una URL no tan conveniente.

Crear un repositorio en la tu cuenta personal de GitHub

Teclear el comando para conocer los enlaces a repositorios remotos

```
git remote
```

Comando para agregar enlace a un repositorio remoto

```
git remote add <name> <url>
```

```
git remote add origin URL
```

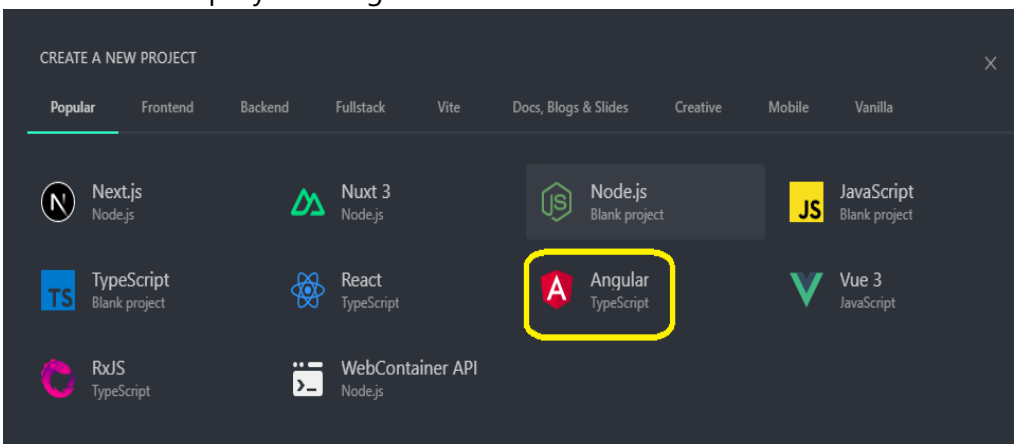
Eliminar la conexión con el repositorio remoto llamado origin

```
git remote rm <name>
```

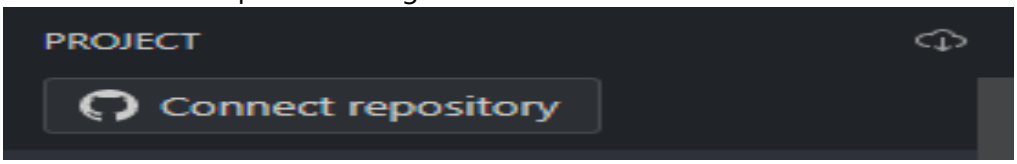
```
git remote rm origin
```

Práctica 03 Manejo de repositorios remotos

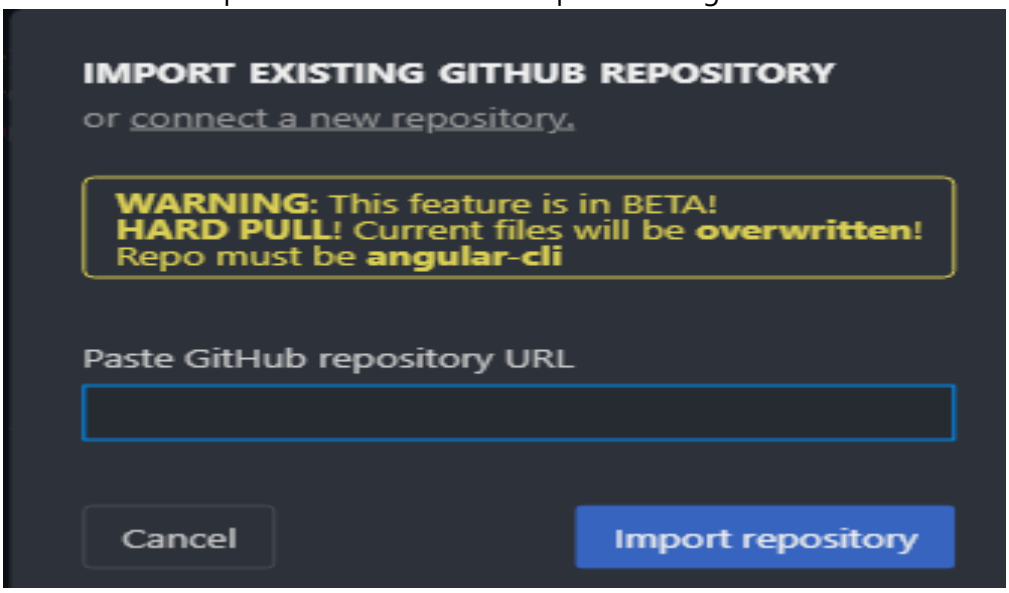
- Crear un repositorio en tu cuenta personal de GitHub
  - Nombrarlo como **prac03-web integral**
  - Agregar una breve descripción
  - Definir accesibilidad pública
- Ubicar la carpeta de la aplicación login creada en la unidad 1
  - Inicializa la carpeta en dado caso qué no lo esté con **git** o **git flow**
  - Crear o modificar el archivo .gitignore de tal manera qué no se suba la carpeta node-modules y todos los archivos *.exe
  - Agregar todos los archivos de la aplicación al Stanging Area
  - Comprometer los cambios.
  - Realiza el merge o fusión de la rama develop a la rama main
  - Crear un enlace o URL hacia el repositorio remoto GitHub recién creado.
  - Enviar los cambios a rama main del repositorio remoto
  - Verificar qué se hayan dado
- Ingresar a la página StackBlitz
  - Registrar con cuenta personal de GitHub
  - Crear un nuevo proyecto Angular



  - Conectar con el repositorio de github



- Conectarlo con repositorio existente de la aplicación login



- Modificar el archivo main.ts

```
import 'zone.js'
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';

import { AppModule } from './app/app.module';

platformBrowserDynamic().bootstrapModule(AppModule)
  .catch(err => console.error(err));
```

(alias) class AppModule
import AppModule

Notas:

Cada comando tecleado en la consola y su salida deberá ser capturado

Una vez terminado el ejercicio crea un archivo en formato **PDF** y enviarlo a la repositorio Git

Nombrar el archivo: **unidad2practica3-[apellidos]-[nombre].pdf**

Anexo I Comandos Básicos

Comando	Descripción
git init	**Inicializa una nueva base de datos Git**
git clone	**Copia una base de datos existente**
git status	**Checa el estatus del proyecto local**
git diff	**Revisa los cambios realizados al proyecto**
git add	**Le indica a Git hacer seguimiento de archivo modificado**
git commit	**Guarda el estado actual del proyecto a la base de datos**
git push	**Copia la base de datos local al sevidor remoto**
git pull	**Copia una base de datos remota a la máquina local**
git log	**Checa la historia del proyecto**
git branch	**Lista, crea o elimina ramas**
git merge	**Unir la historia de dos ramas a una sola**
git stash	**Conserva los cambios actuales a ser utilizados más tarde**