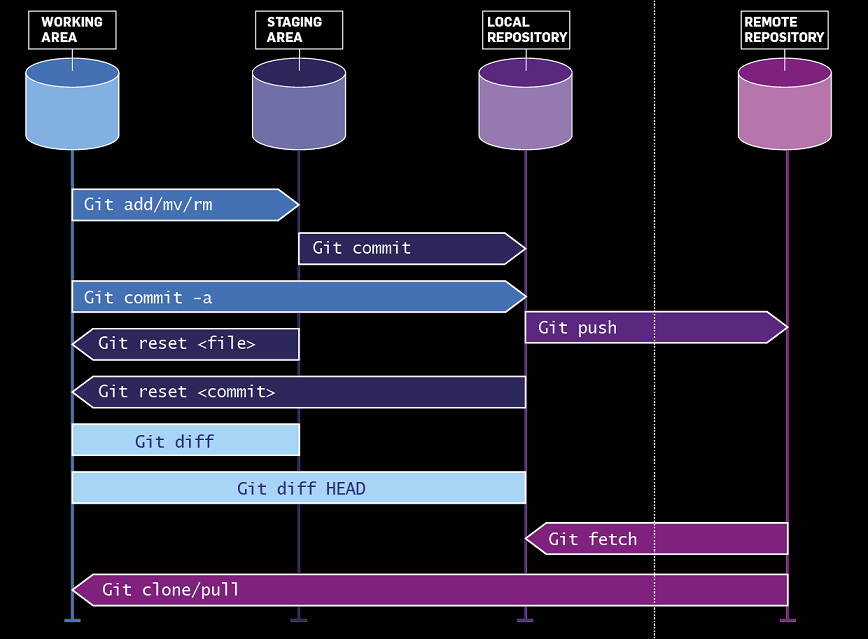
* pwd nos indica la dirección
* git init Creará un nuevo repositorio local GIT. Usando git init [nombre del proyecto] También puedes crear un repositorio dentro de un directorio especificando el nombre del proyecto.
* git clone Lo usas para clonar un repositorio.
* git add Lo usas para agregar archivos al área de de preparación  
   Puede ser git add (archivo) o poner . para abarcar a todos los archivo
* git commit Lo usas para crear un cambio que se guardará en el directorio git.  
   git commit -m “Descripcion”
* git config Lo usas para establecer una configuración específica de usuario, podría ser el email, usuario o tipo de formato.

git config user.name nombre  
git config user.email adsad@asdas  
git config user.password token

* git log nos da el historial de las commit
* git status Lo usas para que muestre la lista de archivos que has cambiado, junto con archivos que serán preparados y confirmados.
* git push Lo usas para enviar confirmaciones a la rama maestra/principal del repositorio remoto.
* git remote Nos permite ver todos los repositorios remotos.
* git checkout Nos permite crear ramas y navegar entre ellas.
* git checkout -- (archivo) Nos permite restaurar el archivo del ultimo commit.
* git diff (archivo) nos permite ver las diferencias entre los archivos
* git pull Lo usas para fusionar todos los cambios que hiciste en el repositorio local con el directorio de trabajo local.
* Archivo .gitignore: escribir nombre de archivos que no queremos hacer commit
* git branch: Nos permite ver las ramas disponibles
* git branch nombredelarama nos permite crear una rama
* git remote add origin URL Para enlazarlo con un repositorio
* git push -u origin rama para subir la información al repositorio de la rama en cuestion

Comando git



* **git init** es el comando para inicializar un directorio como repositorio Git, se ejecuta dentro del directorio del proyecto, y como resultado crea un subdirectorio **.git** que contiene todos los archivos para poder realizar el seguimiento de los cambios, etiquetas, etc.
* **git add <archivo>** luego de la creación, modificación o eliminación de un archivo, los cambios quedan únicamente en el área de trabajo, por lo tanto es necesario pasarlos al área de preparación mediante el uso del comando **git add** , para que sea incluido dentro de la siguiente Confirmación ( *cometer* ).
* **git status** es un comando que permite conocer en qué estado se encuentran los archivos
* **git commit**, con este comando se confirman todos los cambios registrados en el área de preparación, o lo que es lo mismo, se pasan los cambios al repositorio local.
* **git push** es el comando que se utiliza para enviar todas las confirmaciones registradas en el repositorio local a un repositorio remoto.
* **git pull** funciona al inverso de **git push**, trayendo todos los cambios al repositorio local, pero también dejándolos disponibles directamente para su modificación o revisión en el área de trabajo. Es importante mencionar que se utiliza cuando ya se tiene un repositorio local vinculado a uno remoto, al igual que con el comando **git push**.
* **git clone** , en el caso de necesitar "bajar" un repositorio remoto de algún proyecto ya existente se puede ejecutar este comando. Genera un directorio (con el nombre del repositorio o uno especificado explícitamente) que contiene todo lo propio al proyecto, además del subdirectorio **.git** necesario para poder gestionar los cambios y todo lo pertinente al repositorio Git.

**Opciones comunes**

El comando principal, sin parámetros, permite listar todas las ramas del repositorio

**git *branch***

Para crear una rama nueva se usa el comando como se muestra a continuación:

**git *branch <branch>***

Para eliminar una rama, es necesario agregar la opcion **-d**

**git *branch -d <branch>***

Si se desea cambiar el nombre de una rama, se puede utilizar la opción **-m**, es importante mencionar que cambia el nombre de la rama sobre la cual se está trabajando.

**git *branch -m <branch>***

Finalmente, para listar todas las ramas en el repositorio remoto se puede utilizar el comando:

**git *branch -a***

**Ramas en desarrollo y maestras**

En vez de una única rama master, este flujo de trabajo utiliza dos ramas para registrar el historial del proyecto. La rama master almacena el historial de publicación oficial y la rama desarrollar sirve como rama de integración para las funciones. Asimismo, conviene etiquetar todas las confirmaciones de la rama master con un número de versión.

El primer paso es complementar la master predeterminada con una rama develop. Una forma sencilla de hacerlo es que un desarrollador cree una rama develop vacía localmente y la envíe al servidor:

***git branch develop***

**git *push -u origin develop***

Esta rama contendrá el historial completo del proyecto, mientras que la **master**(rama maestra)contendrá una versión abreviada. Otros desarrolladores deben clonar el repositorio central y crear una rama de seguimiento para **develop**.

A la hora de utilizar la biblioteca de extensiones de git-flow, ejecutar git flow init en un repositorio existente creará la rama develop:

**git *flow init***

Initialized empty Git repository in ~/project/.git/

No branches exist yet. Base branches must be created now.

Branch name for production releases: [main]

Branch name for "next release" development: [develop]

How to name your supporting branch prefixes?

Feature branches? [feature/]

Release branches? [release/]

Hotfix branches? [hotfix/]

Support branches? [support/]

Version tag prefix? []

**git branch**

\*develop

main

**Ramas de función**

Todas las funciones nuevas deben residir en su propia rama, que se pueden enviar al repositorio central para copia de seguridad / colaboración. Sin embargo, en vez de ramificarse de la **master** , las ramas **feature** utilizan La **develop** Como rama primaria. Cuando una función está terminada, se vuelve a fusionar en el de desarrollo. Las funciones no deben interactuar nunca directamente con la master.

Ten en cuenta que las ramas **feature** combinadas con la rama **develop** conforman, a todos efectos, el flujo de trabajo de ramas de función. Sin embargo, el flujo de trabajo Gitflow no termina aquí.

Las ramas feature suelen crearse a partir de la última rama **develop** .

Para crear una rama de función sin las extensiones git-flow se pueden ejecutar los comandos:

**git checkout develop**

**git checkout -b feature\_branch**

O bien utilizando la extensión, con el comando:

**git flow feature start feature\_branch**

 Luego de esto se continúa trabajando normalmente con Git.

 Para finalizar una rama de función, cuando el desarrollo de esto haya culminado, hay que fusionar la rama con la rama **develop,** y para esto tenemos dos opciones, sin la extensión git-flow:

**git checkout develop**

**git merge feature\_branch**

O bien con la extensión:

**git flow feature finish feature\_branch**

**Ramas de publicación**

Cuando develop haya adquirido suficientes funciones para una publicación (o se acerque una fecha de publicación predeterminada), debes bifurcar una rama release a partir de una develop. Al crear esta rama, se inicia el siguiente ciclo de publicación, por lo que no pueden añadirse nuevas funciones una vez pasado este punto (en esta rama solo deben producirse las soluciones de errores, la generación de documentación y otras tareas orientadas a la publicación). Cuando está lista para el lanzamiento, la rama release se fusiona en la master y se etiqueta con un número de versión. Además, debería volver a fusionarse en develop, que podría haber progresado desde que se iniciara la publicación.

Utilizar una rama específica para preparar publicaciones hace posible que un equipo perfeccione la publicación actual mientras otro equipo sigue trabajando en las funciones para la siguiente publicación. Asimismo, crea fases de desarrollo bien definidas (por ejemplo, es  fácil decir: "Esta semana nos estamos preparando para la versión 4.0" y verlo escrito en la estructura del repositorio).

Crear ramas **release** es otra operación de ramificación sencilla. Al igual que las ramas **feature** , las ramas **release** en sí se basan en la rama **develop**. Se puede crear una nueva rama **release** utilizando los siguientes métodos. Sin las extensiones de git-flow:

**git checkout develop**

**git checkout -b release/0.1.0**

O bien con las extensiones:

**git flow release start 0.1.0**

**Switched to a new branch 'release/0.1.0'**

En cuanto la publicación esté lista para su lanzamiento, publicación se fusionará en la **master**y la **develop**; y luego se eliminará la rama **release**. Es importante volver a fusionarla en **develop** porque podrían incluir actualizaciones críticas a la rama **release**, y las funciones nuevas tienen que poder acceder a ellas. Si tu organización enfatiza la revisión de código, este sería el lugar ideal para una solicitud de incorporación de cambios.

Para finalizar una rama release, utiliza los métodos siguientes. Sin las extensiones de git-flow:

**git *checkout main***

**git *merge release / 0.1.0***

O con la extensión:

**git *flow release finish '0.1.0'***

**Ramas de corrección**

Las ramas de mantenimiento o de "corrección" (hotfix) sirven para reparar rápidamente las publicaciones de producción. Las ramas hotfix son muy similares a las ramas release y feature, salvo por el hecho de que se basan en la maestra, no en la desarrollar. Esta es la única rama que debería bifurcarse directamente a partir de la master. Cuando se haya terminado de aplicar la corrección, debería fusionarse en la master y la Develop (o la rama release actual), y la master debería etiquetarse con un número de versión actualizada.

Tener una línea de desarrollo específico para la corrección de errores permite que tu equipo aborde las incidencias sin interrumpir el resto del flujo de trabajo ni esperar al siguiente ciclo de publicación. Puedes concebir las ramas de mantenimiento como ramas release ad hoc que trabajan directamente con la master. Se puede crear una nueva rama hotfix utilizando los siguientes métodos. Sin las extensiones de git-flow:

**git checkout main**

**git checkout -b hotfix\_branch**

O bien utilizando las extensiones:

**git flow hotfix start hotfix\_branch**

Al igual que al finalizar una rama **de liberación**, una rama **revisión**se fusiona tanto en la **master c**omo en la **desarrollar**.

Sin las extensiones:

**git checkout main**

**git merge hotfix\_branch**

**git checkout develop**

**git merge hotfix\_branch**

**git branch -D hotfix\_branch**

O con las extensiones:

**git flow hotfix finish hotfix\_branch**