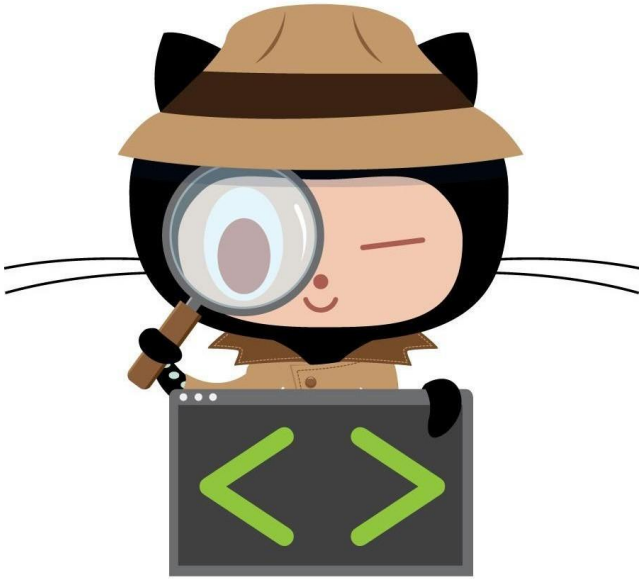
Antes de empezar...

- Git Installed
- GitHub account
- Your favorite editor



¿Que es Git?

Un sistema de **control de versiones** distribuido.

Cualquier proyecto que use Git tendrá una carpeta **.git** que almacena todo el historial del proyecto.

¿Por qué Git?

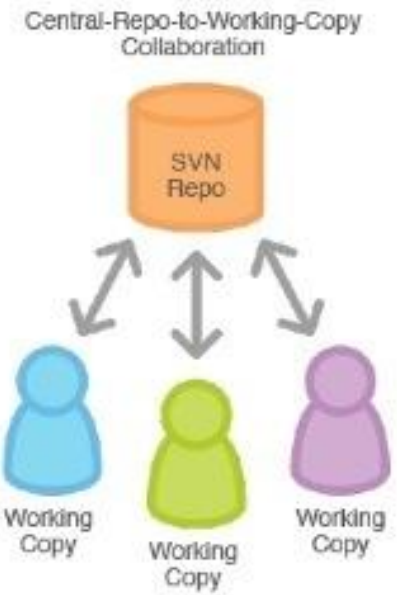
Saber manejar un sistema de control de versiones dejó de ser una opción en el mundo laboral. La demanda de proyectos de software de gran importancia ha crecido exponencialmente, por esta razón el uso de Git es **indispensable** para cualquier programador en el 2021.

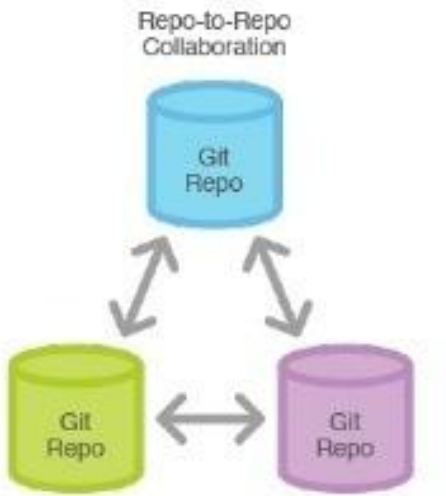
Historia:
Sepa exactamente qué archivos cambiaron, quién hizo esos cambios y cuándo ocurrieron esos cambios.

Respaldo:
Posibilidad de tener diferentes versiones del código en diferentes lugares.

Colaboración:
Colabore fácilmente con otras personas en el mismo proyecto cargando y recibiendo cambios .

Git VS SVN





SVN - Centralizado

Git - Distribuído

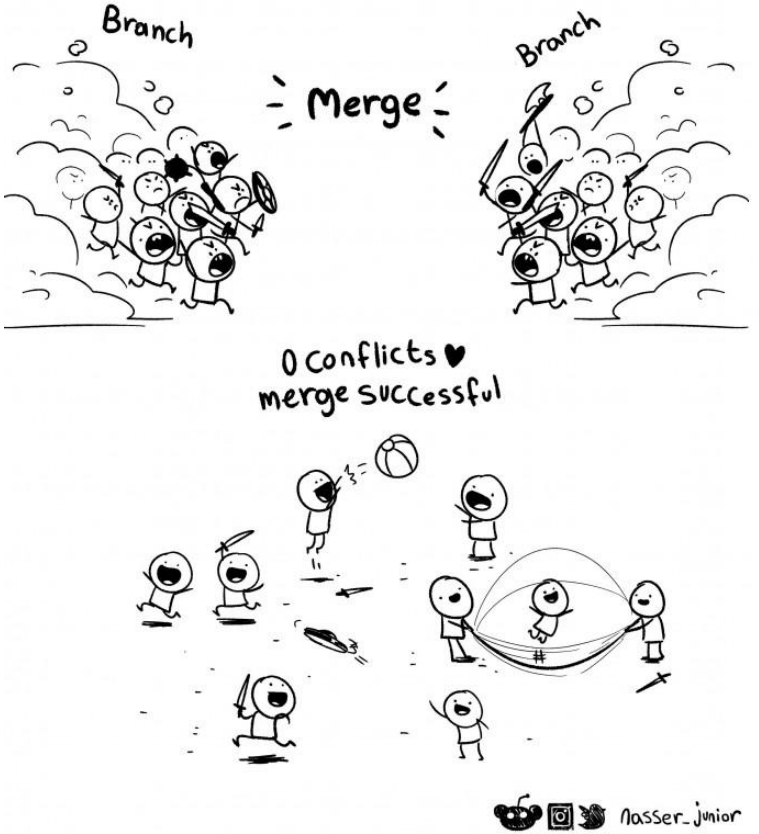
¿Que es GitHub?

GitHub es un sitio web que nos permite usar **git** y crear repositorios **en línea**. También puede almacenar todos sus proyectos en línea de forma gratuita.

¿Que es un repositorio?

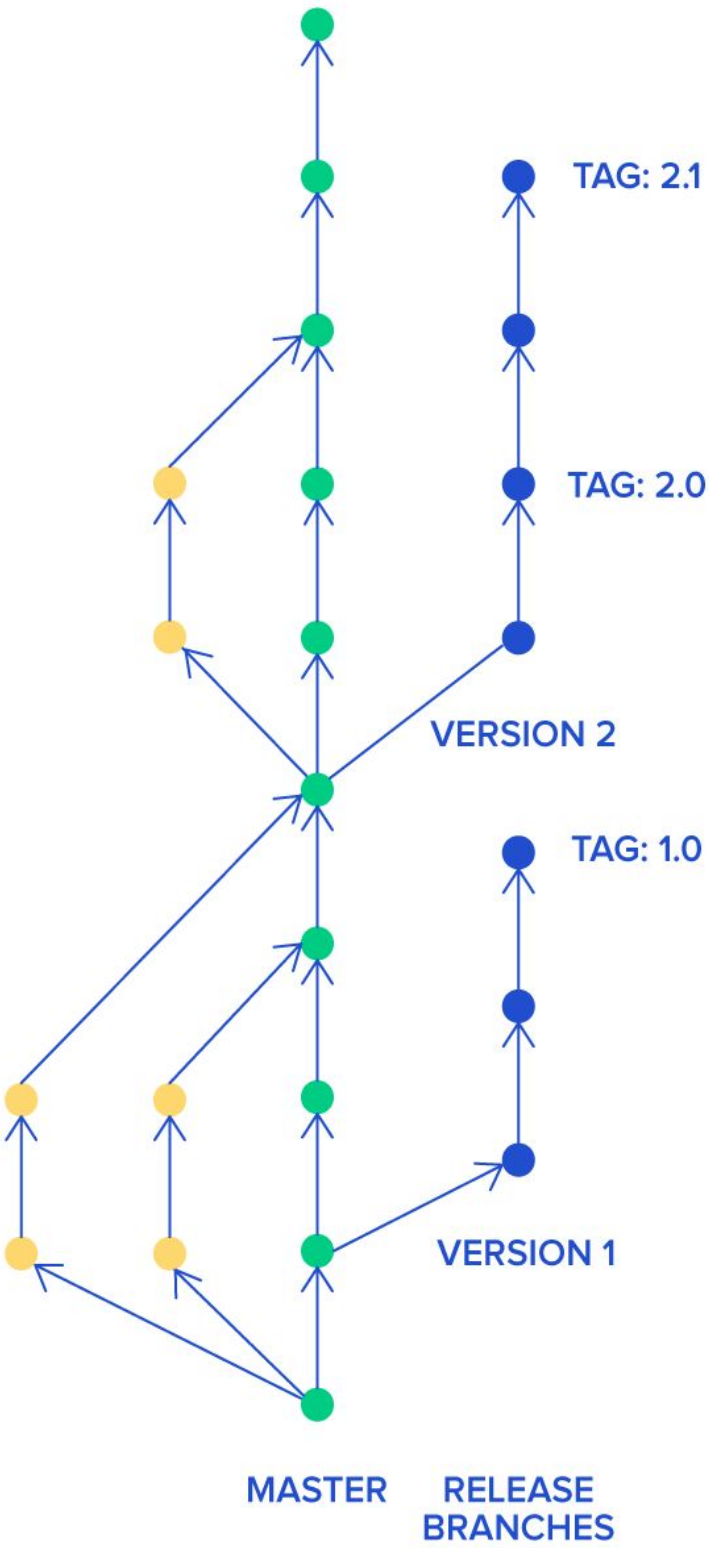
Un repositorio es un contenedor que alberga su proyecto y su historial. Si la carpeta de su proyecto contiene la carpeta **“.git”,** ¡entonces está trabajando con un repositorio!

Un poco de humor



Git Trunk Based

- Una unica rama master de la cual parten los desarrolladores y crean ramas para sus caracteristicas, cuando su rama esta completa y ha sido probada se envia un Merge a la rama principal.
- Crea una rama de ciclo de vida corto partiendo de la master
- Realizo cambios y hago commit en esta rama
- Se debe probar y cuando este listo hago merge
- Actualizo la rama para evitar conflictos
- Hago MR en la rama principal


TAG: 2.1
TAG: 2.0
VERSION 2
TAG: 1.0
VERSION 1
MASTER
RELEASE
BRANCHES

Git Flow

Es una metodologia de flujo de trabajo aplicado a un repositorio git

- Se crea una rama de desarrollo a partir de la maestra.
- Una rama de publicación se crea a partir de la de desarrollo.
- Las ramas de función se crean a partir de la de desarrollo.
- Cuando una función está completa, se fusiona en la rama de desarrollo.
- Cuando la rama de publicación está lista, se fusiona en la de desarrollo y la maestra.
- Si se detecta una incidencia en la maestra, se crea una rama de corrección a partir de la maestra.
- Una vez que la corrección está completa, se fusiona tanto con la de desarrollo como con la maestra.

Git Flow

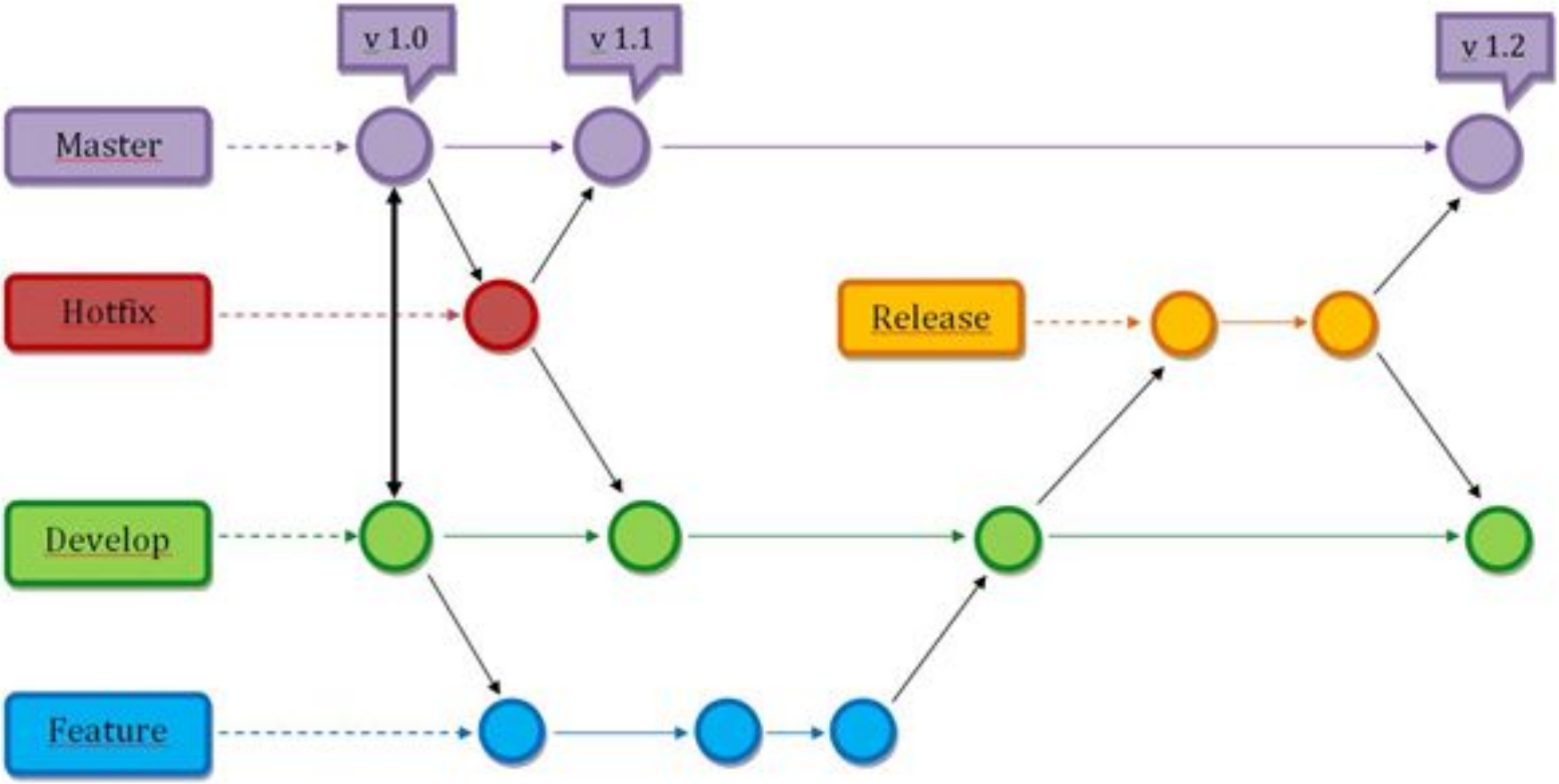
Master: Almancena el historial de versiones, nunca enlaza el código directo

Develop: recibe las integraciones de features, nunca recibe el código directo

Features: parten de develop son temporales, aquí es donde el desarrollador agrega el código y las nuevas características

Release: parten de develop, es una preparación, se despliega el entorno de pruebas, se hacen ajustes y se integran en master y develop

Hotfix: parte de master para arreglar errores urgentes, se itegran en master y dev y se marca la versión del tag


v 1.0
v 1.1
v 1.2
Master
Hotfix
Release
Develop
Feature

Git flow tipo_rama_de_soporte accion nombre_de_rama

- Tipo de rama de soporte: Son las ya mencionadas **feature, release y hotfix**.
- Acción:
- **start** para crear una nueva rama en local. Al ejecutarlo directamente te cambia a ella, por lo que no será necesario hacer un checkout a esta nueva rama.
- **finish** para finalizarla. Recordar en este punto que, dependiendo del tipo de rama de soporte con la que estemos trabajando, esto tendrá diferentes consecuencias, incorporando los cambios a develop y/o master, cambiando a estas y eliminando la rama de soporte (tanto en local como en remoto).
- **publish** para subirla a un repositorio remoto. Esta acción en principio será la menos utilizada, o por lo menos yo es la que menos gasto, ya que las ramas "online" son o deben ser las principales, pero puede suceder que necesitemos que alguna de las de soporte estén disponibles en remoto.

Unos apuntes especiales sobre las ramas feature que está bien saber son: por un lado, que podemos listar las ramas features actuales con git flow y por otro que, para navegar entre ramas feature o volver a ellas después de una pausa, podemos usar

git flow feature checkout nombre_rama,

de modo que concretamos que la rama a la que queremos volver es del tipo indicado.

Trunk-Based

- Es una parctica requerida para la integracion continua
- Cuando se esta iniciando un Proyecto
- Caundo se require iterar rapido
- Cuando el equipo de Desarrollo esta integrado por senior
- Caundo con un enfoque de TDD (Desarrollo basado en pruebas)

Gitflow

- Ideal para proyectos Open Source (codigo abierto)
- Cuando el nuemro d desarrolladores junior es alto
- Si el indice de totacion del equipo es alto
- Cuando ya se cuenta con un product establecido (en produccion)
- Calendario de releases fijo

¡Empecemos!

Si estás en windows, abre una Git-Bash.

Si estás en linux o Mac, abre una terminal.

GitHub usa la dirección de correo electrónico que se establece en su configuración de Git local para asociar las confirmaciones enviadas desde la línea de comandos con su cuenta de GitHub.

Inicializando un nuevo repositorio

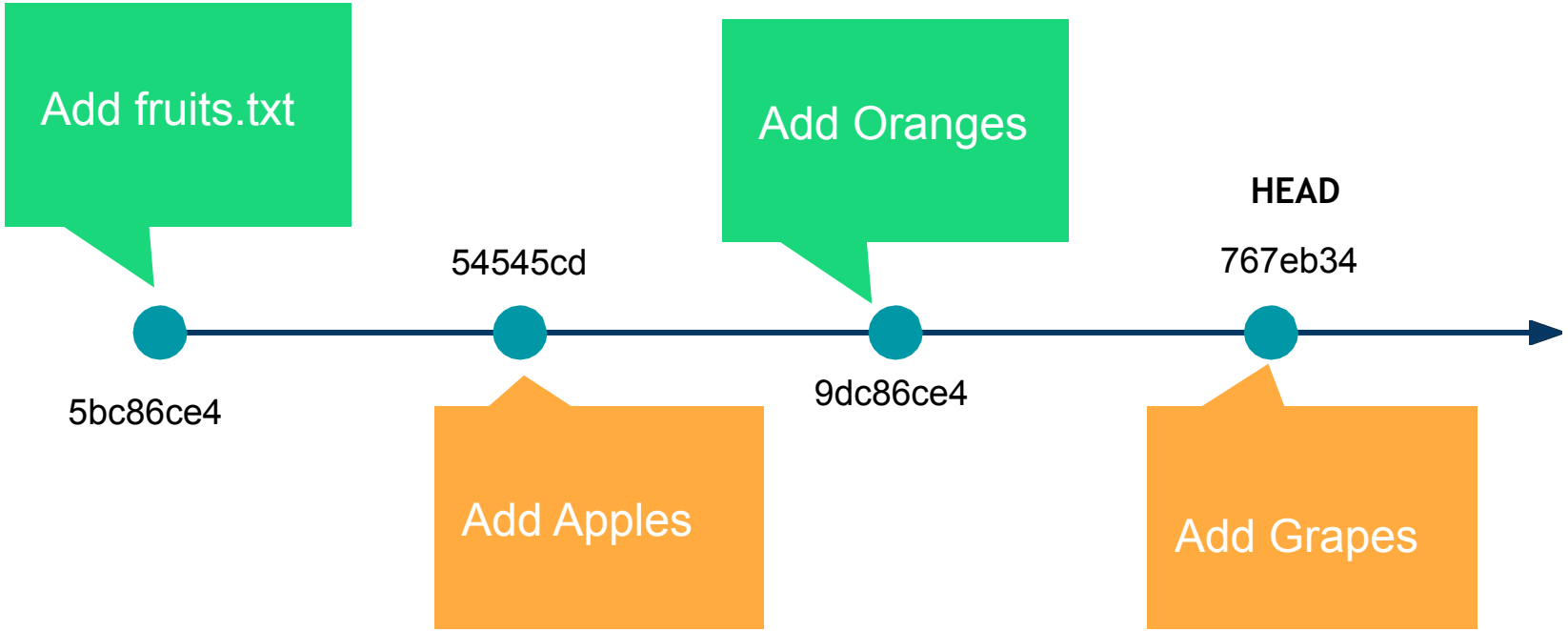
```
# creating a new folder for our project
$ mkdir MyProject
# changing directory to our project folder
$ cd MyProject
# initializing the current folder as a repository
$ git init

Initialized empty Git repository in /home/user/MyProject/.git/
```

Commits

Checkpoints / Snapshot del estado de su repositorio (proyecto) en un momento determinado.


# shows the state of the working directory and the staging area.
$ git status
# Add the files to staging area
$ git add fruits.txt
# Commit the changes into the repository
$ git commit -m "Add fruits.txt"

Git Log

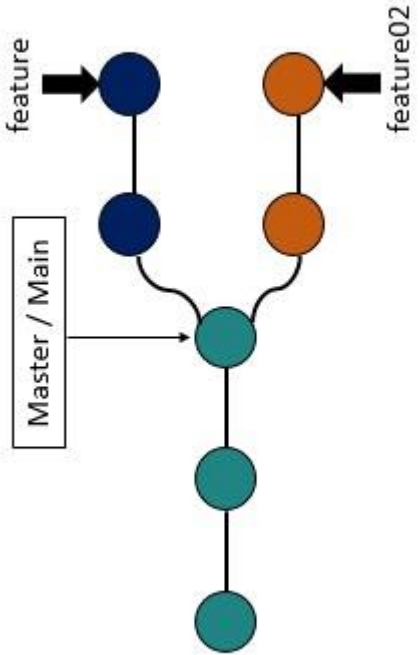

Add fruits.txt
Add Oranges
HEAD
54545cd
767eb34
5bc86ce4
9dc86ce4
Add Apples
Add Grapes

Reset a tus Commits

```
# To reset files from the staging area after git add
$ git reset [options]
# To reset particular commit or file
$ git reset HEAD/file_name [options]
# resets the code and commits
--hard
# resets the code and commits but keeps it in the staging area
--soft
# rests commits and keeps it in the workingdirectory (default)
--mixed
```

Git
Branches

```
# To list all branches
$ git branch
# To create a new branch
$ git branch MyBranch
# To change the control to new branch
$ git checkout MyBranch
# To merge two branches together
(master)$ git merge MyBranch
```



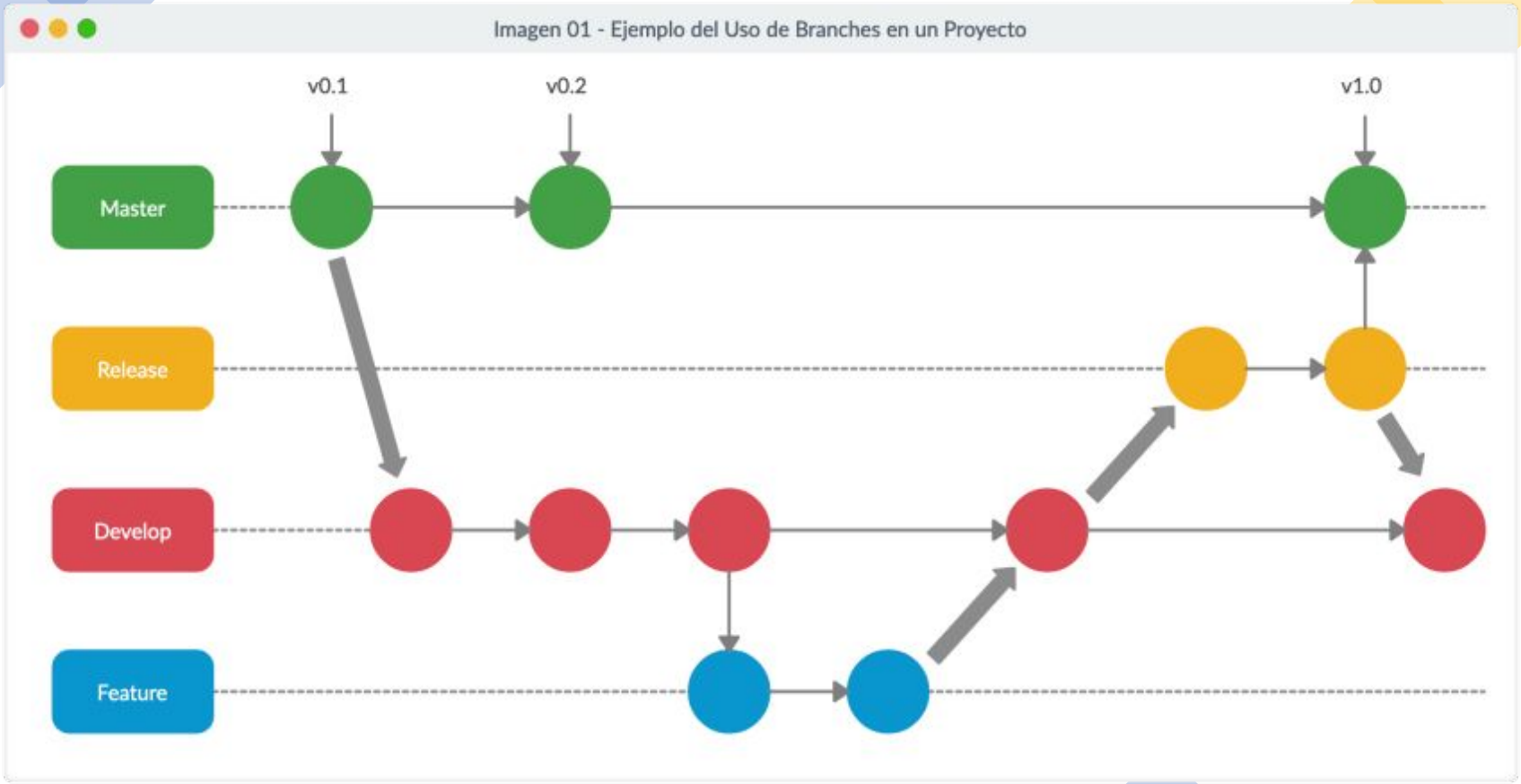
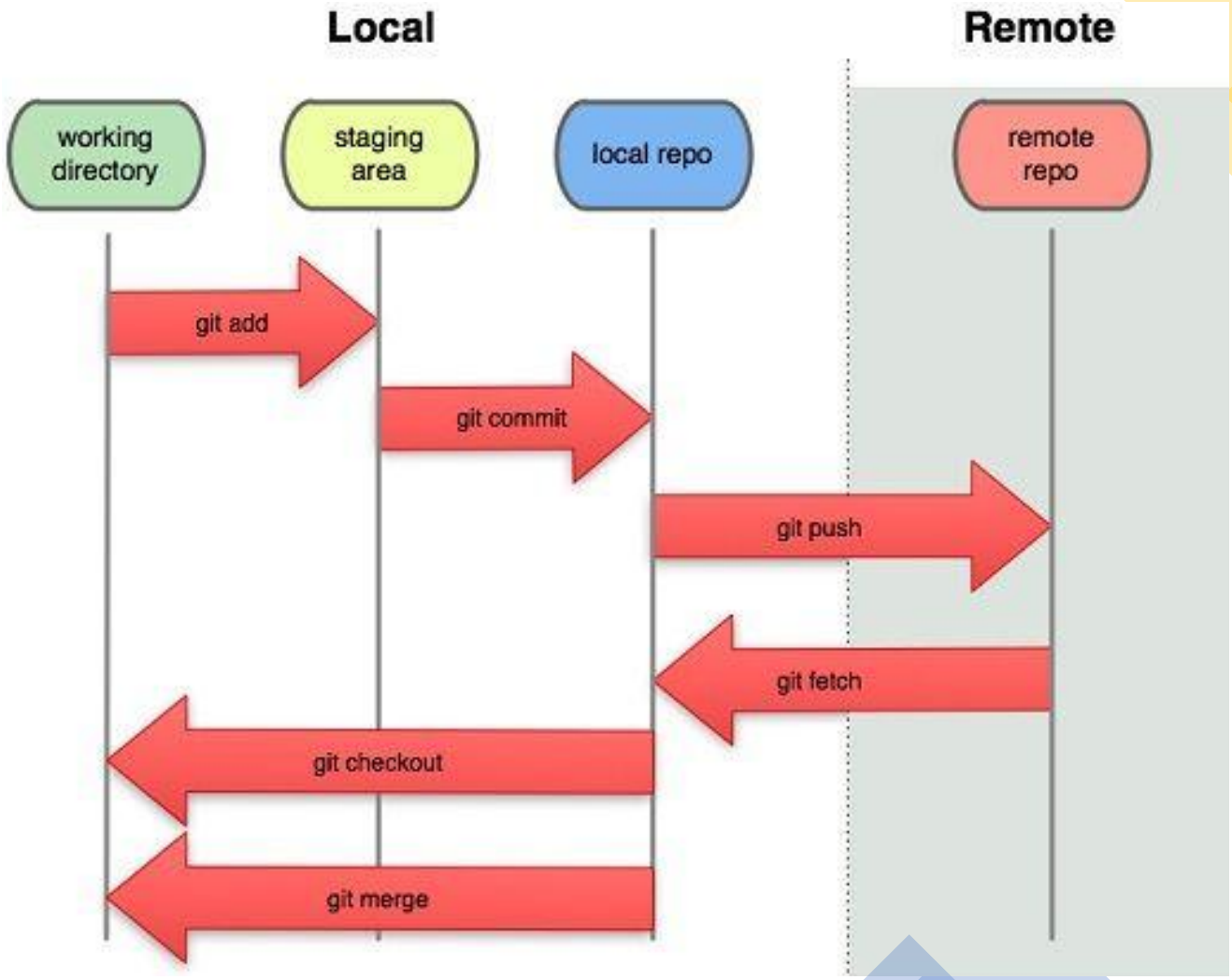


Imagen 01 - Ejemplo del Uso de Branches en un Proyecto
v0.1
v0.2
v1.0
Master
Release
Develop
Feature

Descargar / Actualizar repos

```
# To download a remote repository
$ git clone <repository url>
# To grab changes from a remote repository and add to yours
$ git pull
# To submit your changes to remote repository
$ git push
```


Local
Remote
working directory
staging area
local repo
remote repo
git add
git commit
git push
git fetch
git checkout
git merge

¡Empecemos!

Si estás en windows, abre una Git-Bash.

GitFlow

```
$ git flow init


Initialized empty Git repository in ~/project/.git/
No branches exist yet. Base branches must be created
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
```

Sin las extensiones de git-flow:

```
git checkout develop
git checkout -b feature_branch
```

Cuando se utiliza la extensión de git-flow:

```
git flow feature start feature_branch
```

Sigue trabajando y utiliza Git como lo harías normalmente.

Sin las extensiones de git-flow:

```
git checkout develop
git merge feature_branch
```

Con las extensiones de git-flow:

```
git flow feature finish feature_branch
```

Para finalizar una rama `release`, utiliza los siguientes métodos:

Sin las extensiones de git-flow:

```
git checkout main
git merge release/0.1.0
```

O con la extensión de git-flow:

```
git flow release finish '0.1.0'
```

Cuando se utilizan las extensiones de git-flow:

```
$ git flow hotfix start hotfix_branch
```

Al igual que al finalizar una rama `release`, una rama `hotfix` se fusiona tanto en `main` como en `develop`.

```
git checkout main
git merge hotfix_branch
git checkout develop
git merge hotfix_branch
git branch -D hotfix_branch
```

```
$ git flow hotfix finish hotfix_branch
```

```
git checkout main
git checkout -b develop
git checkout -b feature_branch
# work happens on feature branch
git checkout develop
git merge feature_branch
git checkout main
git merge develop
git branch -d feature_branch
```

```
git checkout main
git checkout -b hotfix_branch
# work is done commits are added to the hotfix_branch
git checkout develop
git merge hotfix_branch
git checkout main
git merge hotfix_branch
```

¡Pongámoslo en practica!