Git - Curso

Source Control Version

v1.0 - r1

SVC

Un sistema de control de versiones es aquel que registra los cambios realizados sobre un archivo o conjunto de archivos a lo largo del tiempo, de tal manera que sea posible recuperar versiones específicas más adelante.

Ejemplos de SVC

- Git (open source - descentralizado)
- Mercurial (open source - descentralizado)
- Subversion (open source - centralizado)
- Visual SourceSafe (propietario - deprecado)
- Etc... > 30

Git

Es un sistema de control de versiones creado en 2005 por Linus Torvalds, desarrollado como código libre.

Popularizado a través del éxito de la plataforma GitHub nacida en 2008 (repository hosting service). Adquirida por Microsoft en 2018.

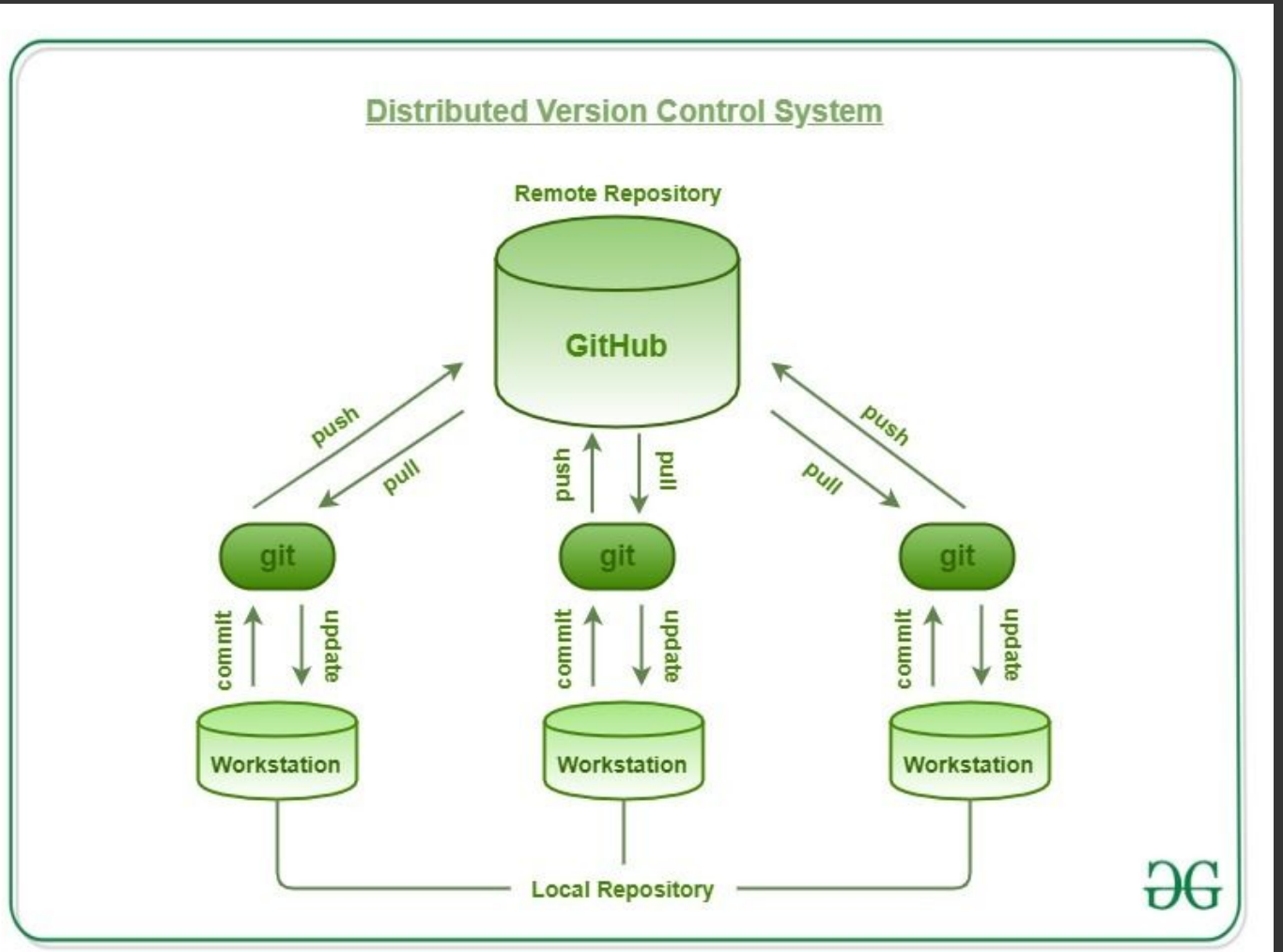
Git Repositorio

Todos los cambios gestionados a través de Git del código fuente, se almacenan en una estructura de datos llamada “Git Repository”

Git Caracteristicas

- Distribuido:

  Cada usuario trabaja con un repositorio clonado.


Distributed Version Control System
Remote Repository
GitHub
push
pull
push
pull
push
pull
git
git
git
commit
update
commit
update
commit
update
Workstation
Workstation
Workstation
Local Repository

Git Caracteristicas

- Branching model:

  Múltiples branchs para mantener versiones independientes del código.

Git Caracteristicas

- Datos seguros:

  Utiliza integridad criptográfica para guardar los commits a fin de asegurar que los datos no han sido modificados posteriormente.

Git Caracteristicas

- Gratuito
- Código abierto.

Plataformas en la nube

- GitHub
- GitLab
- Bitbucket
- Azure DevOps (Azure Repos)

Git CLI

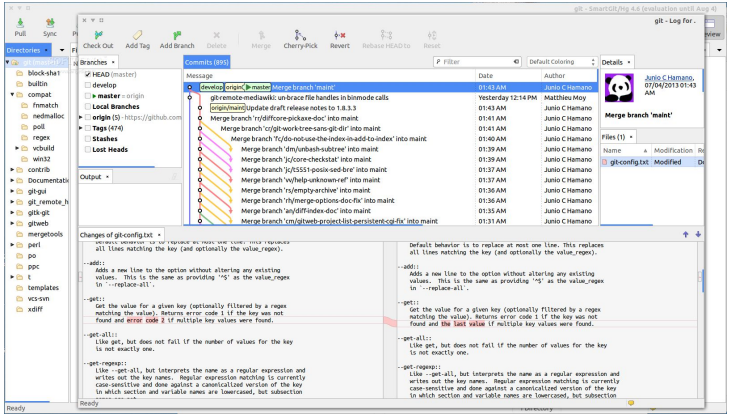
(Git Command Line Interface)

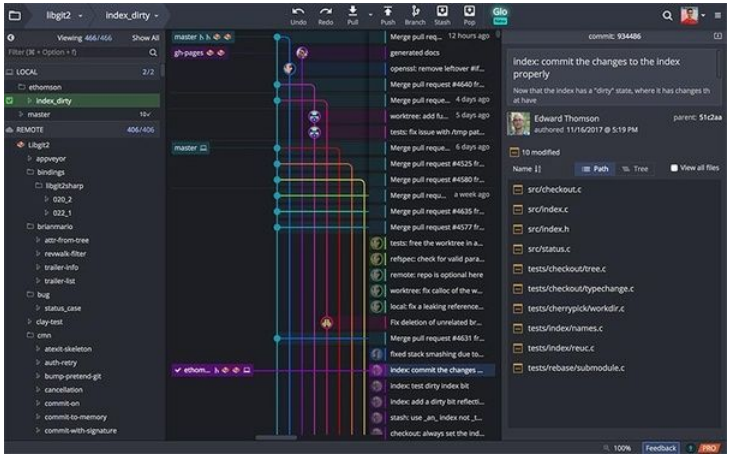
La aplicación Git ofrece una interfaz de usuario por línea de comandos.

Git GUI

(Git Graphical User Interface)

- SmartGit
- SourceTree
- GitKraken
- GitHub Desktop
- Integrado en Visual Studio
- Integrado en IntelliJ
- Plugin para VS Code
- etc..





Primeros pasos (creando)

- Iniciar repo local
- Crear commit en rama principal
- Agregar referencia de repo remoto vacío
- Sincronizar repo local con remoto

Primeros pasos (clonando)

- Clonar
- Crear branch nuevo
- Hacer commits necesarios sobre el branch
- Integrar branch nuevo a branch principal

Nombres de Branch

Deben ser descriptivos, cortos y se debe estandarizar su nomenclatura.

Mensajes de Commit

Deben ser *bien* descriptivos de las modificaciones realizadas en los archivos fuente.

En caso de estar generando un commit con trabajo incompleto (NO RECOMENDADO), utilizaremos las siglas “WIP” (work in progress) al comienzo del mensaje.

Un commit con la descripción WIP se entiende que no es operativo.

Tags

Los tags son referencias que apuntan a un commit específico.

Normalmente utilizado para indicar versiones relevantes del software.

> git tag -a <name> -m <message>

Logs

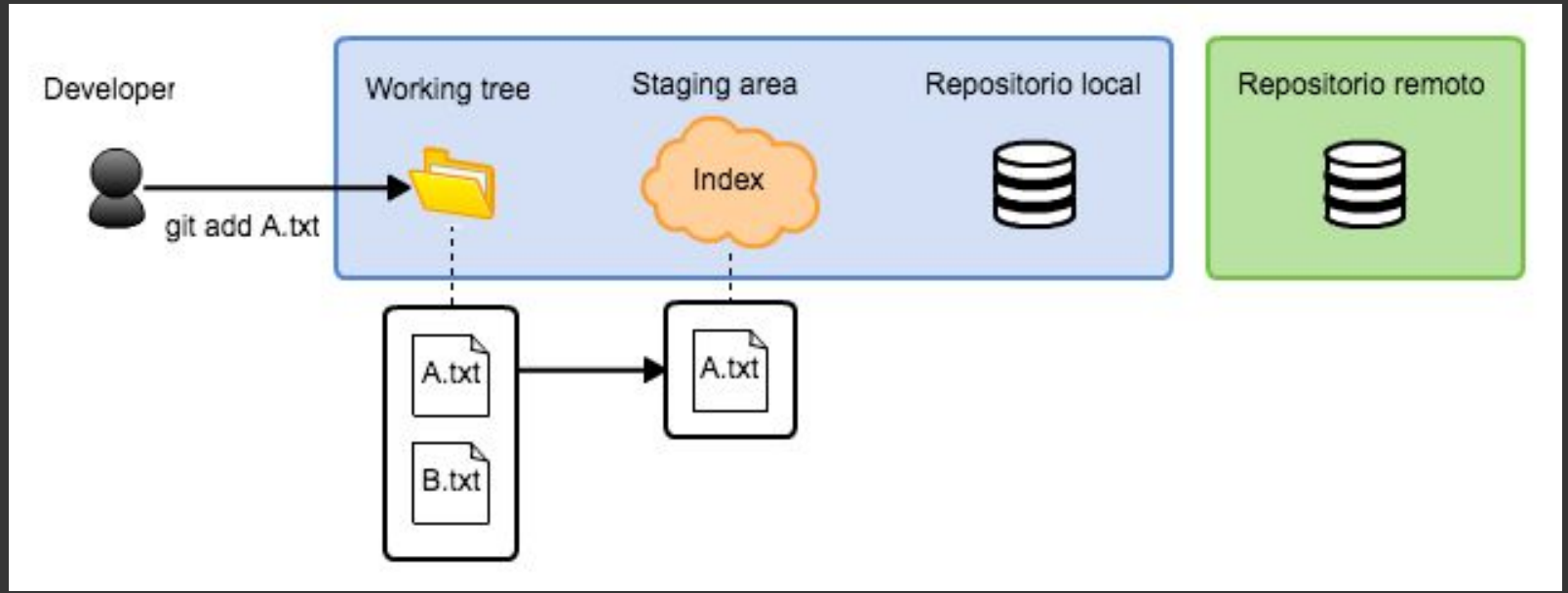
Nos permite identificar la sucesión de commits a lo largo del tiempo.

> git log

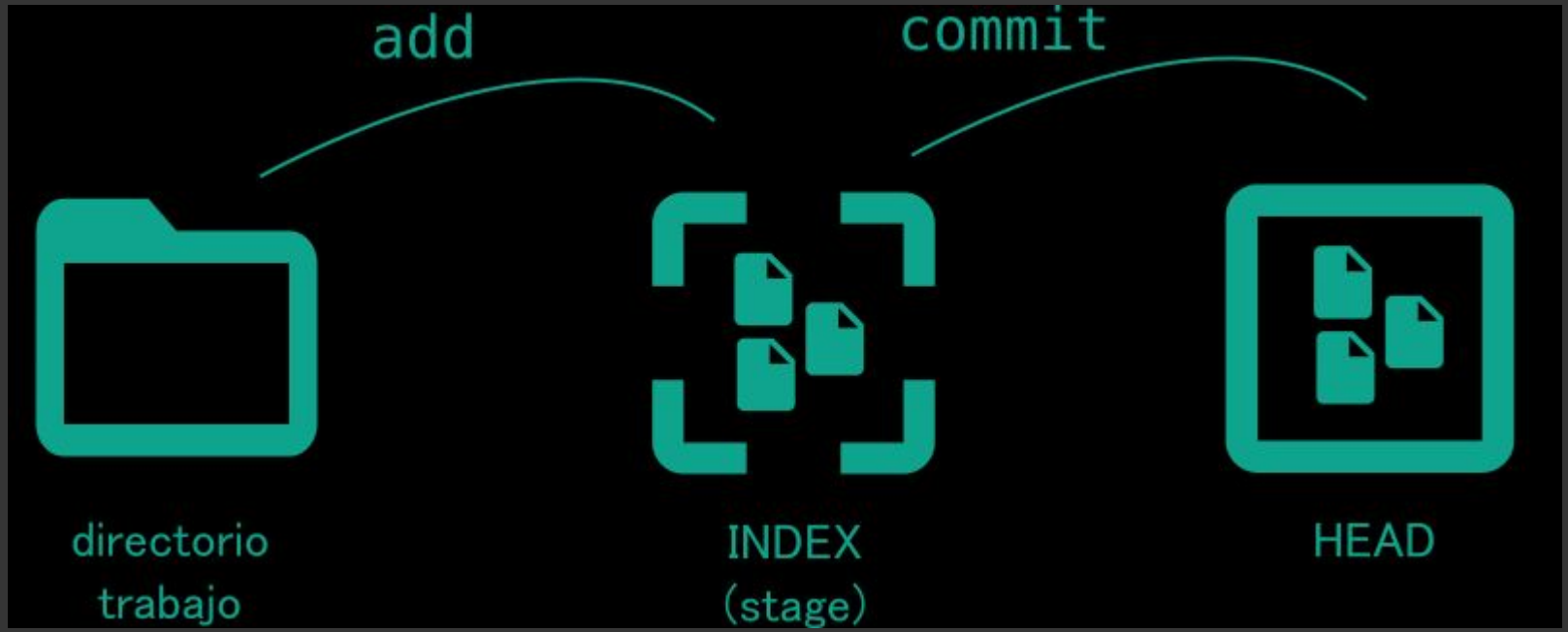
También ofrece múltiples argumentos para modificar la vista de los datos disponibles

> git log --oneline --decorate --graph --all --stat

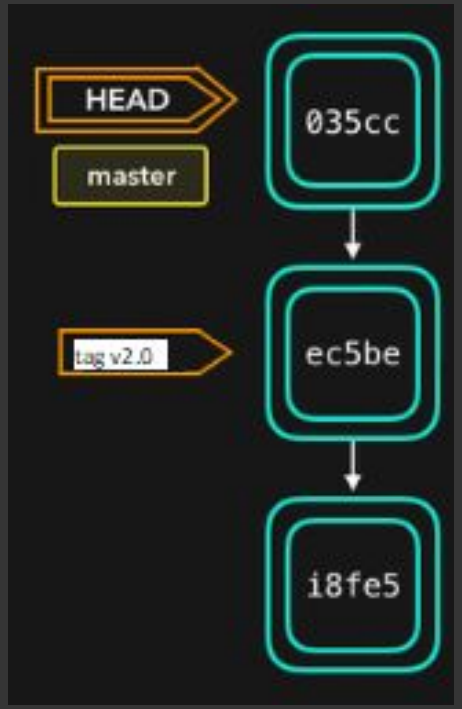
Metodología



Metodología



HEAD



HEAD es el puntero que nos indica a cual versión estamos apuntando.

En el caso que no se apunte a una versión con nombre de branch se indica que se tiene un “detached HEAD”.

Flujos de Trabajo

GitFlow

GitHub Flow

GitLab Flow

Trunk-based development

One Flow

Flujos de Trabajo

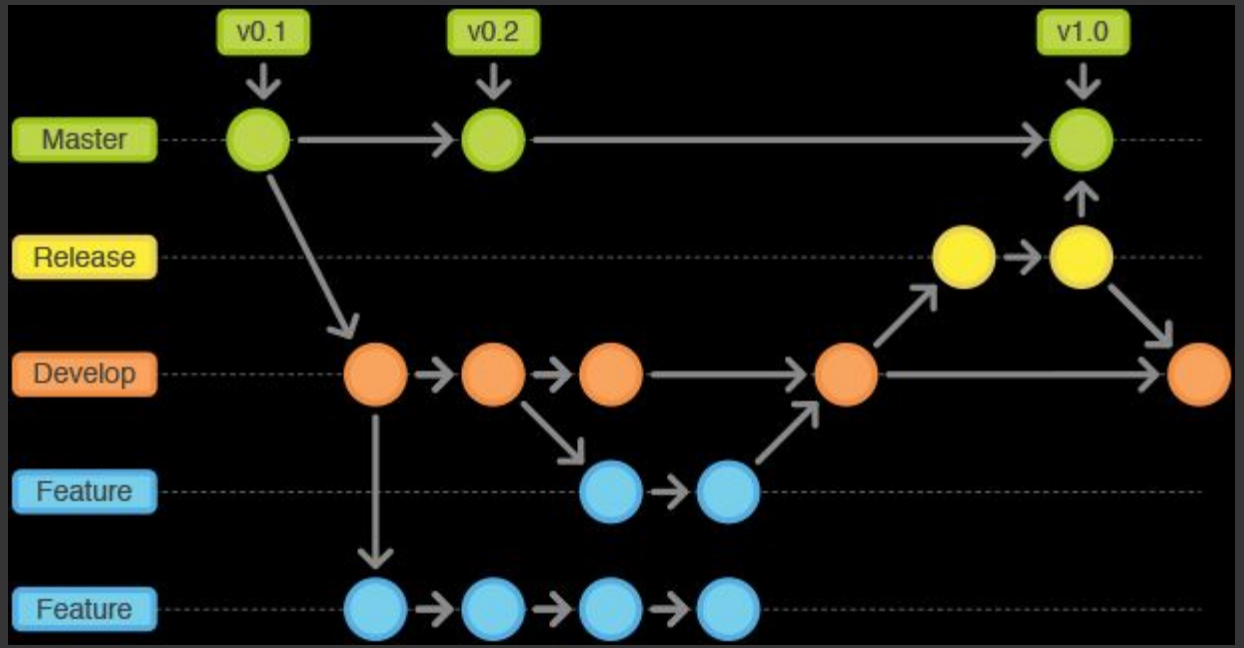
-> Git-Flow

Creado en 2010 por Vincent Driessen.

Está pensado para aquellos proyectos que tienen entregables y ciclos de desarrollo bien definidos.

Está basado en dos grandes ramas con infinito tiempo de vida (ramas *master* y *develop*) y varias ramas de apoyo, unas orientadas al desarrollo de nuevas funcionalidades (*ramas feature-**), otras al arreglo de errores (*hotfix-**).

Git-Flow



Git-Flow (I)

Master

Es la rama principal. Contiene el repositorio que se encuentra publicado en producción, por lo que debe estar siempre estable.

Development

Es una rama sacada de *Master*. Es la rama de integración, todas las nuevas funcionalidades se deben integrar en esta rama. Luego que se realice la integración y se corrijan los errores (en caso de haber alguno), es decir que la rama se encuentre estable, se puede hacer un merge de development sobre la rama *Master*.

Git-Flow (II)

Features

Cada nueva funcionalidad se debe realizar en una rama nueva, específica para esa funcionalidad. Estas se deben sacar de *Development*. Una vez que la funcionalidad esté desarrollada, se hace un merge de la rama sobre *Development*, donde se integrará con las demás funcionalidades.

Git-Flow (III)

Hotfix

Son errores de software que surgen en producción, por lo que se deben arreglar y publicar de forma urgente. Es por ello, que son ramas sacadas de *Master*. Una vez corregido el error, se debe hacer una unificación de la rama sobre *Master*. Al final, para que no quede desactualizada, se debe realizar la unificación de *Master* sobre *Development*.

Release

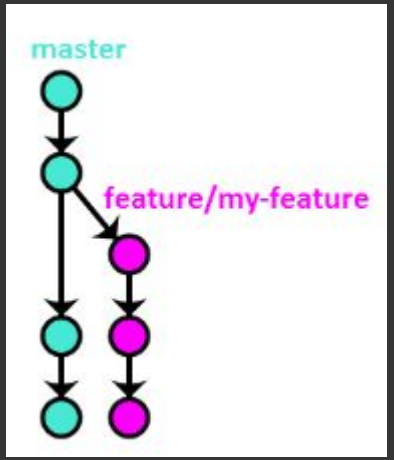
Las ramas de release apoyan la preparación de nuevas versiones de producción. Para ellos se arreglan muchos errores menores y se preparan adecuadamente los metadatos. Se suelen original de la rama develop y deben fusionarse en las ramas master y develop.

One-Flow (I)

Master (como eternal branch)

Feature

Hotfix



Lecturas Adicionales (I)

Git Docs

https://git-scm.com/doc

Git-Flow

https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow

Wiki

https://es.wikipedia.org/wiki/Git

Bitbucket Tutorial

https://www.atlassian.com/git/tutorials/learn-git-with-bitbucket-cloud

Lecturas Adicionales (II)

One Flow

https://www.endoflineblog.com/oneflow-a-git-branching-model-and-workflow

Visualización de comandos

https://dev.to/lydiahallie/cs-visualized-useful-git-commands-37p1

Git para Windows

https://git-scm.com/downloads

Git Cheat Sheet

https://education.github.com/git-cheat-sheet-education.pdf

Demo (I)

1 Crear repositorio vacío en github

2 Agregar colaboradores

3 Crear repo local

4 Crear primer commit

5 Agregar referencia de repo remoto

6 Sincronizar via push

Demo (II)

7 Crear nueva rama

8 Pushear nueva rama a repo remoto

9 Generar commits sobre rama

10 Hacer merge en main

11 Hacer push de main para sincronizar con remoto

Demo (III) (Clonando)

1 Clonar repo

2 Continuar con el flujo de la diapositiva anterior

Demo (IV) (Resolver conflictos de merge)

1 Disponer de dos ramas que modifican las mismas partes de un archivo

2 Generar un merge con conflictos

3 Resolver conflictos

4 Hacer commit y push

Comandos básicos (I)

Git init: crear repo local.

Git add <files>: genera stage con los archivos indicados (previo commit).

Git commit -m <message>: crea commit sobre la rama actual.

Git status: verificamos el estado actual del repo local.

Git branch <*branch_name*>: crea nuevo branch.

Git checkout <*branch_name*>: nos posiciona en branch indicado.

Comandos básicos (II)

Git push: sube commit de branch actual a repo remoto.

Git fetch: descarga al repo local información de branchs de repo remoto.

Git pull: descarga últimos commits de branch remoto al branch local actual.

Git merge <branch_name_to_merge>: trae los cambios del branch indicado al branch actual.

Git log: nos da información del histórico de commits.