Noviembre 2023

Presentación

- Nombre
- Puesto de trabajo
- Experiencia
- Tecnologías
- Conocimiento y experiencia con controles de versiones
- Conocimiento y experiencia específicos de GIT
- Correo a utilizar para el curso
- Contacto: formacion.ecs.es@arrow.com

Software necesario

- GIT - https://git-scm.com
- SourceTree - https://www.sourcetreeapp.com/
- VS Code - https://code.visualstudio.com/

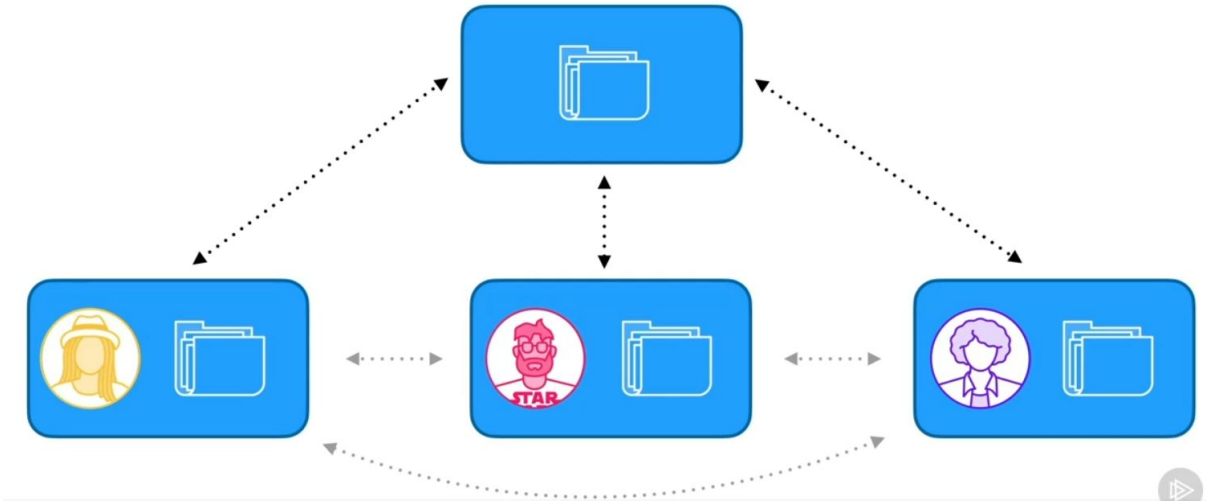
Introducción a GIT

- ¿Qué es GIT?
- Conceptos básicos teóricos de GIT
  - Control de versiones distribuido
  - Características de GIT
  - Ficheros en GIT y el estatus
- Introducción a línea de comando
- Comandos básicos de GIT
  - config (configurar entorno)
  - init
  - add
  - status
  - commit
  - log
  - diff

¿Qué es GIT?

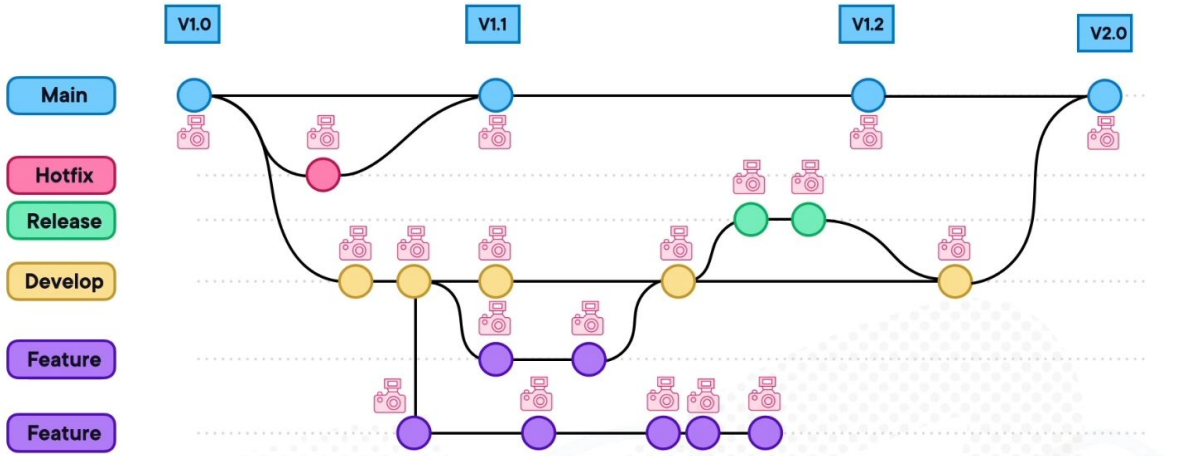
- GIT es un sistema de control de versiones diseñado para manejar todo tipo de proyectos de todo tamaño con velocidad y eficiencia
- Permite volver a cualquier estado del proyecto en cualquier momento del tiempo
- Un clon local del proyecto es un repositorio de control de versiones completo

¿Qué es un sistema de control de versiones distribuido?



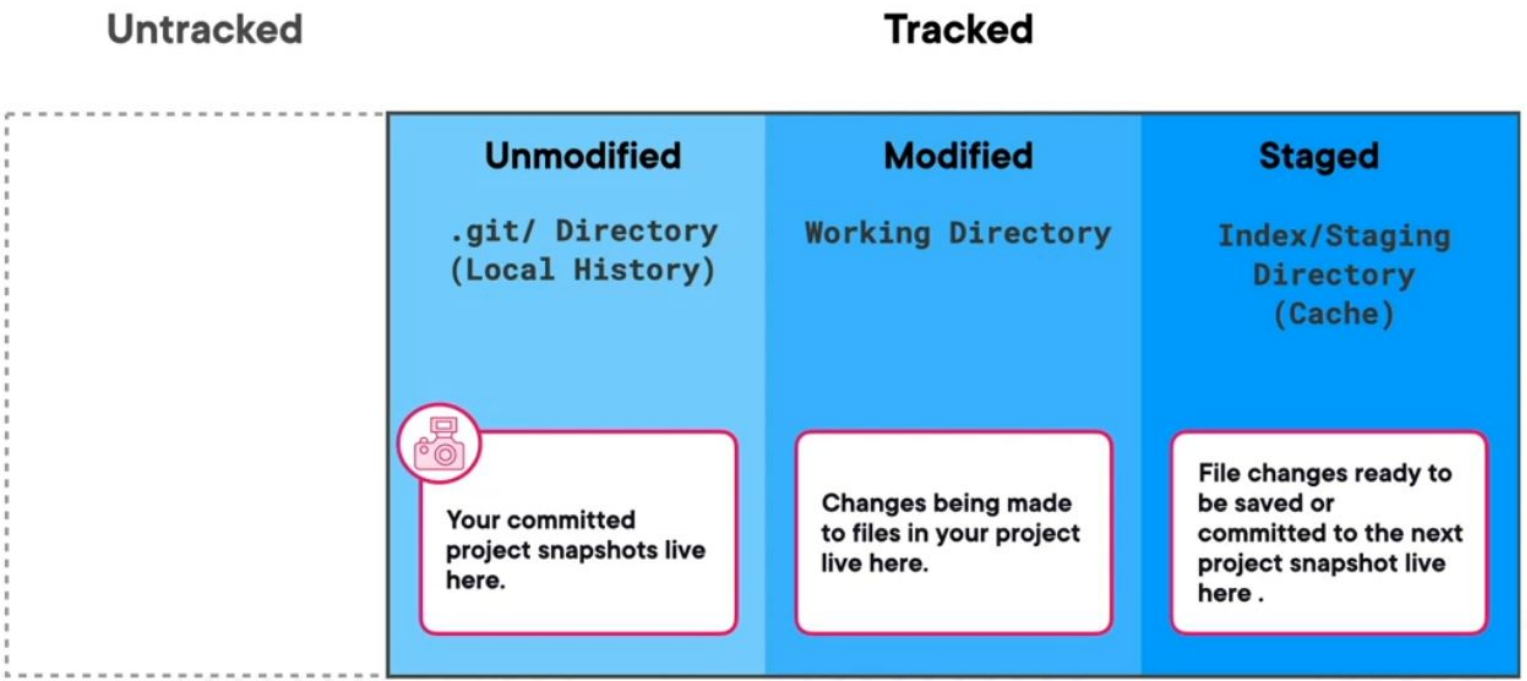
Características de GIT

- Se puede trabajar completamente offline
- Cualquier cambio es intencionado: GIT no realiza acciones por sí solo
- Está diseñado para desarrollos no lineales
- Snapshots: manera en que se guarda el estado de algo en un punto específico del tiempo

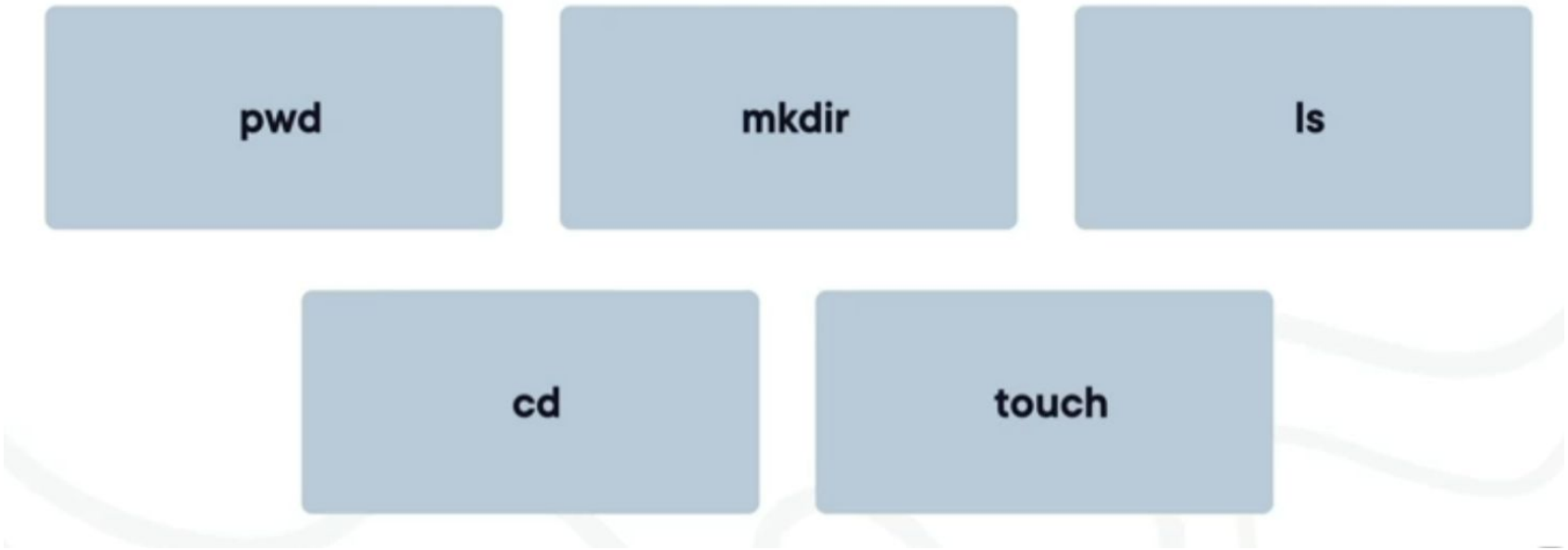


Ficheros en GIT: status

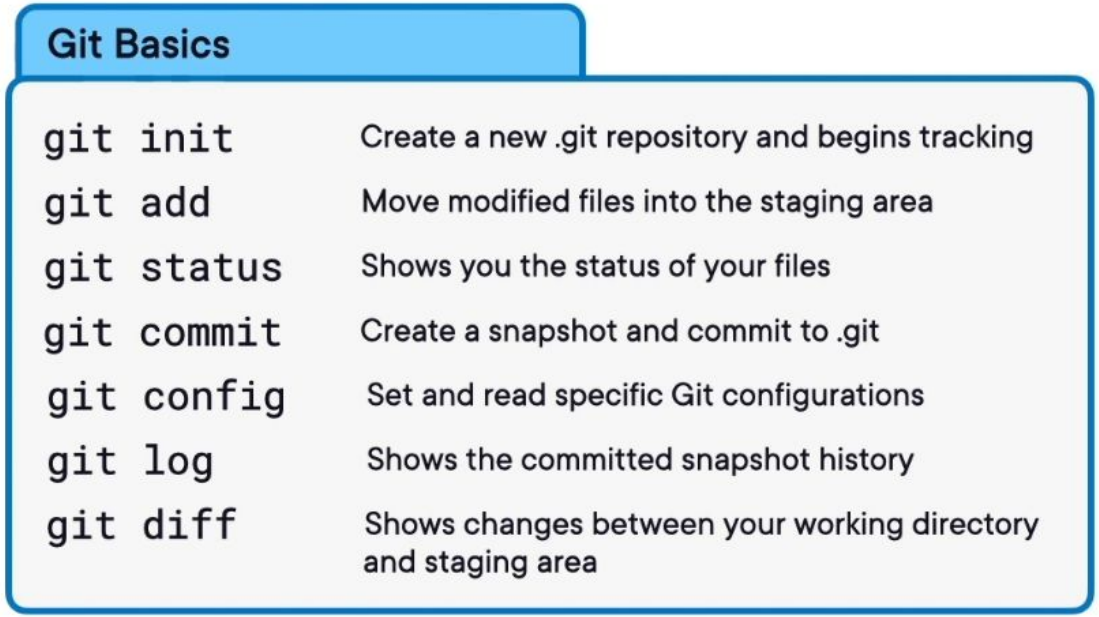
Directorios en GIT: status



Introducción a línea de comando



Comandos básicos de GIT



Comandos de GIT

Git Basics

`git init`	Create a new .git repository and begins tracking
`git add`	Move modified files into the staging area
`git status`	Shows you the status of your files
`git commit`	Create a snapshot and commit to .git
`git config`	Set and read specific Git configurations
`git log`	Shows the committed snapshot history
`git diff`	Shows changes between your working directory and staging area

Remote Repositories

`git clone`	Copies an entire repository into a new local .git directory
`git remote`	Create and show linked repositories
`git push`	Send updates to associate repositories
`git pull`	Retrieves and integrates changes from other repositories
`git fetch`	Retrieves but doesn't integrate changes from other repositories

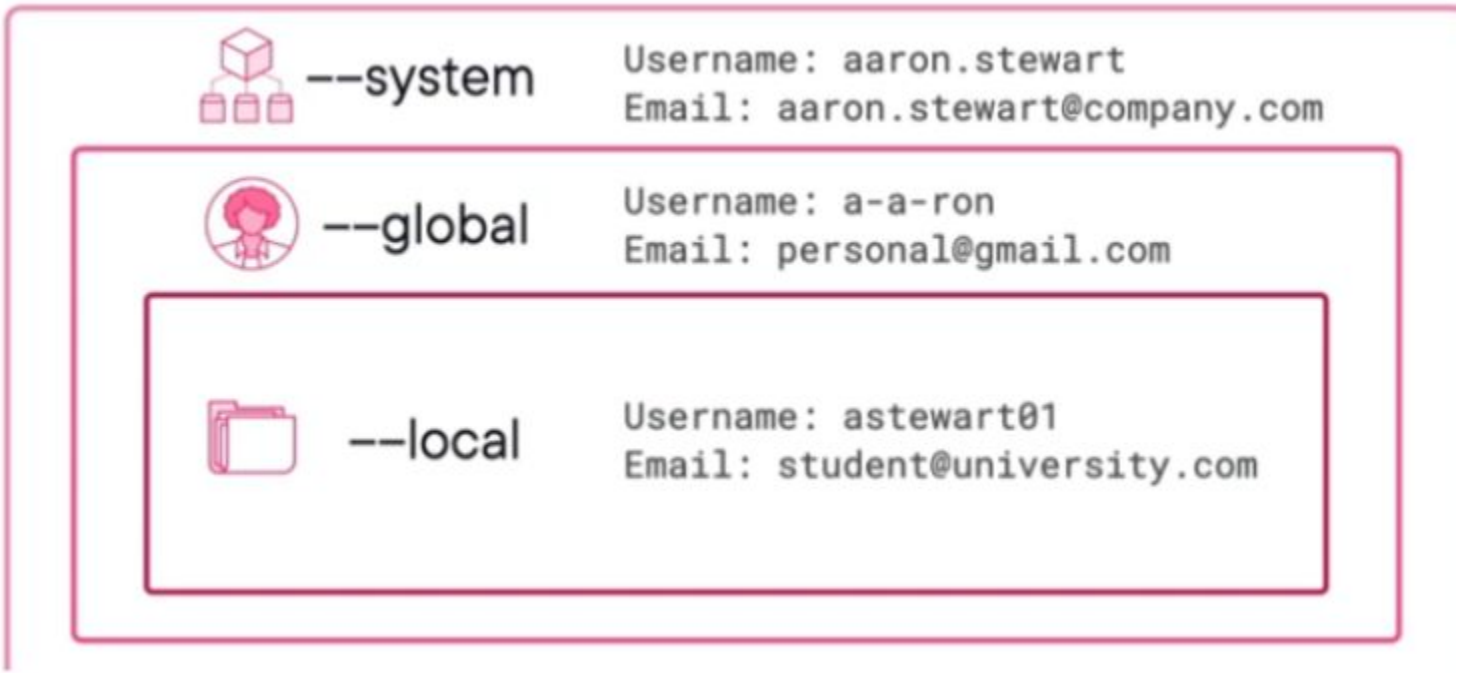
Git Branches

`git branch`	List, create, or delete branches
`git checkout`	Switch between branches
`git merge`	Bring changes from one branch into another

Undoing Changes

`git revert`	Create a new commit that undoes a previous commit
`git reset`	Remove files from the staging area

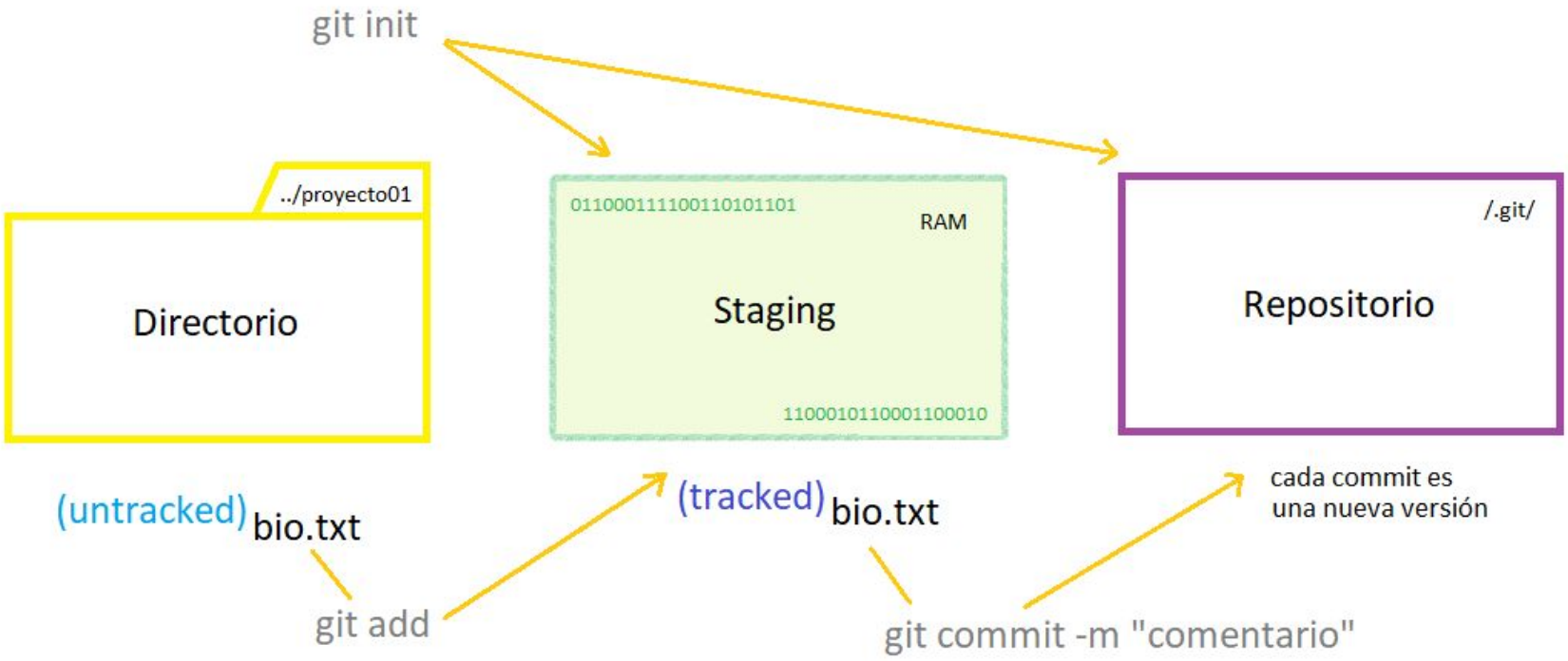
Niveles de configuración de GIT



Conceptos básicos de GIT

- Instalación de GIT
- Uso de git bash
- Niveles de configuración de GIT
- Diferencia entre windows y unix based
- Comandos y ayuda
- Uso de tags
- Versiones
- Releases
- Versiones vs Releases

Repositorio Local



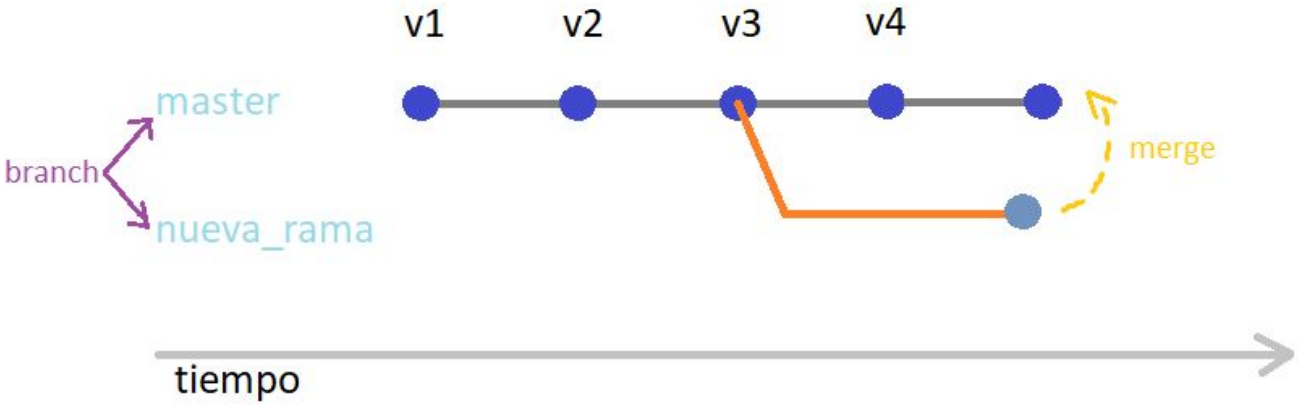
Repositorios y flujo de trabajo

- Trabajar con otros en un repositorio compartido
- Conflictos con ficheros: qué son y cómo arreglarlos
- Modificar y arreglar commits
- Casos prácticos
    - Flujos de trabajo
- Malas prácticas

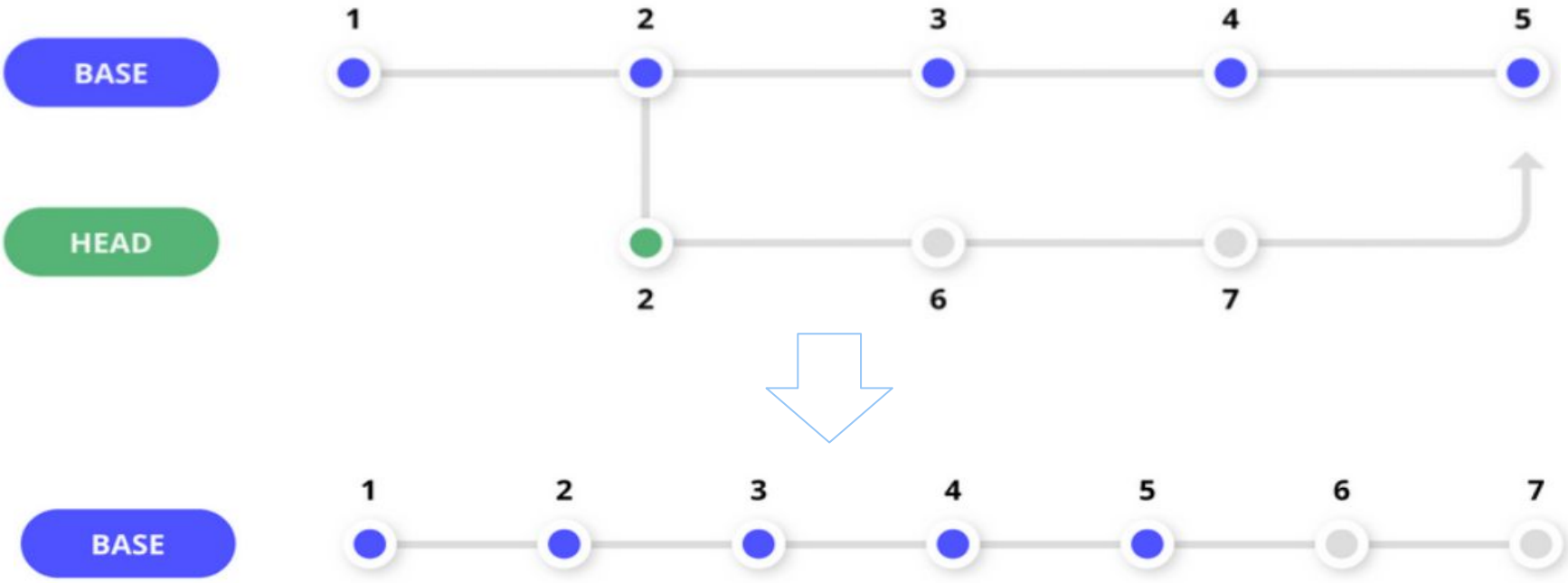
Repositorios remotos - GitHub

- ¿Qué es GitHub?
- Instalación de GITHUB
- Creación de repos
- Clonación de repositorios
- Trabajo en equipo
- Uso de grupos y administración
- Permisos
- GitHub pages
- Alternativas a GitHub

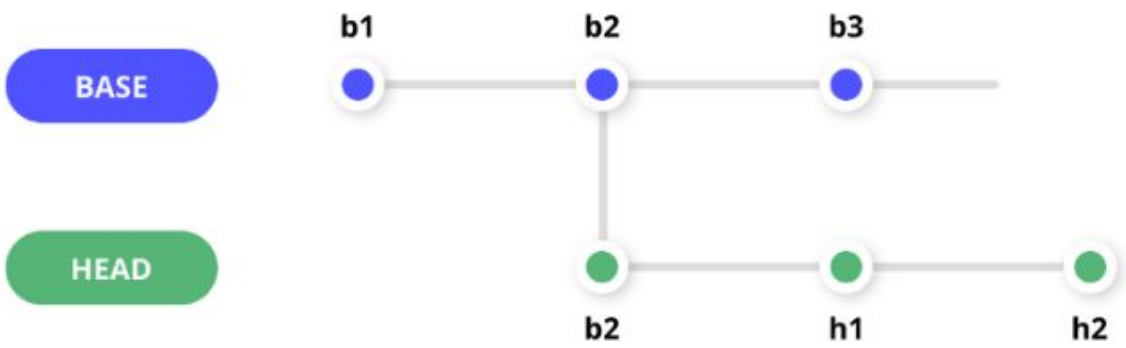
Ramas (branch)



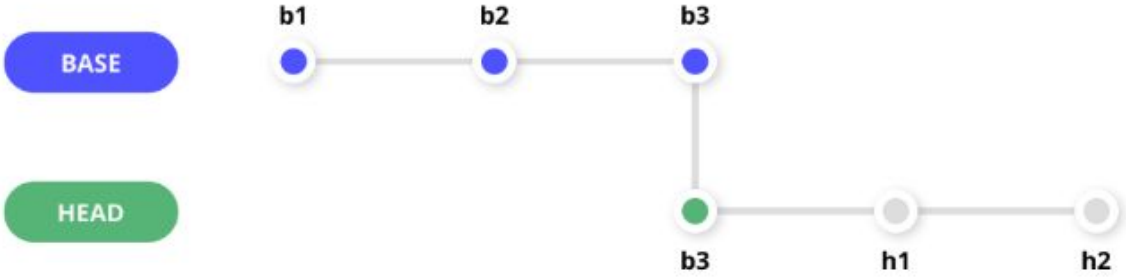
Merge



Rebase



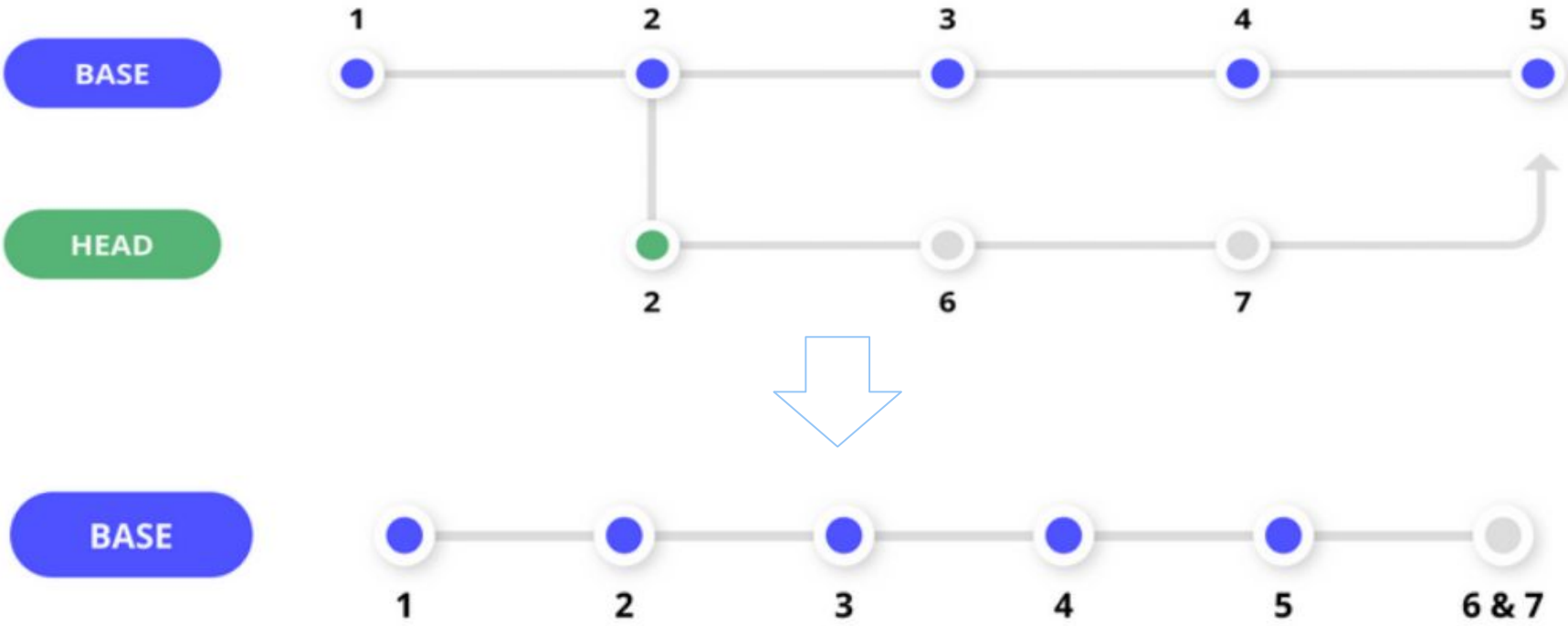
Estado inicial



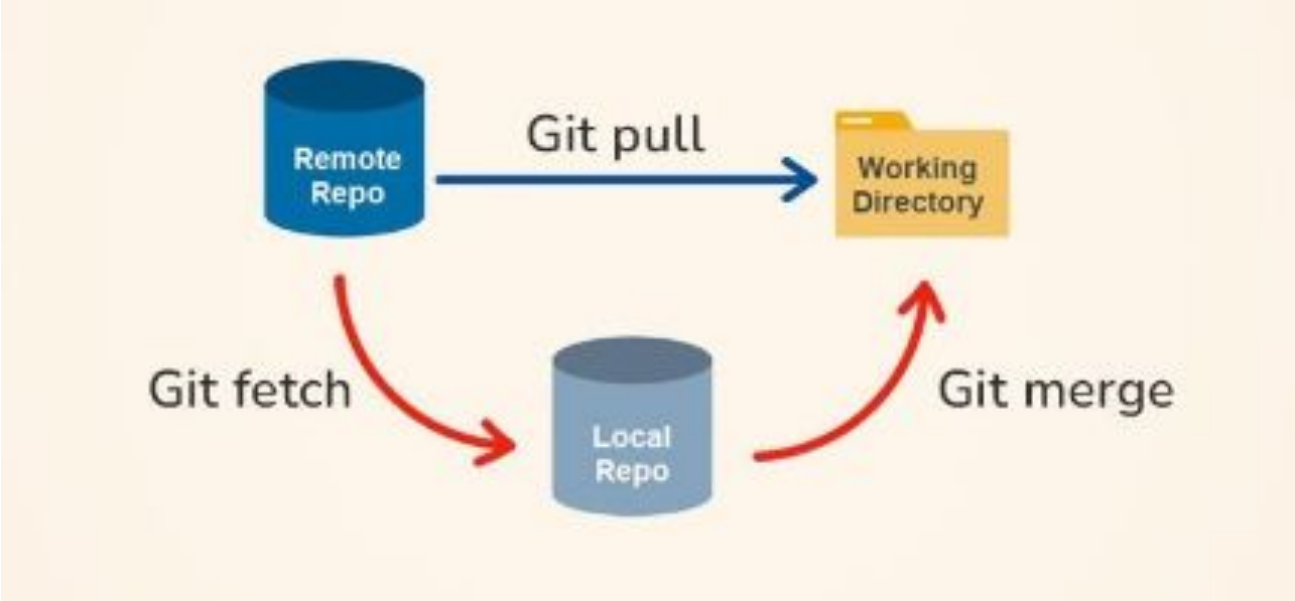
Tras aplicar rebase + merge de la rama Base a la rama Head

Tras aplicar rebase + merge de la rama head a la rama Base

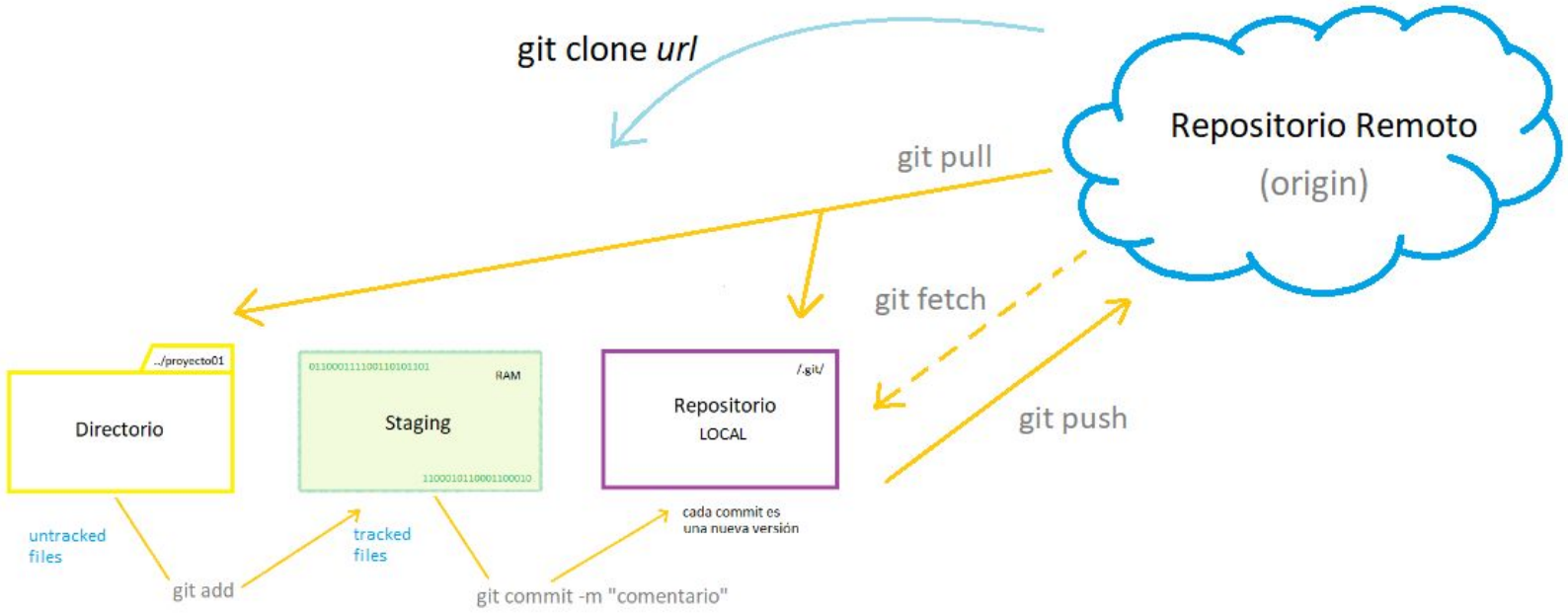
Squash



Pull vs Fetch & Merge



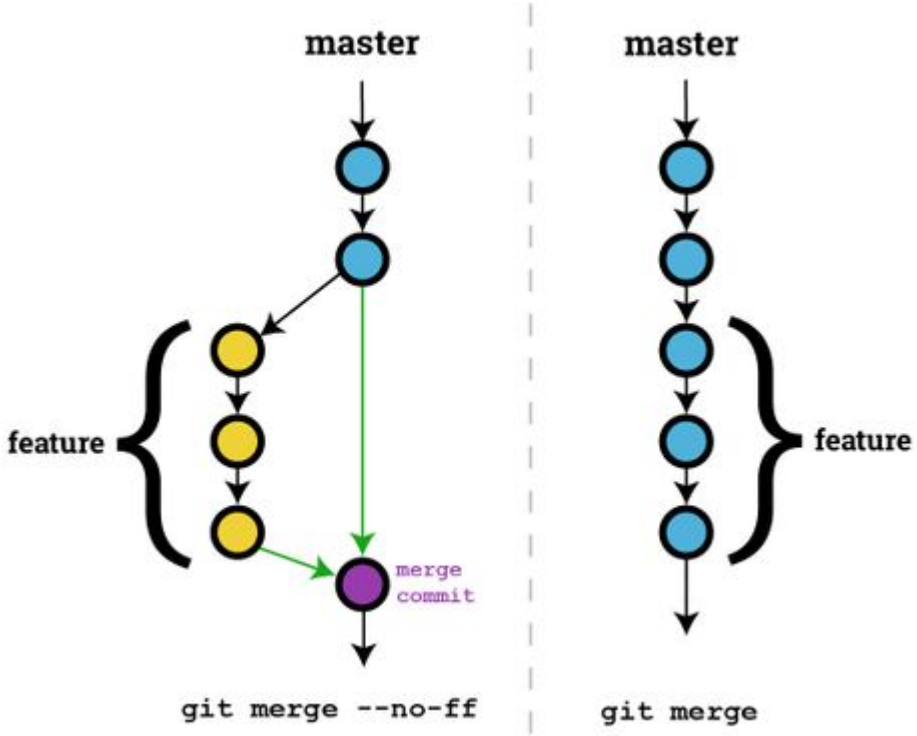
Repositorio Local vs Remoto



Fast-Forward vs No Fast-Forward

En el primer caso, el *merge* fusiona lo cambios y añade los *commits* de la rama *Head* en la rama *Base*. Esto es posible siempre y cuando los commits se produzcan en la rama *Head*. Básicamente lo que se produce es un desplazamiento del puntero (*fast-forward*) de la rama *Base* al último *commit* de la rama *Head*.

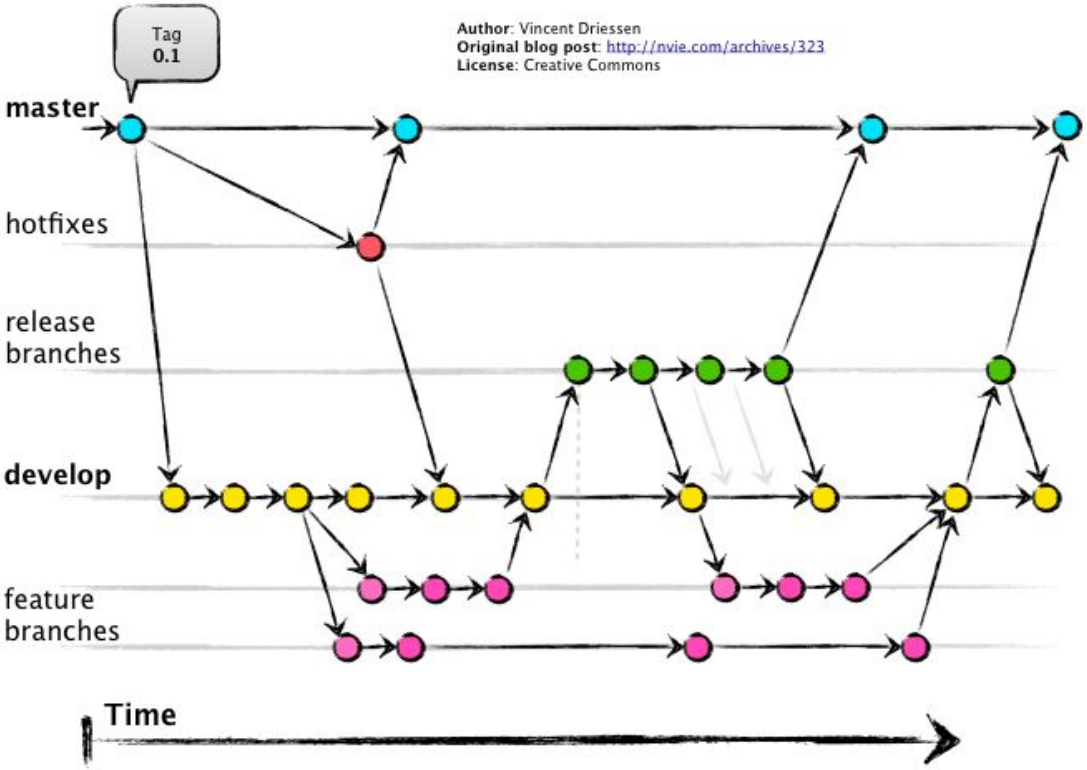
En el segundo caso, con el *merge* se añade un *commit* adicional en la rama *Base* que deja constancia de la unión de la rama *Head* con la rama *Base*. Este *commit* adicional aparece, bien porque lo forzamos para que no se produzca un *fast forward* (*--no-ff*); o bien porque la rama *Base* también contenía cambios, existiendo cambios en ambas ramas en el momento de la fusión.



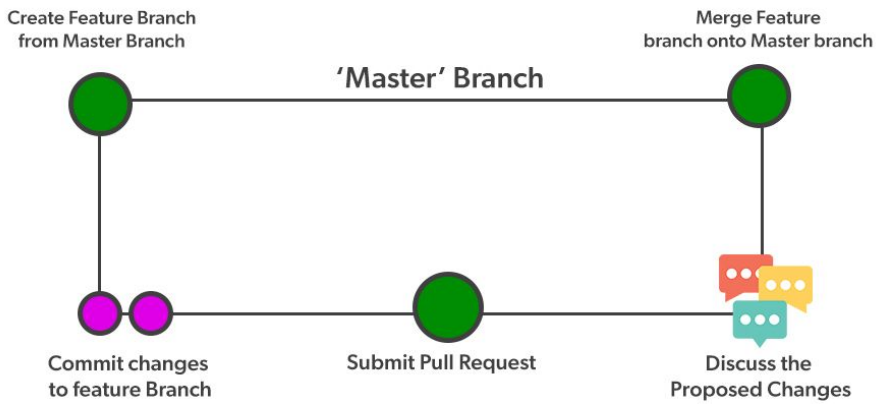
Estrategias de branching

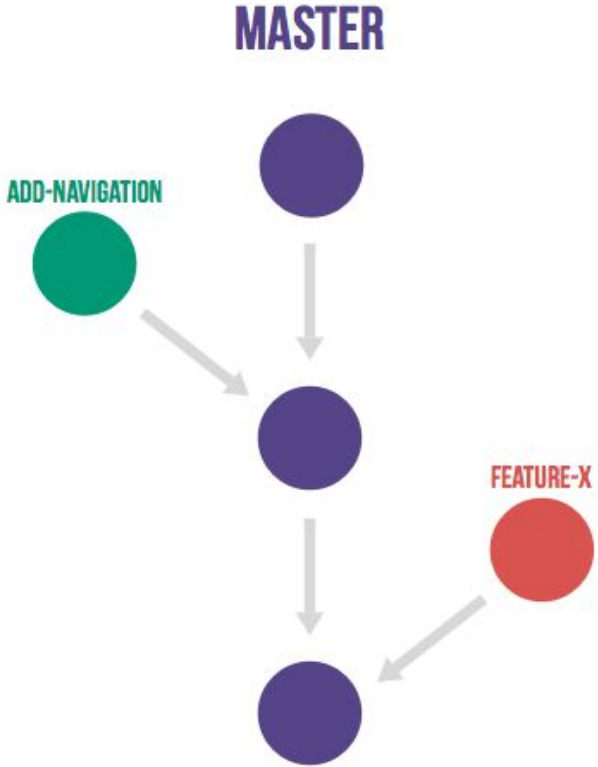
- Mainline based
- Branch based
  - GitFlow - el habitual
    - main
    - develop
    - feature
    - release
    - hotfix
  - GitHub flow - igual pero sin release
  - GitLab flow - añade ramas por entorno
  - OneFlow - no tiene develop

Git Flow

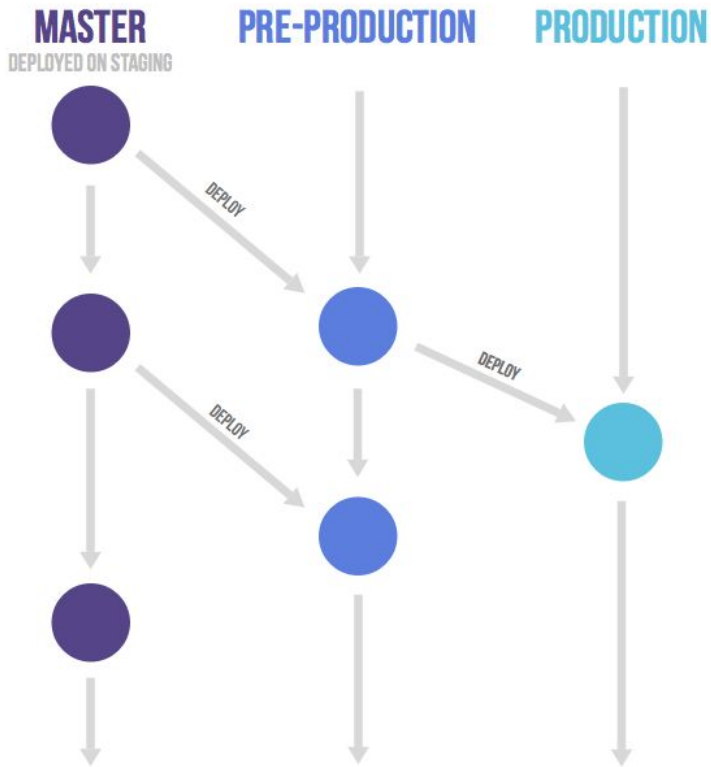

Tag
0.1
Author: Vincent Driessen
Original blog post: http://nvie.com/archives/323
License: Creative Commons
master
hotfixes
release
branches
develop
feature
branches
Time

GitHub Flow


Create Feature Branch
from Master Branch
'Master' Branch
Merge Feature
branch onto Master branch
Commit changes
to feature Branch
Submit Pull Request
Discuss the
Proposed Changes


MASTER
ADD-NAVIGATION
FEATURE-X

GitLab Flow


MASTER
DEPLOYED ON STAGING
PRE-PRODUCTION
PRODUCTION
DEPLOY
DEPLOY
DEPLOY

One Flow