



Universidad
Francisco de Vitoria
UFV Madrid

Desarrollo e Integración del Software

Tema 1

Control de versiones. Git.

*Grado en Ingeniería Informática
Escuela Politécnica Superior*



Universidad
Francisco de Vitoria
UFV Madrid

Control de versiones

*Grado en Ingeniería Informática
Escuela Politécnica Superior*

¿Que es el control de versiones?

- Un sistema que permite almacenar los cambios realizados en diferentes ficheros.
- Permite el desarrollo colaborativo.
- Permite conocer quién realizó los cambios y cuando lo hizo.
- **Permite revertir los cambios a un estado anterior.**

Existen diferentes sistemas de control de versiones, el mas famoso y extendido a día de hoy es Git.

Tipos de gestores de versiones

- Existen distintas aproximaciones para el control de versiones:
 - Locales.
 - Centralizados.
 - Distribuidos.

Gestores locales

- Consiste en la copia de los ficheros a otro directorio.
- Enfoque muy común, pero **propenso a errores**.
- Es fácil olvidar el directorio en que nos encontramos y borrar o sobrescribir información.
- La solución a este problema fue crear VCS locales con base de datos que contenían el registro de cambios aplicados a los ficheros.

Gestores centralizados

- Permiten la colaboración entre desarrolladores de distintos sistemas.
- Tienen un único servidor que contiene todos los archivos versionados.

Gestores centralizados

- Ventajas:
 - Todo el mundo puede saber en qué está trabajando cada desarrollador.
 - Los administradores tienen un control detallado de lo que puede hacer cada usuario.
 - Mucho más fácil de gestionar.

- Desventajas:
 - Obvio: **El servidor centralizado.**
 - Si el servidor cae, no se puede colaborar ni guardar cambios versionados.
 - Si el soporte de la BBDD falla, se pierde absolutamente todo (necesidad de copias de seguridad) salvo la copias locales

- Los clientes no sólo descargan la última instantánea de los archivos: **replican completamente el repositorio.**
 - Ventajas: Si el servidor cae, se puede restaurar con la copia local de algún usuario.



Universidad
Francisco de Vitoria
UFV Madrid

Git

*Grado en Ingeniería Informática
Escuela Politécnica Superior*

- Nació en Abril de 2005
- Creado por Linus Torvalds para ayudarlo durante el desarrollo del kernel de Linux.
- Principalmente es usado para mantener el histórico de los cambios en un proyecto de desarrollo, pero puede utilizarse para cualquier tipo de fichero.



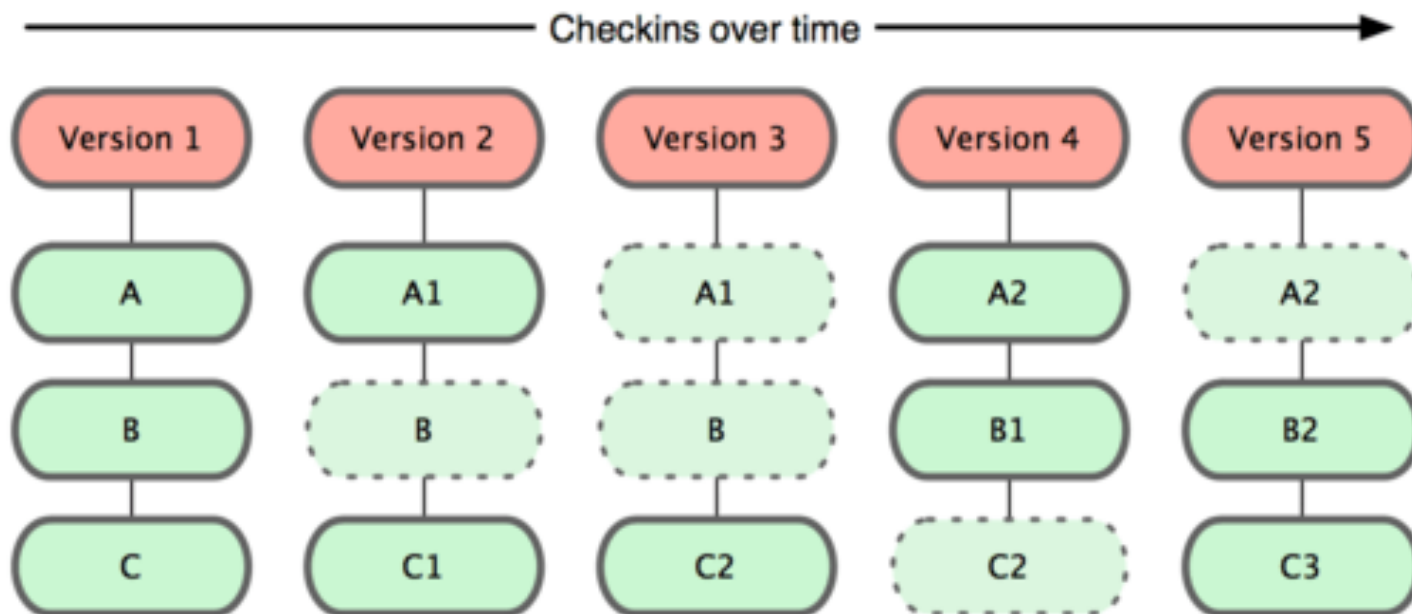
Git. Principales características

- Gestión distribuida. Cada usuario dispone de una copia local del historial completo.
- Gestión eficiente de proyectos grandes.
- La gran mayoría de grandes proyectos Opensource están alojados en Git.

- Git modela sus datos como un conjunto de instantáneas de un mini sistema de archivos.
- Hace una foto del aspecto de todos tus archivos en ese momento, y guarda una referencia a esa instantánea.

Git. Almacenamiento de la información

- Si los archivos no se han modificado, Git no almacena el archivo de nuevo, sólo un enlace al archivo anterior idéntico que ya tiene almacenado.



- La mayoría de las operaciones en Git sólo necesitan archivos y recursos locales para operar (*Mayor velocidad que la competencia*)
- Para navegar por la historia del proyecto se accede a la BD local.
- Visualizar los cambios de un fichero de hace un mes implica únicamente un cálculo de diferencias de forma local

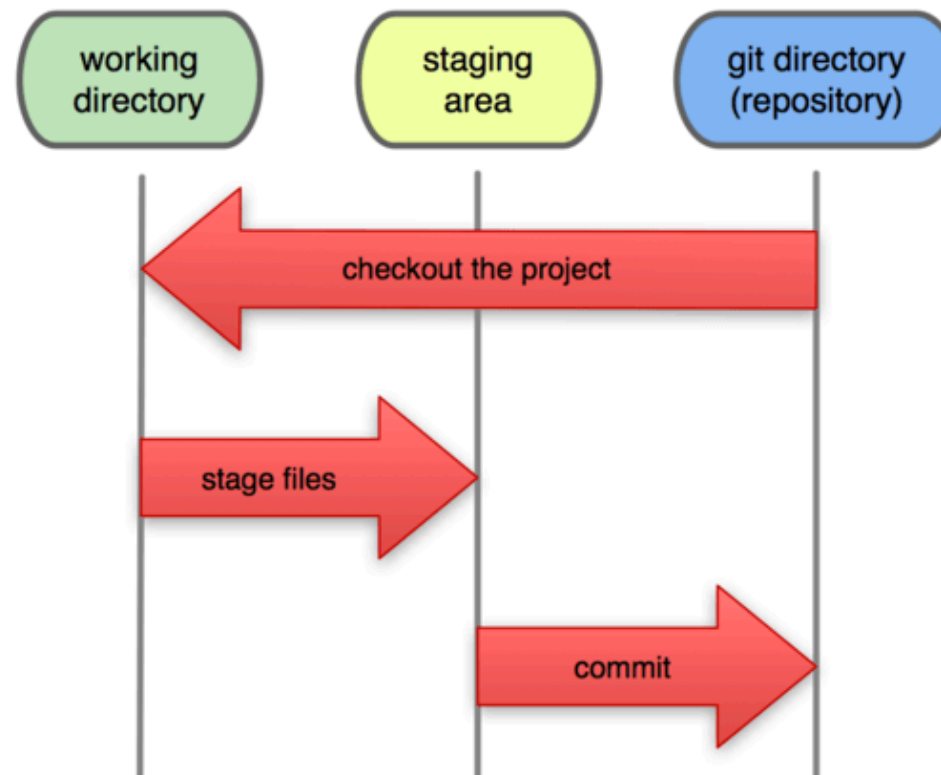
- Todo es verificado mediante una suma de comprobación (*checksum*) antes de ser almacenado, y es identificado a partir de ese momento mediante dicha suma.
 - Es imposible cambiar los contenidos de cualquier archivo o directorio sin que Git lo sepa.
- Git guarda todo por el valor hash de sus contenidos.

Git. Los tres estados.

- Confirmado (**committed**) significa que los datos están almacenados de manera segura en tu base de datos local.
- Modificado (**modified**) significa que has modificado el archivo pero todavía no lo has confirmado a tu base de datos.
- Preparado (**staged**) significa que has marcado un archivo modificado en su versión actual para que vaya en tu próxima confirmación.

Git. Los tres estados


Local Operations





- El **directorio de Git** (.git en la carpeta del proyecto) es donde se almacenan **los metadatos** y la base de datos de objetos para tu proyecto. Es lo que se copia cuando clonas un repositorio desde otro ordenador.
- El **directorio de trabajo** es una copia de una versión del proyecto. Estos archivos se sacan de la base de datos comprimida en el directorio de Git, y se colocan en disco para que los puedas usar o modificar.
- El **área de preparación** es un sencillo archivo, generalmente contenido en tu directorio de Git, que almacena información acerca de lo que va a ir en tu próxima confirmación.


- El flujo de trabajo básico en Git es algo así:
 - Modificas una serie de archivos en tu directorio de trabajo.
 - Preparas los archivos, añadiéndolos a tu área de preparación (staging).
 - Confirmas los cambios (commit), lo que toma los archivos tal y como están en el área de preparación, y almacena esas instantáneas de manera permanente en tu directorio de Git.

Git. Los tres estados

in case of fire 

 git commit

 git push

 git out

Git. Alojamiento Git

Estos son algunos de los alojamientos git más importantes hoy en día:

- Github
- Bitbucket
- Gitlab
- SourceForge



- Github es el proveedor de alojamiento de repositorios Git más grande a día de hoy.
- Más de 40 millones de usuarios.
- Recientemente comprado por Microsoft por 7,5B\$. (Junio 2018).



Git. Interfaces gráficas

- Git puede utilizarse tanto desde un terminal como desde una interfaz gráfica.
- Existen muchos clientes gráficos para Git.
 - SourceTree está muy extendido y es gratuito. (Atlassian)
 - Github dispone de un cliente para su plataforma.
 - Git Kraken y Tower son clientes multiplataforma (Mac OS & Windows) modernos y muy potentes.
- Una interfaz gráfica simplifica muchas veces el uso de Git, sobretodo ante conflictos que aparecen durante su uso, sin embargo es esencial entender los comandos que la UI ejecuta por nosotros.

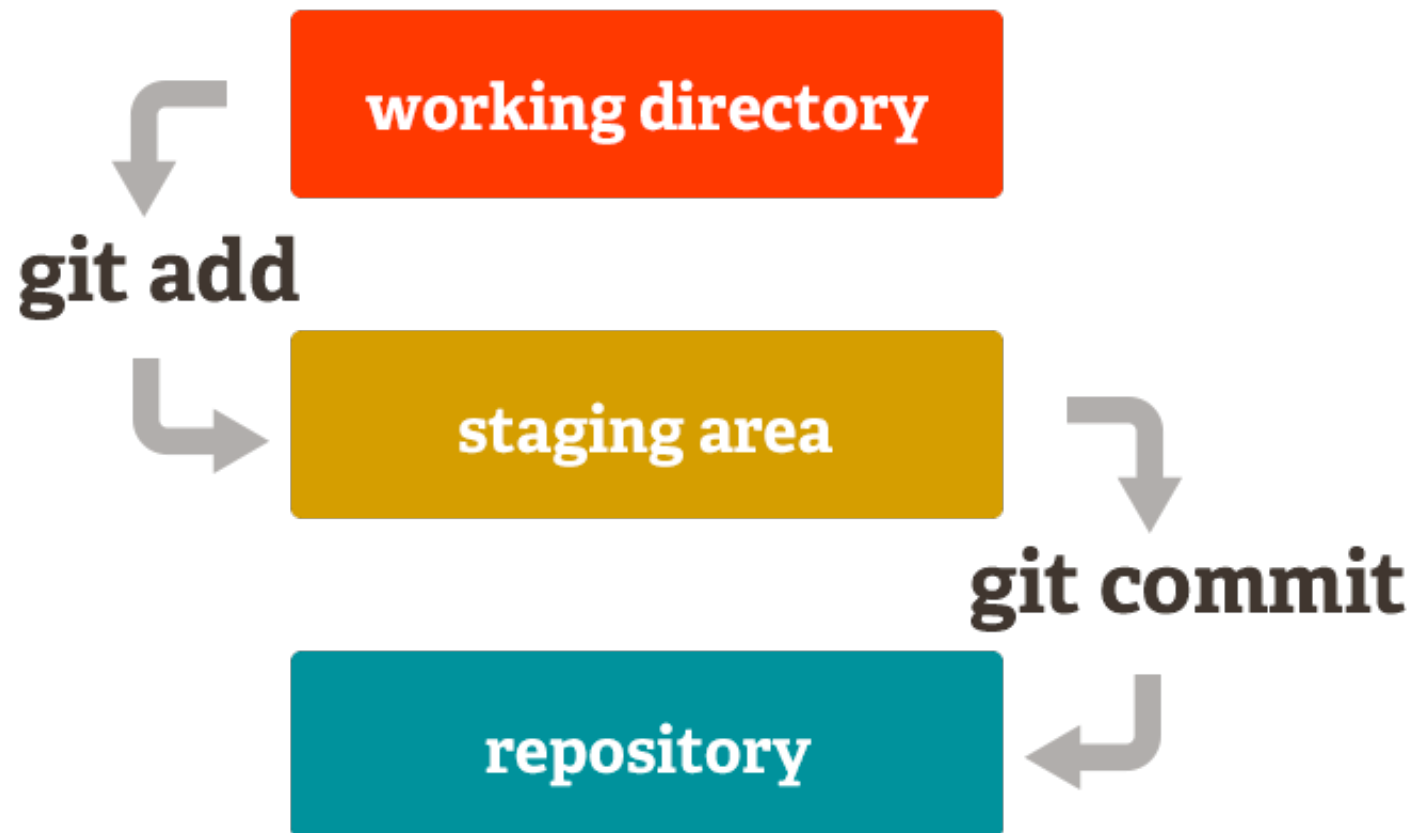
Git. Comandos básicos

En el mismo curso del aula virtual se encuentra un PDF con los comandos de git más utilizados y la explicación de cada uno de ellos.

Common Git Commands



- `$git config`
- `$git init`
- `$git clone <path>`
- `$git add <file_name>`
- `$git commit`
- `$git status`
- `$git remote`
- `$git checkout <branch_name>`
- `$git branch`
- `$git push`
- `$git pull`
- `$git merge <branch_name>`
- `$git diff`
- `$git reset`
- `$git revert`
- `$git tag`
- `$git log`



Git. Ramificaciones

- Las ramificaciones permiten independizar flujos de trabajo.
- A partir de un punto común (master o develop), se pueden lanzar distintas ramas que permitirán trabajar en diferentes features sin interferir en el flujo principal.
- Una vez se termina la nueva feature, se puede reintegrar en la rama principal.
- Las ramas que se reintegran en el flujo principal se pueden eliminar, puesto que su contenido queda en el hilo principal del desarrollo.

- Rama puntual: es aquella de corta duración que se abre para un tema o para una funcionalidad muy concreta.
- Ramas de largo recorrido: varias ramas siempre abiertas, e ir las usando en diferentes etapas del ciclo de desarrollo; realizando frecuentes fusiones entre ellas.
 - La idea es mantener en la rama master únicamente el código totalmente estable.
 - Y disponer de otras ramas paralelas siendo la principal la denominada desarrollo (develop), en las que trabajar y realizar pruebas.

Git. Ramificaciones

- Creación de una nueva rama: `git branch "nombre rama"`.
o `git checkout -b "nombre rama"`
- Movimiento entre ramas (pasar de una a otra): `git checkout "nombre rama"`.
- Eliminar una rama: `git branch -d "nombre rama"`.
- Reintegrar una rama en otra: `git merge "nombre rama"`.
 - (Este comando se debe ejecutar estando en en la rama destino)

Git. Gitflow

- Gitflow es una metodología para trabajar con Git
- Fue diseñado por Vincent Driessen en 2010.
- Permite trabajar con Git de forma ordenada y escalable y asegura que grandes equipos puedan trabajar en paralelo.

<https://nvie.com/posts/a-successful-git-branching-model/>

<https://datasift.github.io/gitflow/IntroducingGitFlow.html>

<https://danielkummer.github.io/git-flow-cheatsheet/>

Gitflow. Definición

- Podemos definirlo como una serie de “reglas” para facilitar el trabajo en equipo.
- El trabajo se organiza en dos ramas principales:
 - Master: todos los commits a Master deben estar preparados para subir a producción.
 - Develop: contendrá la siguiente versión planificada del proyecto.
- Cada vez que se incorpora código a la rama Master, se genera una nueva versión.

Gitflow: Ramas auxiliares

- Se proponen las siguientes ramas auxiliares:
 - Feature:
 - Se originan a partir de la rama develop.
 - Se incorporan siempre a la rama develop.
 - Nombre: cualquiera que no sea master, develop, hotfix-* o release-*
 - Release:
 - Se originan a partir de la rama develop.
 - Se incorporan a master y develop.
 - Nombre: release-*

Gitflow: Ramas auxiliares

Hotfix:

- Se origina a partir de la rama master.
- Se incorporan a Master y develop.
- Nombre: hotfix-*.

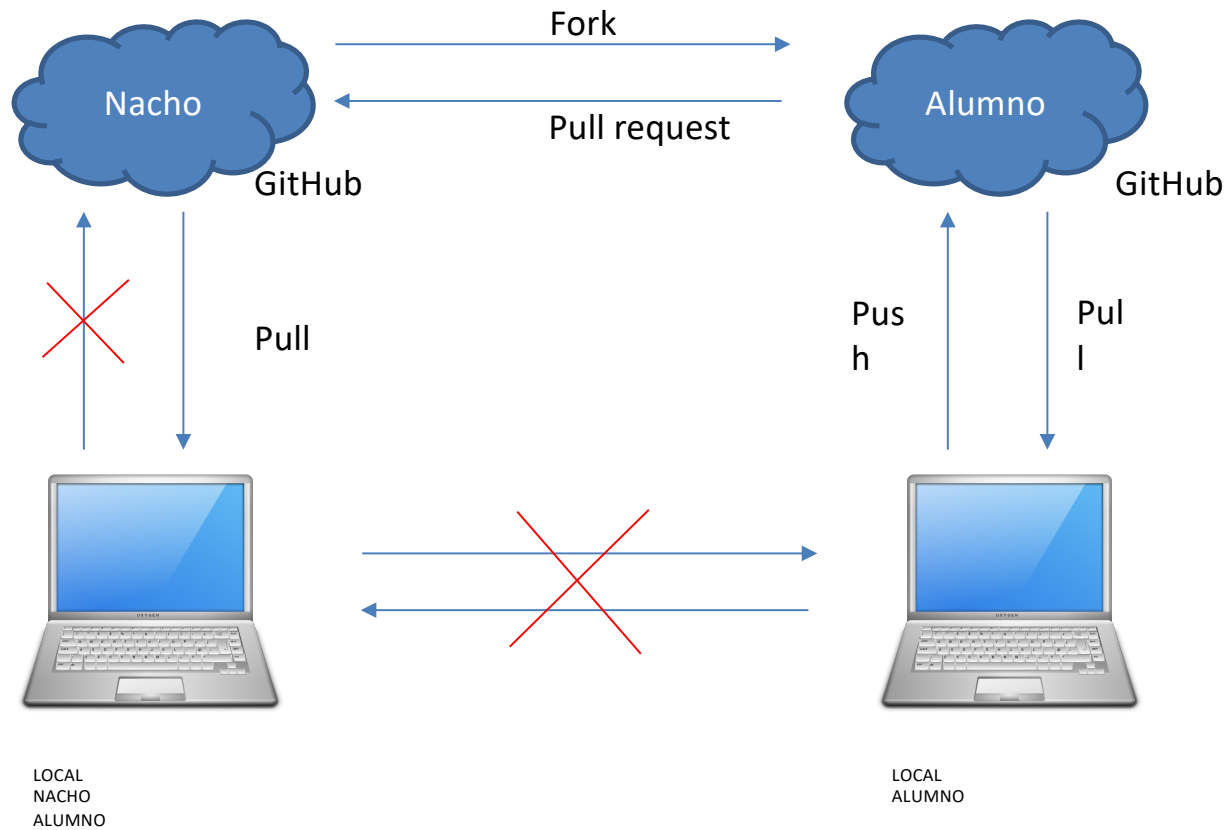
Etiquetas (Tags)

- Git tiene la posibilidad de etiquetar puntos específicos del historial como importantes.
- Esta funcionalidad se usa típicamente para marcar versiones de lanzamiento
 - Creación de la versión 1.0 de una aplicación.
- Hay dos tipos de etiquetas:
 - Ligeras
 - Anotadas

Etiquetas (Tags)

- Etiquetas ligeras:
 - Puntero a un commit específico.
- Etiquetas anotadas:
 - Se guardan en la base de datos de Git como objetos enteros.
 - Tienen un *checksum*.
 - Contienen el nombre del etiquetador, correo electrónico y fecha.
 - Tienen un mensaje asociado
 - Pueden ser firmadas y verificadas con *GNU Privacy Guard* (GPG).

Git Fork



Git fetch
equivale a:

- Git pull
- Git merge

Pull/Merge Request

- Las pull requests o merge requests son un mecanismo para que los desarrolladores notifiquen a los miembros de su equipo que han terminado una funcionalidad.
- Una vez la rama con la funcionalidad está lista, el desarrollador realiza la pull request mediante su cuenta en el servidor (Github, Bitbucket, Gitlab, etc.).
- Todas las personas involucradas saben que deben revisar el código, aunque puede asignarse a una persona en concreto la revisión de la petición.

Pull/Merge Request

- Ofrece un foro/comentarios para discutir sobre los cambios realizados.
- Si se presentan problemas con los cambios, se puede añadir información en el pull request e incluso añadir nuevos commits.
- Toda esta actividad queda registrada en la petición actual.

