Taller de GIT 2019

Pablo Martínez López

GUI
Grupo Universitario de Informática
Escuela de Ingeniería Informática, Universidad de Valladolid

Taller de GIT, 2019/20
Dia, XX de Noviembre de 2019

¿Qué es Git?

- Git es una herramienta de control de versiones.
- Nos permite un seguimiento del trabajo y sus versiones.
- Combina trabajo local y trabajo remoto.
- Utilizado generalmente para trabajo en equipo, aunque puede ser utilizado de forma individual.

¿Qué NO es Git?

- Es común confundir Git con GitHub, GitLab...etc
- Git es el software que nos permite trabajar con los remotos.
- GitHub u otras herramientas similares son interfaces web que permiten visualizar los repositorios remotos.

Iniciando el repositorio

Vamos a crearnos un repositorio local y uno remoto.
Para el local simplemente crearemos un nuevo directorio.
Para el remoto utilizaremos **GitHub**.

- **git init**
- **git clone**
- **git remote add origin{dirección ssh o https}**

Esquema general

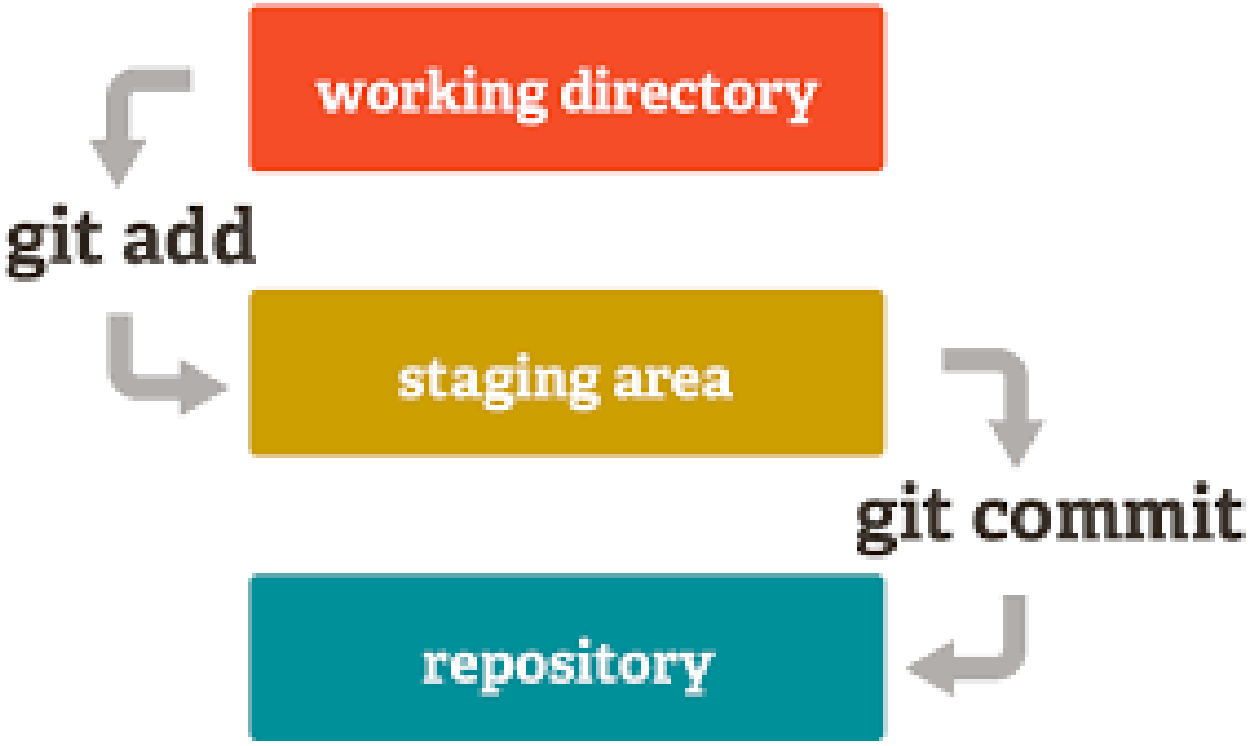


Figura: Esquema general

Add

- Con **git add** añadimos archivos a nuestro espacio de trabajo.
- Con **git add -remove** eliminamos archivos añadidos.
- Podemos utilizar espacios de nombres u opciones de comando: git add *.java ó git add –all.
- Ver: git - -help.

Commit

- Los commits añaden los archivos añadidos (add) al *stage area*.
- Llevan un mensaje que explican los cambios que se han hecho desde el último commit.
- Con **git commit -m "mensaje"** introducimos un pequeño mensaje.
- Para cambiar el editor predefinido: **git config –global core.editor "vim"**

Status y log

- Con **git status** podemos ver el estado del repositorio actual.
- Nos proporciona información sobre la rama, el área de trabajo, el *stage area*...etc.
- Con **git log** podemos ver el historial de *commits* en la rama actual.
- En breves veremos qué son las **ramas**.

Revert y Reset

- Con **git revert** realizamos un *commit* que revierte el anterior *commit* realizado.
- Con **git reset** reseteamos o deshacemos el punto donde nos encontremos actualmente. Esto lo hace de varias formas dependiendo el argumento del comando. Hay que tener cuidado.
  - git reset –**soft** deshace el último commit.
  - git reset –**mixed** (por defecto) deshace el último commit **y archivos añadidos**.
  - git reset –**hard** deshace commit y add, y **MODIFICA EL ESPACIO DE TRABAJO**.
  - Si fuese su primer commit, **BORRARÍA EL ARCHIVO**.

Push

- Con **git push**, "empujamos" los archivos del *stage area* del repositorio local al remoto.
- La sintaxis del comando es: git push 'origin' <*rama*>.
- Un push puede llevar varios commits.

Pull y Fetch

- Con **git pull**, "nos traemos" los cambios del servidor remoto a nuestro espacio de trabajo local.
- La sintaxis del comando es: git pull origin <*rama*>.
- **Git fetch** es una operación intermedia incluida en el pull que sirve para traer los cambios del remoto al *staging area*.

¿Qué son las ramas?

- Las ramas son **BIFURCACIONES** de nuestro proyecto.
- Siguen un esquema de árbol. No son líneas paralelas, contienen el código de la rama padre.
- Nos sirven para dividir el trabajo, tener una visualización clara del proyecto y para trabajar *de forma segura*.
- La rama principal se denomina **master**.

Branch y checkout

- Son las operaciones principales de las ramas.
- Con **git branch** creamos una rama (seguido del nombre de la rama) o vemos en que rama estamos (sin argumentos).
- Con **git checkout** nos movemos de rama. La opción **git checkout -b 'rama'** nos permite crear una rama e instantáneamente movernos a ella.

Merge teórico

- **¡CUESTIÓN MÁS DELICADA DE GIT!**
- Consiste en juntar una rama bifurcada con un padre.
- ¿Problema? $\rightarrow$ Un padre puede tener muchos hijos (ramas) y al volver a juntarse cada uno puede haber tratado las cosas de forma distinta.
- Se genera lo que se denomina **conflicto**.
- A pesar de que vamos a explicarlo por comandos, las interfaces web como **GitHub** o **GitLab** permiten hacerlo también de forma más visual.

Merge práctico

- Nos situamos mentalmente en la rama B que queremos mergear.
- Nos desplazamos a la rama a la que queremos fusionar la rama actual con **git checkout A**.
- Fusionamos la rama secundaria a la principal con **git merge B**.

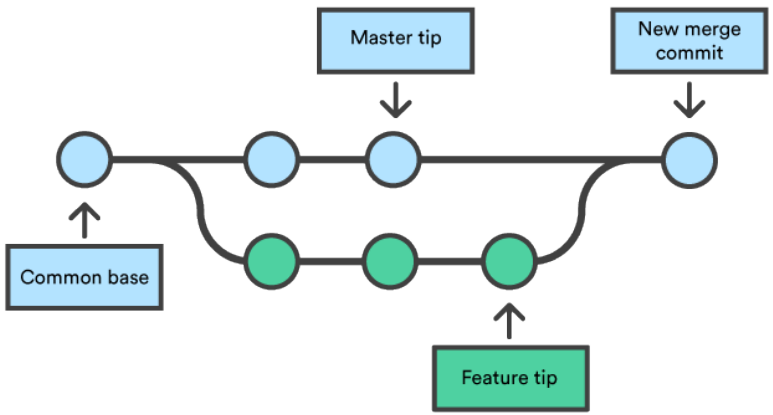


Figura: Ejemplo de ramas

Resolviendo los conflictos - I

- Vamos a crear un archivo txt de ejemplo en master y a bifurcar una rama tras ello.
- Escribimos, añadimos y hacemos commit en cada rama.
- Al hacer merge desde master...**¡CONFLICTO!**
- Editamos el archivo eligendo qué queremos conservar.
- Añadimos el archivo y realizamos un commit. El merge ha sido completado.

Resolviendo los conflictos - II

- Git es una herramienta esencial en proyectos a una gran escala y solucionar conflictos muy grandes puede ser muy muy costoso.
- Para evitar esto, hay que tener constancia de en qué trabaja cada uno y cómo lo hace. Es decir, ORGANIZACIÓN.
- Algunas interfaces web permiten solucionar conflictos de forma gráfica.

Figura: ¡CUIDADO!

Metodología básica

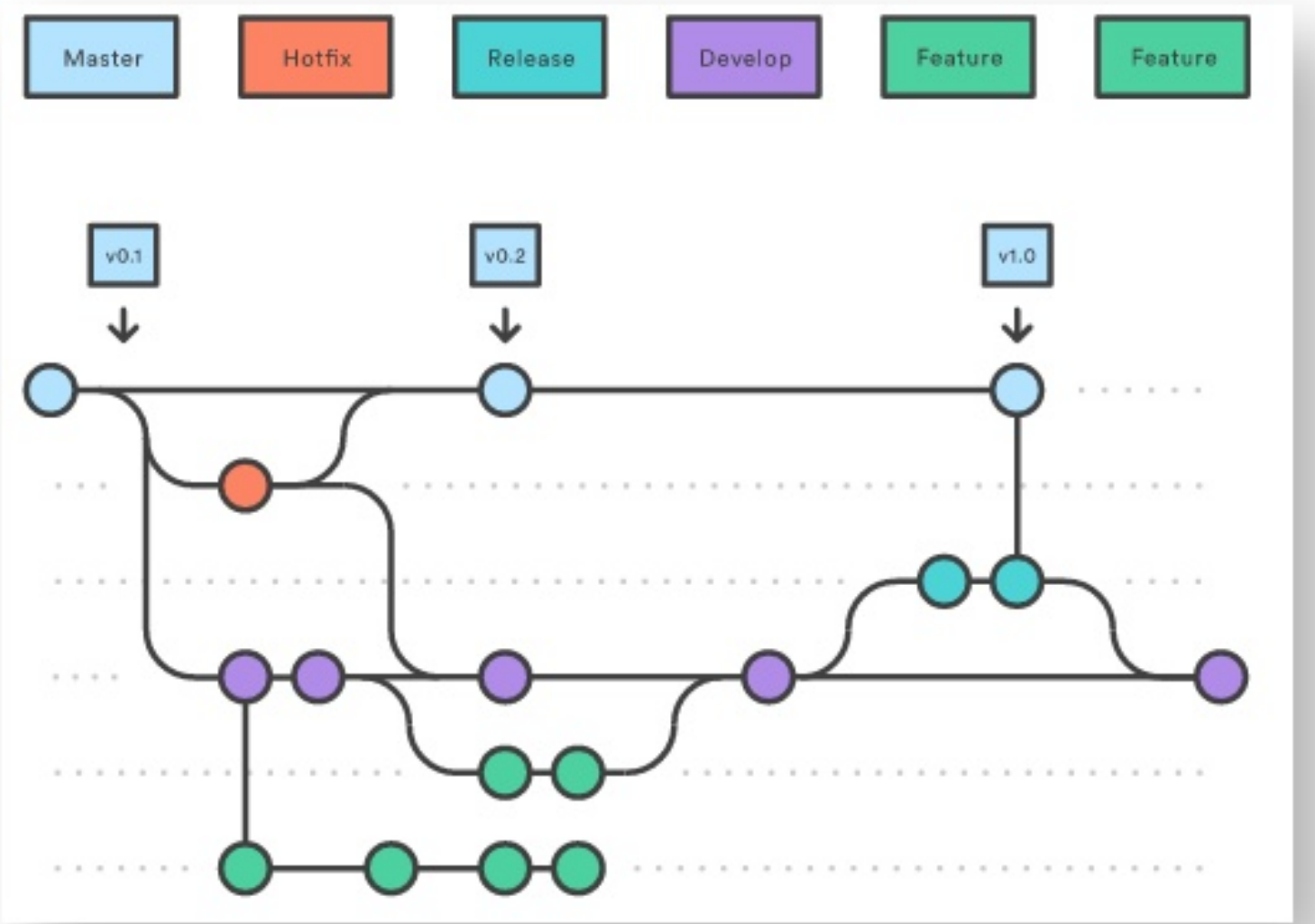
De nuevo recalcamos que una de las características que hacen a Git tan potente es su facilidad para organizar el trabajo. Pero tenemos que poner de nuestra parte.
Vamos a diferenciar varias ramas *"clásicas"* que nos permiten trabajar de forma eficiente:

- **master** - Rama principal donde está todo en orden.
- **develop** - Rama principal del DESARROLLO.
- **rama por miembro** - Subdivisiones de develop.

Otras buenas prácticas

- Crear una rama por característica en desarrollo a partir de develop o sub-ramas de esta. Se denominan **feature branches** y los **issues** son una buena forma de realizar seguimiento de ellas.
- Utilizar **master** sólo de "ventana al público". Pasaremos cada cambio a develop pero únicamente cambios significantes a master. Las ramas para realizar los últimos ajustes antes de mergear a master se denominan **release branches**. (- -)
- A pesar de que nunca debemos trabajar directamente sobre **master**, sí es común en etapas finales trabajar sobre esta o crear pequeñas ramas de master (**hotfix branches)** para tratar pequeños bugs.

Ejemplo visual



¡Gracias por asistir!

Esperamos que el taller haya sido ameno y que hayáis asentado los conceptos. Sin embargo, para aprender de verdad Git, **hay que utilizar Git**. Intentad utilizarlo día a día y evitad volver a llenar vuestro ordenador de versiones enviadas por Telegram.

Las diapositivas, como complemento teórico las encontraréis aquí: **https://github.com/HylianPablo/TallerGit2019**

- Para cualquier duda podéis pasaros por la sede del GUI o preguntar por Telegram: **@HylianPablo**
- Si estáis interesados en más eventos y talleres podéis seguirnos en nuestras redes sociales: Twitter: **@GUI_UVa** e Instagram: **@gui_uva**