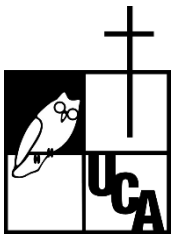
Universidad Centroamericana “José Simeón Cañas”
Facultad de Ingeniería y Arquitectura

Tutorías - Fundamentos de Programación Ciclo 01/2021
Tutor: Ernesto José Canales Guillén



Git y GitHub

¿Qué es GitHub?

GitHub es una plataforma de desarrollo colaborativo de software para alojar proyectos utilizando el sistema de control de versiones Git.

¿Para qué sirve?

GitHub aloja tu repositorio de código y te brinda herramientas muy útiles para el trabajo en equipo, dentro de un proyecto. Además de eso, puedes contribuir a mejorar el software de los demás. Para poder alcanzar esta meta, GitHub provee de funcionalidades para hacer un fork y solicitar pulls.

Realizar un fork es simplemente clonar un repositorio ajeno (genera una copia en tu cuenta), para eliminar algún bug o modificar cosas de él. Una vez realizadas tus modificaciones puedes enviar un pull al dueño del proyecto. Éste podrá analizar los cambios que has realizado fácilmente, y si considera interesante tu contribución, adjuntarlo con el repositorio original.

¿Qué herramientas proporciona?

En la actualidad, GitHub es mucho más que un servicio de alojamiento de código. Además de éste, se ofrecen varias herramientas útiles para el trabajo en equipo. Entre ellas, caben destacar:

- Una wiki para el mantenimiento de las distintas versiones de las páginas.
- Un sistema de seguimiento de problemas que permiten a los miembros de tu equipo detallar un problema con tu software o una sugerencia que deseen hacer.
- Una herramienta de revisión de código, donde se pueden añadir anotaciones en cualquier punto de un fichero y debatir sobre determinados cambios realizados en un commit específico.
- Un visor de ramas donde se pueden comparar los progresos realizados en las distintas ramas de nuestro repositorio.

¿Qué es Git?

Git es un sistema de control de versiones. Un sistema de control de versiones nos va a servir para trabajar en equipo de una manera mucho más simple y optima cuando estamos desarrollando software. Con Git vamos a poder controlar todos los cambios que se hacen en nuestra aplicación y en nuestro código y vamos a tener control absoluto de todo lo que pasa en el código, pudiendo volver atrás en el tiempo, pudiendo abrir diferentes ramas de desarrollo, etc.

¿Qué son las ramas en Git?

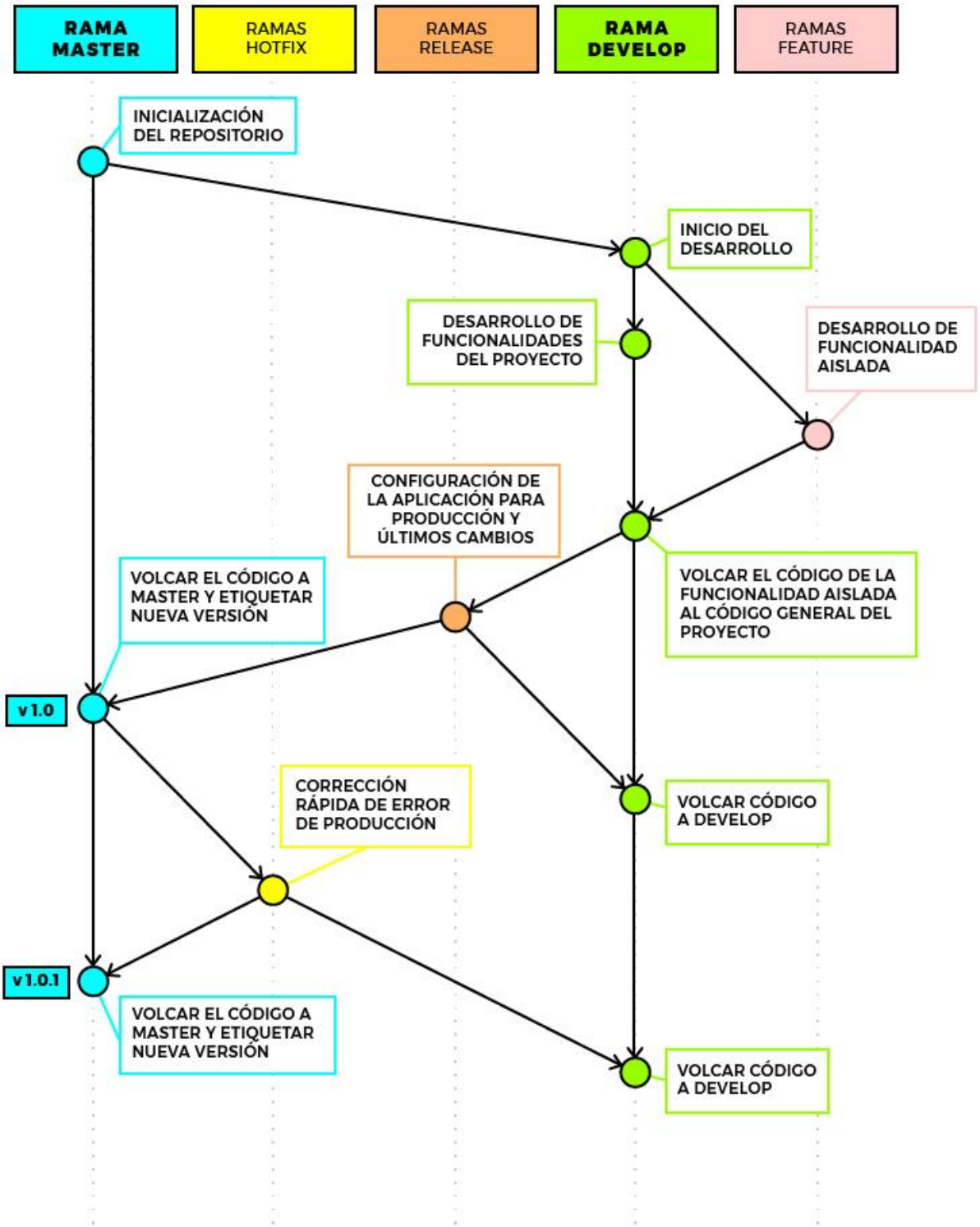
El concepto principal de un repositorio GIT es la rama. Una rama representa la evolución del código de un proyecto y dentro de cada rama habrá varios puntos o commits en los que se haya almacenado información en el servidor.

Un repositorio GIT se compone de un conjunto de ramas que se relacionan entre si para componer el código del proyecto. Los puntos en los que las ramas convergen representan aquellos momentos en los que se vuelca el contenido de una rama en otra, para traspasar las funcionalidades o mejoras realizadas.

En base a los tipos de proyectos que se realizan dentro de la organización y la estructura del equipo de desarrollo presente es necesario escoger cómo se van a manejar las ramas, lo que se conoce como el modelo de ramas del repositorio.

Nombre de ramas comúnmente usadas:

- Rama master: la rama principal del proyecto. Sobre esta rama no se hace trabajo, solamente se utiliza como copia de las versiones del software que se despliegan. Esta rama estará configurada con los valores de configuración para el entorno del servidor de producción.
- Rama develop: la rama de trabajo diario del proyecto. La mayor parte de los cambios que se realizan sobre el código del proyecto se realizan en esta rama. Esta rama estará configurada con los valores de configuración para los entornos de desarrollo.
- Ramas feature: cuando se necesita desarrollar una funcionalidad aislada en un proyecto se abre una rama de este tipo y se realiza el desarrollo completo de dicha funcionalidad en esta rama. Una vez completado el desarrollo, se vuelca la rama dentro de la rama develop para incluir la nueva funcionalidad en el código del proyecto.
- Ramas release: cuando se va a realizar un despliegue de una versión del código en el servidor de producción se abre una rama de este tipo. Una vez abierta se hacen los cambios de configuración relativos al entorno de producción y se sube el código del proyecto al servidor. Cuando el despliegue se completa de forma satisfactoria, la rama release se vuelca a la rama master y se añade una etiqueta con el número de versión correspondiente.
- Rama hotfix: cuando se quiere solventar un error pequeño en el código que está desplegado en el servidor, se abre una rama de este tipo desde la rama master y se realizan los cambios pertinentes en ella. Una vez completados los cambios se vuelcan tanto en la rama develop como en la rama master, que se etiqueta con la nueva versión. En ese momento, el arreglo estará disponible en las dos ramas fundamentales del repositorio del proyecto.


RAMA MASTER
RAMAS HOTFIX
RAMAS RELEASE
RAMA DEVELOP
RAMAS FEATURE
INICIALIZACIÓN DEL REPOSITORIO
INICIO DEL DESARROLLO
DESARROLLO DE FUNCIONALIDADES DEL PROYECTO
DESARROLLO DE FUNCIONALIDAD AISLADA
CONFIGURACIÓN DE LA APLICACIÓN PARA PRODUCCIÓN Y ÚLTIMOS CAMBIOS
VOLCAR EL CÓDIGO A MASTER Y ETIQUETAR NUEVA VERSIÓN
VOLCAR EL CÓDIGO DE LA FUNCIONALIDAD AISLADA AL CÓDIGO GENERAL DEL PROYECTO
v 1.0
CORRECCIÓN RÁPIDA DE ERROR DE PRODUCCIÓN
VOLCAR CÓDIGO A DEVELOP
v 1.0.1
VOLCAR EL CÓDIGO A MASTER Y ETIQUETAR NUEVA VERSIÓN
VOLCAR CÓDIGO A DEVELOP

Comándanos básicos de Git:

Git Cheat Sheet

REBELLABS by ZEROTURNAROUND

Create a Repository

From scratch -- Create a new local repository
```
$ git init [project name]
```

Download from an existing repository
```
$ git clone my_url
```

Observe your Repository

List new or modified files not yet committed
```
$ git status
```

Show the changes to files not yet staged
```
$ git diff
```

Show the changes to staged files
```
$ git diff --cached
```

Show all staged and unstaged file changes
```
$ git diff HEAD
```

Show the changes between two commit ids
```
$ git diff commit1 commit2
```

List the change dates and authors for a file
```
$ git blame [file]
```

Show the file changes for a commit id and/or file
```
$ git show [commit]:[file]
```

Show full change history
```
$ git log
```

Show change history for file/directory including diffs
```
$ git log -p [file/directory]
```

Working with Branches

List all local branches
```
$ git branch
```

List all branches, local and remote
```
$ git branch -av
```

Switch to a branch, my_branch, and update working directory
```
$ git checkout my_branch
```

Create a new branch called new_branch
```
$ git branch new_branch
```

Delete the branch called my_branch
```
$ git branch -d my_branch
```

Merge branch_a into branch_b
```
$ git checkout branch_b
$ git merge branch_a
```

Tag the current commit
```
$ git tag my_tag
```

Make a change

Stages the file, ready for commit
```
$ git add [file]
```

Stage all changed files, ready for commit
```
$ git add .
```

Commit all staged files to versioned history
```
$ git commit -m "commit message"
```

Commit all your tracked files to versioned history
```
$git commit -am "commit message"
```

Unstages file, keeping the file changes
```
$ git reset [file]
```

Revert everything to the last commit
```
$ git reset --hard
```

Synchronize

Get the latest changes from origin (no merge)
```
$ git fetch
```

Fetch the latest changes from origin and merge
```
$ git pull
```

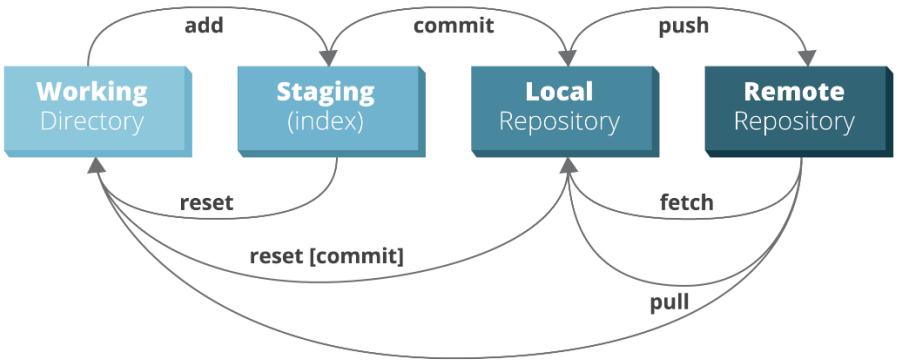
Fetch the latest changes from origin and rebase
```
$ git pull --rebase
```

Push local changes to the origin
```
$ git push
```

Finally!

When in doubt, use git help
```
$ git command --help
```

Or visit https://training.github.com/ for official GitHub training.



Posted in r/**git** by u/**spite77**

reddit