Workshop de Git

Ing. Rodrigo García

Despegar.com

August 3, 2017

Contenido

¿Qué es Git?

Definición

Git es un sistema de control de versión libre, de código abierto y **distribuido**, diseñado para manejar desde proyectos chicos a muy grandes en forma veloz y eficiente. [1]

Comentario

Así como se denomina distribuido, en realidad nosotros lo usamos en forma centralizada. Cuando mandamos nuestros cambios, lo hacemos a un repositorio remoto. De todos modos, cada uno tiene en su máquina un repositorio con toda la información histórica de las ramas en las que trabajamos, aún cuando no tenemos conexión al servidor, y se puede configurar para usarlo en forma distribuida.

[1] http://git-scm.com/

¿Qué es Git?

Definición

Git es un sistema de control de versión libre, de código abierto y **distribuido**, diseñado para manejar desde proyectos chicos a muy grandes en forma veloz y eficiente. [1]

Comentario

Así como se denomina distribuido, en realidad nosotros lo usamos en forma centralizada. Cuando mandamos nuestros cambios, lo hacemos a un repositorio remoto. De todos modos, cada uno tiene en su máquina un repositorio con toda la información histórica de las ramas en las que trabajamos, aún cuando no tenemos conexión al servidor, y se puede configurar para usarlo en forma distribuida.

[1] `http://git-scm.com/`

Instalando...

- Ubuntu (Linux): $ sudo apt install git
- Windows:
  - Página oficial: `http://git-scm.com/download/win`
  - Link alternativo: `http://code.google.com/p/msysgit/downloads/list?can=3`
  - Tortoise Git: `http://code.google.com/p/tortoisegit/`
  - **Importante:** al instalar, poner la opción putty y no openSSH.
- Mac: `http://git-scm.com/download/mac`
- GUI para Eclipse: EGit (buscarlo en el marketplace)

Sea cual sea la versión, siempre disponemos de un shell donde ejecutar comandos. En linux y mac es la consola unix, y en windows nos instala el Git Shell.

Instalando...

- Ubuntu (Linux): $ sudo apt install git
- Windows:
  - Página oficial: `http://git-scm.com/download/win`
  - Link alternativo: `http://code.google.com/p/msysgit/downloads/list?can=3`
  - Tortoise Git: `http://code.google.com/p/tortoisegit/`
  - **Importante:** al instalar, poner la opción putty y no openSSH.
- Mac: `http://git-scm.com/download/mac`
- GUI para Eclipse: EGit (buscarlo en el marketplace)

Sea cual sea la versión, siempre disponemos de un shell donde ejecutar comandos. En linux y mac es la consola unix, y en windows nos instala el Git Shell.

Instalando...

- Ubuntu (Linux): $ sudo apt install git
- Windows:
  - Página oficial: `http://git-scm.com/download/win`
  - Link alternativo: `http://code.google.com/p/msysgit/downloads/list?can=3`
  - Tortoise Git: `http://code.google.com/p/tortoisegit/`
  - **Importante:** al instalar, poner la opción putty y no openSSH.
- Mac: `http://git-scm.com/download/mac`
- GUI para Eclipse: EGit (buscarlo en el marketplace)

Sea cual sea la versión, siempre disponemos de un shell donde ejecutar comandos. En linux y mac es la consola unix, y en windows nos instala el Git Shell.

Instalando...

- Ubuntu (Linux): $ sudo apt install git
- Windows:
  - Página oficial: `http://git-scm.com/download/win`
  - Link alternativo: `http://code.google.com/p/msysgit/downloads/list?can=3`
  - Tortoise Git: `http://code.google.com/p/tortoisegit/`
  - **Importante:** al instalar, poner la opción putty y no openSSH.
- Mac: `http://git-scm.com/download/mac`
- GUI para Eclipse: EGit (buscarlo en el marketplace)

Sea cual sea la versión, siempre disponemos de un shell donde ejecutar comandos. En linux y mac es la consola unix, y en windows nos instala el Git Shell.

Instalando... (2)

Ayuda!

```
$ git --help
$ git help <comando>
```

En SS.OO. Unix (manual más completo que el help)

```
$ man git
```

En general (man en la web)

```
http://git-scm.com/docs
```

Instalando... (2)

Ayuda!

```
$ git --help
$ git help <comando>
```

En SS.OO. Unix (manual más completo que el help)

```
$ man git
```

En general (man en la web)

http://git-scm.com/docs

Instalando... (2)

Ayuda!

```
$ git --help
$ git help <comando>
```

En SS.OO. Unix (manual más completo que el help)

```
$ man git
```

En general (man en la web)

```
http://git-scm.com/docs
```

Configurando...

Ejecutamos lo siguiente en el shell de git para configurar nuestro usuario [2]:

Comandos

```
$ git config --global user.name "Juan Perez"
$ git config --global user.email "jperez@despegar.com"
```

Esta información va a servir para indicarle al repositorio quiénes somos y cómo mostrar nuestros commits.

[2] Para más información: `http://wiki.despegar.it/index.php/Git`

Configurando... (2)

Generación de claves (Unix)

- $ mkdir -p ~/.ssh/
- $ cd ~/.ssh/
- $ ssh-keygen -C jperez@despegar.com -f id_desp_rsa
- Ir a `https://github.com/settings/keys` y agregar la clave generada en id_desp_rsa.pub

Configurando... (2)

Generación de claves (Unix)

- $ mkdir -p ~/.ssh/
- $ cd ~/.ssh/
- $ ssh-keygen -C jperez@despegar.com -f id_desp_rsa
- Ir a `https://github.com/settings/keys` y agregar la clave generada en id_desp_rsa.pub

Configurando... (2)

Generación de claves (Unix)

- $ mkdir -p ~/.ssh/
- $ cd ~/.ssh/
- $ ssh-keygen -C jperez@despegar.com -f id_desp_rsa
- Ir a `https://github.com/settings/keys` y agregar la clave generada en id_desp_rsa.pub

Configurando... (2)

Generación de claves (Unix)

- $ mkdir -p ~/.ssh/
- $ cd ~/.ssh/
- $ ssh-keygen -C jperez@despegar.com -f id_desp_rsa
- Ir a `https://github.com/settings/keys` y agregar la clave generada en id_desp_rsa.pub

Configurando... (3)

Generación de claves (Windows)

1. Usando putty key generator generar una clave. Bajar el binario e instalarlo.
2. Reemplazar en Key comment el tu_usuario@despegar.com.
3. Guardar las constraseñas generadas. (tanto la pública como la privada)
4. Copiar la clave que se generó (Public key for pasting into OpenSHH...).
5. Ir a `https://github.com/settings/keys` y pegar la clave generada con putty key generator.

Configurando... (3)

Generación de claves (Windows)

1. Usando putty key generator generar una clave. Bajar el binario e instalarlo.
2. Reemplazar en Key comment el tu_usuario@despegar.com.
3. Guardar las constraseñas generadas. (tanto la pública como la privada)
4. Copiar la clave que se generó (Public key for pasting into OpenSHH...).
5. Ir a `https://github.com/settings/keys` y pegar la clave generada con putty key generator.

Configurando... (3)

Generación de claves (Windows)

1. Usando putty key generator generar una clave. Bajar el binario e instalarlo.
2. Reemplazar en Key comment el tu_usuario@despegar.com.
3. Guardar las constraseñas generadas. (tanto la pública como la privada)
4. Copiar la clave que se generó (Public key for pasting into OpenSHH...).
5. Ir a `https://github.com/settings/keys` y pegar la clave generada con putty key generator.

Configurando... (3)

Generación de claves (Windows)

1. Usando putty key generator generar una clave. Bajar el binario e instalarlo.
2. Reemplazar en Key comment el tu_usuario@despegar.com.
3. Guardar las constraseñas generadas. (tanto la pública como la privada)
4. Copiar la clave que se generó (Public key for pasting into OpenSHH...).
5. Ir a `https://github.com/settings/keys` y pegar la clave generada con putty key generator.

Configurando... (3)

Generación de claves (Windows)

1. Usando putty key generator generar una clave. Bajar el binario e instalarlo.
2. Reemplazar en Key comment el tu_usuario@despegar.com.
3. Guardar las constraseñas generadas. (tanto la pública como la privada)
4. Copiar la clave que se generó (Public key for pasting into OpenSHH...).
5. Ir a `https://github.com/settings/keys` y pegar la clave generada con putty key generator.

Flujo de Git

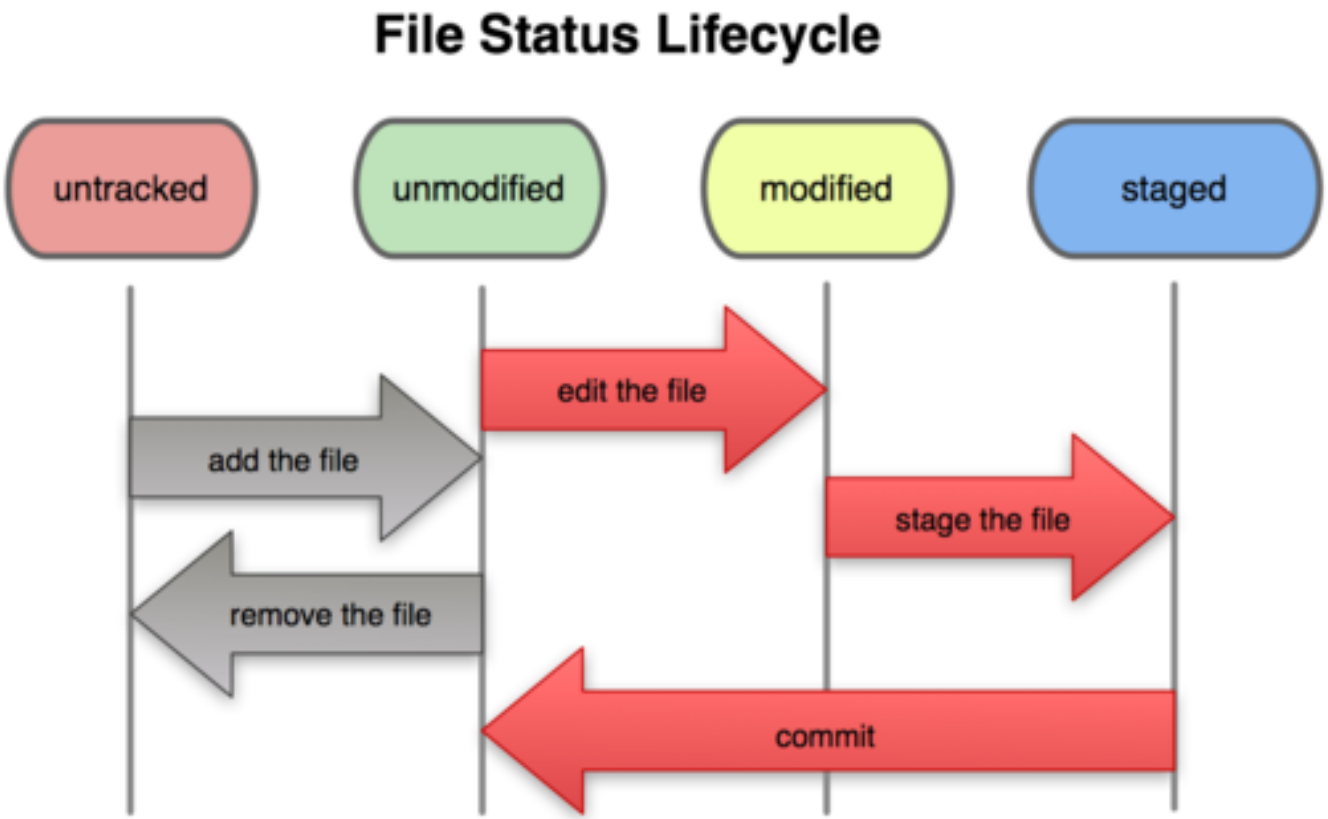


Figure: Ciclo de vida de un archivo

Flujo de Git (2)

Referencia

- Untracked: Archivos nuevos (similar a SVN).
- Unmodified: Archivos sin cambiar desde el estado HEAD de nuestro repositorio local.
- Modified: Archivos modificados, aún sin stage.
- Staged: Archivos listos para ser commiteados.

Flujo de Git (2)

Referencia

- Untracked: Archivos nuevos (similar a SVN).
- Unmodified: Archivos sin cambiar desde el estado HEAD de nuestro repositorio local.
- Modified: Archivos modificados, aún sin stage.
- Staged: Archivos listos para ser commiteados.

Flujo de Git (2)

Referencia

- Untracked: Archivos nuevos (similar a SVN).
- Unmodified: Archivos sin cambiar desde el estado HEAD de nuestro repositorio local.
- Modified: Archivos modificados, aún sin stage.
- Staged: Archivos listos para ser commiteados.

Flujo de Git (2)

Referencia

- Untracked: Archivos nuevos (similar a SVN).
- Unmodified: Archivos sin cambiar desde el estado HEAD de nuestro repositorio local.
- Modified: Archivos modificados, aún sin stage.
- Staged: Archivos listos para ser commiteados.

Non-fast-forward updates

¿Repositorio local?

Algo para resaltar es que, a diferencia de p. ej. SVN, lo que nosotros mandamos (**push**) al servidor no son sólo cambios, sino el estado completo del repositorio. Es por eso que si intentamos subir nuestros cambios mientras que otro modificó archivos, por más de que sean diferentes, el servidor lo va a rechazar. Nos avisa que estaríamos perdiendo cambios y nos obliga a hacer un **pull** para que los mezcle. Esto se conoce como non-fast-forward updates.

Comandos

Inicializar repositorio

```
$ echo "#test" >> README.md
$ git init
$ git add README.md
$ git commit -m "First commit"
$ git remote add \
 origin git@github.com:<user>/<my-repo>.git
$ git push -u origin master
```

Clonar repositorio remoto (configura automáticamente el orígen)

```
$ git clone git@github.com:<user>/<my-repo>.git
```

Comandos

Inicializar repositorio

```
$ echo "#test" >> README.md
$ git init
$ git add README.md
$ git commit -m "First commit"
$ git remote add \
 origin git@github.com:<user>/<my-repo>.git
$ git push -u origin master
```

Clonar repositorio remoto (configura automáticamente el orígen)

```
$ git clone git@github.com:<user>/<my-repo>.git
```

Comandos (2)

Crear ramas

```
$ git branch <RAMA> develop && git checkout <RAMA>
  # Crear la rama en el punto actual.
  # Es necesario hacer checkout a la misma.
$ git checkout -b <RAMA> develop
  # Crear rama en el punto actual y hacerle checkout.
$ echo "# <nombre>" >> <nombre>.md
  # Crear un nuevo archivo.
$ git add <nombre>.md
  # Agregamos al registro.
$ git status
  # Verificamos el estado del repo local.
```

Comandos (3)

Commit sobre la rama

```
$ git commit -m "Agregar buen mensaje"
  # Realizamos el commit de los cambios.
$ git log
  # Ver los cambios realizados.
```

Subiendo la rama local

```
$ git push origin <RAMA>
  # Subimos el estado de nuestro repo al repo remoto
$ git checkout -b merging && git merge <RAMA>
  # Vamos a la rama merging, luego hacemos un
  merge de la rama, Las diferencias se resuelven
  automáticamente si es posible. En caso de conflictos,
  el proceso se detiene (merging) es necesario manual.
```

Comandos (3)

Commit sobre la rama

```
$ git commit -m "Agregar buen mensaje"
  # Realizamos el commit de los cambios.
$ git log
  # Ver los cambios realizados.
```

Subiendo la rama local

```
$ git push origin <RAMA>
  # Subimos el estado de nuestro repo al repo remoto
$ git checkout -b merging && git merge <RAMA>
  # Vamos a la rama merging, luego hacemos un
  merge de la rama, Las diferencias se resuelven
  automáticamente si es posible. En caso de conflictos,
  el proceso se detiene (merging) es necesario manual.
```

Conflictos... conflictos everywhere

Resolver conflictos

```
$ git push origin merging
  # Intentamos subir todos los cambios...
$ git pull && git status
  # Traemos y vemos la situación actual del merge
$ git mergetool
  # Muestra los conflictos y las diferencias
  # Si es que configuramos un mergetool...
```

Merge tool

Configuración

```
$ vim ~/.gitconfig
[merge]
  tool = meldMerge
[mergetool "meldMerge"]
  cmd = meld --diff $LOCAL $MERGED $REMOTE
[diff]
  tool = meldDiff
[difftool "meldDiff"]
  cmd = meld --diff $LOCAL $REMOTE
```

Merge tool (2)

Ejecución

```
$ git difftool
$ git mergetool
```

Herramientas

Algunas opciones:

- Meld
- Beyond Compare

Mas completas:

- gitk
- GitKraken

Merge tool (2)

Ejecución

```
$ git difftool
$ git mergetool
```

Herramientas

Algunas opciones:

- Meld
- Beyond Compare

Mas completas:

- gitk
- GitKraken

Merge tool (2)

Ejecución

```
$ git difftool
$ git mergetool
```

Herramientas

Algunas opciones:

- Meld
- Beyond Compare

Mas completas:

- gitk
- GitKraken

Creando y obteniéndose ramas

Sujeto “A“ empieza una feature

Cuando creamos una rama, se desprende de aquella en la que estamos parados. ($ git status)

```
$ git checkout -b feature-nueva
$ git push origin feature-nueva
```

Sujeto ”B” quiere trabajar en la misma feature

```
$ git fetch  origin
$ git branch -a
# *master
# remotes/origin/HEAD -> origin/master
# remotes/origin/feature-nueva
$ git checkout feature-nueva
```

Creando y obteniéndose ramas

Sujeto “A“ empieza una feature

Cuando creamos una rama, se desprende de aquella en la que estamos parados. ($ git status)

```
$ git checkout -b feature-nueva
$ git push origin feature-nueva
```

Sujeto ”B” quiere trabajar en la misma feature

```
$ git fetch  origin
$ git branch -a
# *master
# remotes/origin/HEAD -> origin/master
# remotes/origin/feature-nueva
$ git checkout feature-nueva
```

Borrar una rama

En mi repo local

Si es una rama local o sólo queremos eliminarla de nuestro repo, este paso es único.

```
$ git branch -d <rama>
```

En origen remoto

Si además queremos borrarla del origen remoto

```
$ git push origin :<rama>
```

Borrar una rama

En mi repo local

Si es una rama local o sólo queremos eliminarla de nuestro repo, este paso es único.

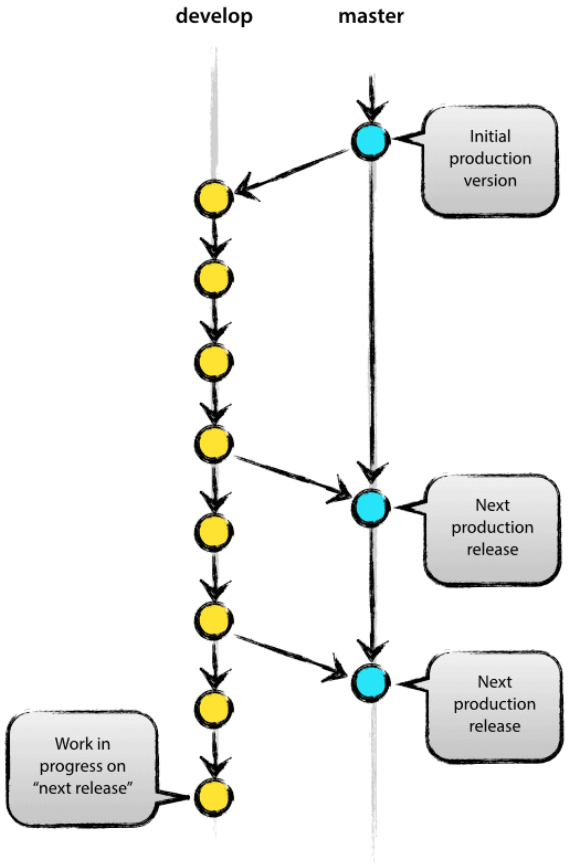
```
$ git branch -d <rama>
```

En origen remoto

Si además queremos borrarla del origen remoto

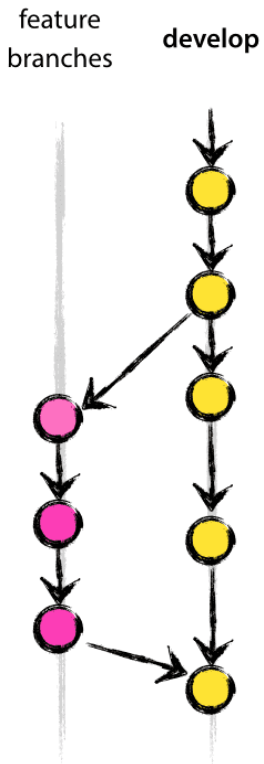
```
$ git push origin :<rama>
```

Main Branches



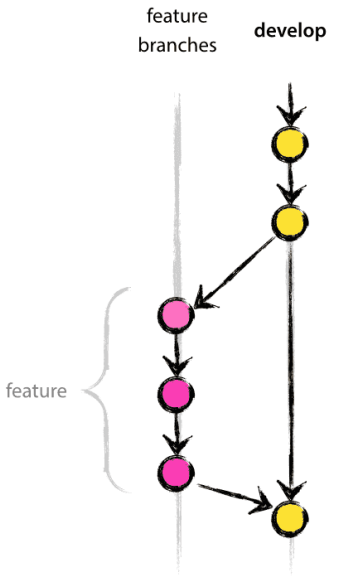
- Se conocen como nuestras ramas maestras.
- Master es siempre nuestra rama productiva.
- Develop es la rama de integración, donde podemos trabajar tranquilos. Sabemos que puede no estar estable.
- En un equipo grande quizás nos falte algo mas.

Support Branches - Featured branches

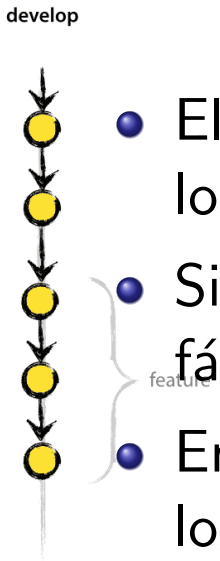


- Se conocen como ramas de soporte.
- Nos permite desarrollar nuevos cambios sin afectar la rama develop.
- La creamos: git checkout -b myfeature develop.
- Luego de terminar volvemos a develop: git checkout develop.
- Integramos la rama feature en develop: git merge –no-ff myfeature

Importancia de usar –no-ff



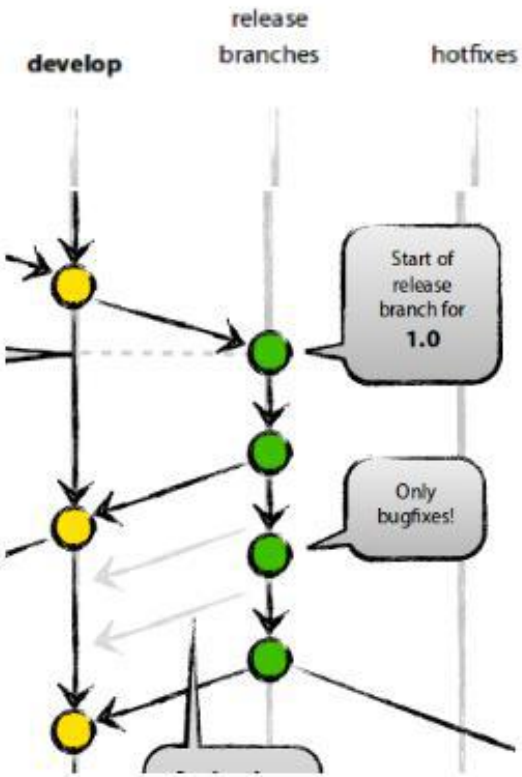
git merge --no-ff



git merge
(plain)

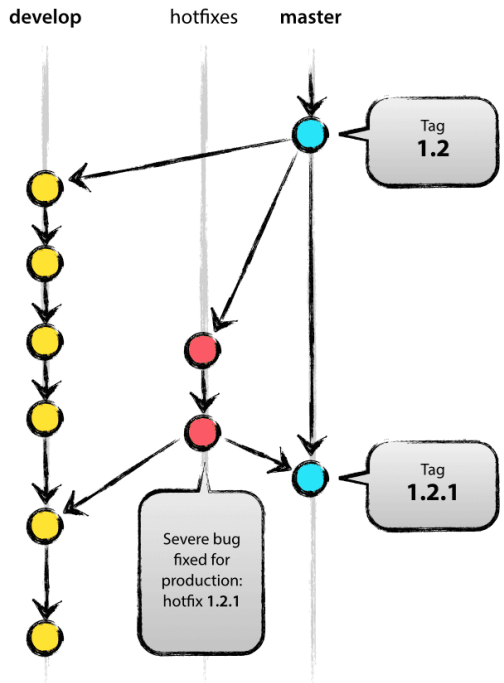
- El flag –no-ff, nos permite integrar los cambios en un nuevo commit.
- Sin el flag no podríamos identificar fácilmente los cambios del feature.
- En caso de necesitar volver atrás los cambios mergeados es mas fácil.

Support Branches - Release branches



- Release branches nos permiten preparar un despliegue en producción: git checkout -b release-1.0 develop
- Solo deberíamos realizara pequeños bugfix.
- Cada pequeño bugfix podemos integrarlo en develop.
- Utilizando github se realiza un pull request.
- Sin github es podria hacerse en merge en master (usar –no-ff)

Support Branches - Hotfix branches



- Murio prod!!! rajemo!!!.
- No, usamos una hotfix branch.
- La creamos: git checkout -b hotfix-1.2.1 master
- Arreglamos el problema podemos subir a master.
- git checkout master && git merge –no-ff hotfix-1.2.1
- No olvidarse de hacer el tag: git tag -a 1.2.1 -m "hotfix de prod"

Sino hay tabla

- Commit sin comentarios, hay tabla.
- Features en el master, hay tabla.
- Hotfix en el master, hay tabla.
- "git add ." con conflictos de merge, hay tabla.
- git merge sin usar flag –no-ff, hay tabla.
- merge directo sobre el master sin pull request, hay tabla.
- git reset sobre commit publico, hay tabla.
- commit –amed sobre commit publico, hay tabla.
- rebase sobre rama publica, hay tabla.

Tagging

Listar

```
$ git tag
```

Crear un tag

```
$ git tag -a v1.4 -m "creando el tag 1.4"
```

Eliminarlo

```
$ git push origin :refs/tags/v1.4
#Evitar eliminar un tag productivo.
```

[2]Para más información:
http://git-scm.com/book/en/Git-Basics-Tagging

Stash

Stash

¿Qué pasa si estoy por la mitad de un cambio y necesito moverme de branch para ver otra cosa? Si intentamos hacer checkout, nos va a pedir que commiteemos los cambios ya que de otra manera los vamos a perder. Otra opción es el comando **stash**. [3] Básicamente, la idea es que los cambios quedan almacenados y al hacer git status no los vemos. Una vez que volvamos a trabajar en ese branch, podemos recuperarlos con git stash apply.

[3] Para más información:
`http://git-scm.com/book/en/Git-Tools-Stashing`

Cherry - Cherry Pick

Cherry

En ciertas circunstancias podríamos llegar a necesitar aplicar cambios en ramas que divergieron. Es decir, si las mergeara, traería otros cambios que yo no quiero. El comando cherry nos permite ver, dado un branch, qué cambios se hicieron desde que las dos ramas se separaron. `http://git-scm.com/docs/git-cherry`.

Cherry Pick

Cherry pick nos permite aplicar esos cambios de uno o más commits específicos a un branch.
`http://git-scm.com/docs/git-cherry-pick`.

Cherry - Cherry Pick

Cherry

En ciertas circunstancias podríamos llegar a necesitar aplicar cambios en ramas que divergieron. Es decir, si las mergeara, traería otros cambios que yo no quiero. El comando cherry nos permite ver, dado un branch, qué cambios se hicieron desde que las dos ramas se separaron. `http://git-scm.com/docs/git-cherry`.

Cherry Pick

Cherry pick nos permite aplicar esos cambios de uno o más commits específicos a un branch.
`http://git-scm.com/docs/git-cherry-pick`.

Gitignore

.gitignore

Tenemos la posibilidad de decirle a git que ignore cierto patrón de archivos en el índice. Para ello basta crear un archivo con el nombre .gitignore y escribir adentro los nombres de archivos o carpetas que queremos ignorar. Si queremos que aplique a todo el repositorio, lo ponemos en la raíz. Si es más específico pueden agregarse múltiples en diferentes carpetas. Algo importante es nunca poner .gitignore adentro ya que el .gitignore en sí mismo debe ser commiteado para que afecte a todos los que utilizan el repositorio.

Referencias

Links útiles

- Git official website, `http://git-scm.com/`
- Pro Git Book, `http://git-scm.com/book`
- Git Tutorial, `http://www.slideshare.net/eykanal/git-introductory-talk`
- Git guía rápida, `http://www.edy.es/dev/docs/git-guia-rapida/`
- Basic branching and merging, `http://git-scm.com/book/en/Git-Branching-Basic-Branching-and-Merging`
- A successful branch model, `http://nvie.com/posts/a-successful-git-branching-model/`

Referencias (2)

Links útiles

- Some git tips, `http://mislav.uniqpath.com/2010/07/git-tips/`
- Revert commit by hash, `http://stackoverflow.com/questions/1895059/git-revert-to-a-commit-by-sha-hash`
- Revert bad merge, `https://condor-wiki.cs.wisc.edu/index.cgi/wiki?p=RevertingBadMerges`