

Entendiendo git

cursos@cursodegit.com
@cursodegit

Nuestro blog :
www.aprendegit.com
@aprendegit

Alfonso Alba García
@aalbagarcia

¿porqué estás aquí?

¿Qué nos permite git?

- ➊ Auditoría del código:
 - ➋ Saber qué ha pasado y cuando
 - ➋ Saber quién lo ha hecho

¿Qué nos permite git?

- Desarrollo ágil:
 - Es sencillo crear espacios para probar cosas sin que interrumpa nuestro flujo de trabajo normal
 - Es sencillo cambiar de un tema a otro
 - Refactoring es más sencillo
 - Se integra a la perfección con la metodología TDD

¿Qué nos permite git?

- ➊ Gestión de errores:
 - ➋ Dar marcha atrás cuando nos equivocamos
 - ➋ Volver a un punto anterior del desarrollo

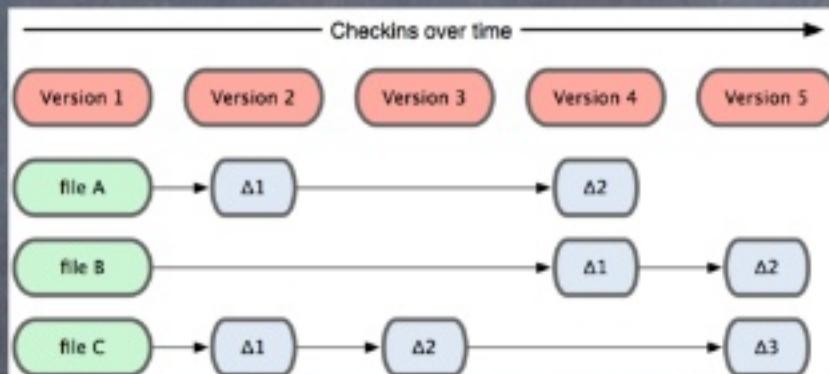
¿Qué nos permite git?

- ➊ Control de código:
 - ➋ Saber qué se ha entregado al cliente
 - ➋ Saber qué se ha desplegado en producción
 - ➋ Saber en qué estado está el desarrollo de una determinada característica

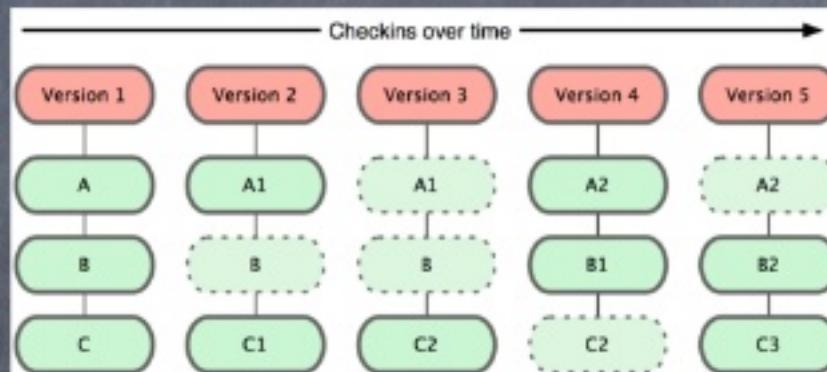
¿Qué nos permite git?

- Trabajo en equipo:
 - Optimiza y permite el trabajo distribuido de equipos de muchas personas
 - Extremadamente flexible: se implementa casi cualquier flujo de trabajo, aunque lo normal es que git te los optimice y los tengas que cambiar...

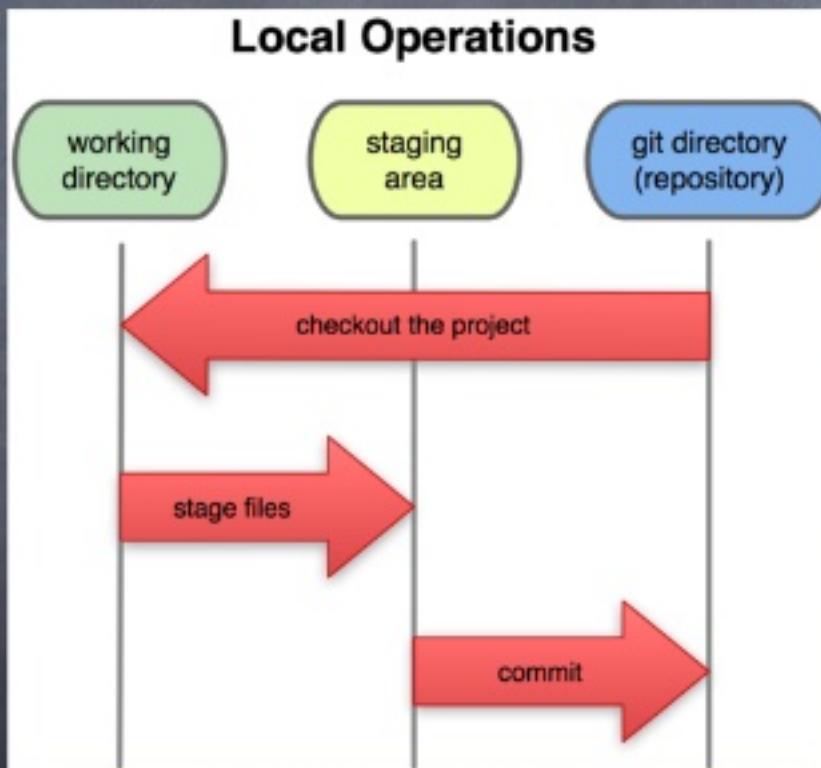
Conceptos básicos



Conceptos básicos



Conceptos básicos



.gitignore

.gitignore

- Tres formas de ignorar ficheros (por orden de prioridad)
 - .git/info/exclude -> sólo este repositorio
 - .gitignore -> todos los que clonian el repositorio
 - .gitignore_local -> todos mis repositorios locales
- git config --get core.excludesfile
- Se aplica el último patrón que se lee

.gitignore: Patrones

- Líneas en blanco: se ignoran
- # Comentarios
- ! negación
- Patrones terminados en / se interpretan como carpetas
- Si el patrón empieza por /, se interpreta como ruta relativa
- El resto de patrones se interpretan como Shell globs

.gitignore: Patrones

- * → Comodín (no incluye el separador de directorios)
 - el patrón web/*.html ignora el fichero web/index.html pero no ignora web/blog/index.html
- [] → Selección.
- ? → cualquier carácter (sólo uno)

.gitignore: ejemplos

- Ejemplos de patrones:

```
.idea/  
*.html  
web/*.*.html  
/*.*.php  
*[xn]ib
```

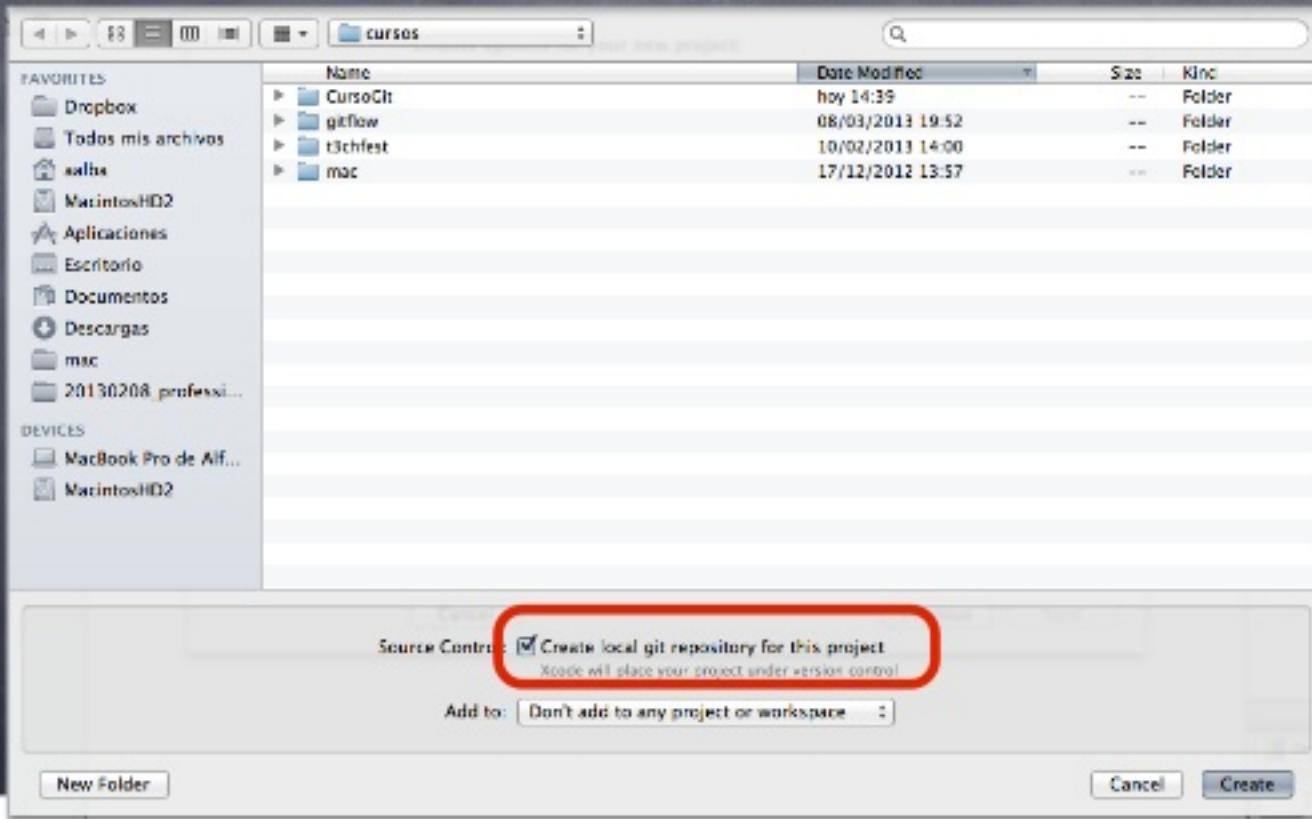
.gitignore

- Más información:
 - [Página de manual de .gitignore](#)
 - [Serie de artículos sobre .gitignore en aprendegit.com](#)

El fichero .gitignore para proyectos XCode

- <http://stackoverflow.com/questions/49478/git-ignore-file-for-xcode-projects>
- <https://gist.github.com/adamgit/3786883>

Creación de un proyecto con git



Creación de un repositorio

Crea una carpeta, entra dentro y ejecuta el comando

```
git init
```

el repositorio se inicializa y aparece la carpeta .git



Estado del repositorio

¿Cómo están los ficheros en los que estamos trabajando?

```
git status
```

Estado del repositorio

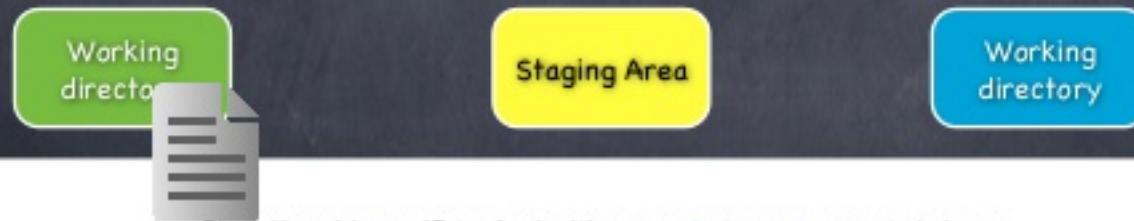
```
$ git status
# On branch lugares
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   app/controllers/backend/application_controller.rb
#       modified:   config/routes.rb
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       app/assets/javascripts/backend/cities.js.coffee
#       app/assets/javascripts/backend/resource.js.coffee
#       app/assets/stylesheets/backend/cities.css.scss
#       app/assets/stylesheets/backend/resource.css.scss
#       app/controllers/backend/cities_controller.rb
#       app/controllers/backend/resource_controller.rb
#       app/helpers/backend/cities_helper.rb
#       app/helpers/backend/resource_helper.rb
#       app/views/backend/
#       app/views/layouts/backend.html.erb
#       test/controllers/backend/cities_controller_test.rb
#       test/controllers/backend/resource_controller_test.rb
#       test/helpers/backend/cities_helper_test.rb
#       test/helpers/backend/resource_helper_test.rb
no changes added to commit (use "git add" and/or "git commit -a")
```

Añadiendo ficheros

Añadimos ficheros al Staging Area

```
git add *
```

```
git add <fichero> <fichero>
```

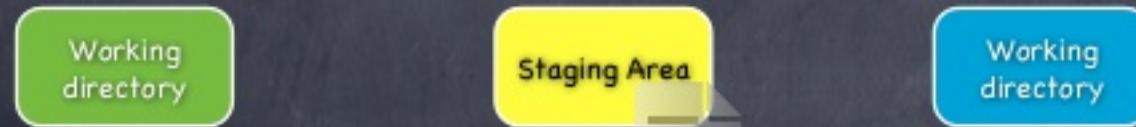


Añadiendo ficheros

Añadimos ficheros al Staging Area

```
git add *
```

```
git add <fichero> <fichero>
```





Añadiendo ficheros

Se puede iniciar un proceso interactivo para seleccionar qué partes de un fichero queremos subir al staging area. Ejecutamos

```
git add -i
```

```
$ git add -i
      staged      unstaged path
 1: unchanged      +2/-2 index.html

*** Commands ***
 1: [s]tatus   2: [u]pdate   3: [r]evert   4: [a]dd untracked
 5: [p]atch   6: [d]iff     7: [q]uit     8: [h]elp
What now>
```



Añadiendo ficheros

Seleccionar la opción “p” y posteriormente el fichero seleccionando el número que nos muestra la interfaz

```
$ git add -i
      staged      unstaged path
1: unchanged      +6/-2 index.html

*** Commands ***
1: [s]tatus  2: [u]pdate  3: [r]evert  4: [a]dd untracked
5: [p]atch   6: [d]iff    7: [q]uit    8: [h]elp
What now> p
      staged      unstaged path
1: unchanged      +6/-2 [i]ndex.html
Patch update>> 1
      staged      unstaged path
* 1: unchanged      +6/-2 [i]ndex.html
Patch update>>
```



Añadiendo ficheros

En este punto, presionamos de nuevo “Intro” y entramos en el modo interactivo



```
$ git add -i
      staged      unstaged path
 1: unchanged          +6/-2 index.html

*** Commands ***
 1: [s]tatus   2: [u]pdate   3: [r]evert   4: [a]dd untracked
 5: [p]atch   6: [d]iff     7: [q]uit     8: [h]elp
What now> p
      staged      unstaged path
 1: unchanged          +6/-2 [i]ndex.html
Patch update>> 1
      staged      unstaged path
* 1: unchanged          +6/-2 [i]ndex.html
Patch update>>
diff --git a/index.html b/index.html
index b08ce9a..3c50d0c 100644
--- a/index.html
+++ b/index.html
@@ -21,8 +21,8 @@
<li><a href="#">Github</a></li>
<li><a href="#">XCode</a></li>
<li><a href="#">¿Quiénes somos?</a></li>
-
<li><a href="#">Cursos</a></li>
-
<li><a href="#">Contacto</a></li>
+
<li><a href="#">Cursos de git</a></li>
+
<li><a href="#">Contacta con nosotros</a></li>
</ul>
</div>
</div>
Stage this hunk [y,n,q,a,d,/,,j,J,g,e,?]?
```



Añadiendo ficheros

Puedes seleccionar qué hacer con los pedazos (hunks) de código que has modificado en el fichero

```
$ git add -i
...
Patch update>>
diff --git a/index.html b/index.html
index b08ce9a..3c50d0c 100644
--- a/index.html
+++ b/index.html
@@ -21,8 +21,8 @@
     <li><a href="#">Github</a></li>
     <li><a href="#">XCode</a></li>
     <li><a href="#">¿Quiénes somos?</a></li>
-    <li><a href="#">Cursos</a></li>
-    <li><a href="#">Contacto</a></li>
+    <li><a href="#">Cursos de git</a></li>
+    <li><a href="#">Contacta con nosotros</a></li>
     </ul>
     </div>
 </div>
```

Stage this hunk [y,n,q,a,d,/,j,J,g,e,?]?



Añadiendo ficheros

Para saber qué es cada una de las opciones, pulsa ?
y luego “Intro”

```
$ git add -i
...
Patch update>>
diff --git a/index.html b/index.html
index b08ce9a..3c50d0c 100644
--- a/index.html
+++ b/index.html
@@ -21,8 +21,8 @@
     <li><a href="#">Github</a></li>
     <li><a href="#">XCode</a></li>
     <li><a href="#">¿Quiénes somos?</a></li>
-     <li><a href="#">Cursos</a></li>
-     <li><a href="#">Contacto</a></li>
+     <li><a href="#">Cursos de git</a></li>
+     <li><a href="#">Contacta con nosotros</a></li>
     </ul>
   </div>
</div>
```

Stage this hunk [y,n,q,a,d,/,j,J,g,e,?] ?

Añadiendo ficheros

```
Stage this hunk [y,n,q,a,d/,j,J,g,e,?] ?  
y - stage this hunk  
n - do not stage this hunk  
q - quit; do not stage this hunk nor any of the remaining ones  
a - stage this hunk and all later hunks in the file  
d - do not stage this hunk nor any of the later hunks in the file  
g - select a hunk to go to  
/ - search for a hunk matching the given regex  
j - leave this hunk undecided, see next undecided hunk  
J - leave this hunk undecided, see next hunk  
k - leave this hunk undecided, see previous undecided hunk  
K - leave this hunk undecided, see previous hunk  
s - split the current hunk into smaller hunks  
e - manually edit the current hunk  
? - print help
```

Añadiendo ficheros

```
Stage this hunk [y,n,q,a,d/,j,J,g,e,?] ?  
y - Pasar este trozo al área de ensayo  
n - No pasar este trozo  
q - terminar. No pasa este trozo ni ninguno de los que quedan al staging area  
a - Pasa este pedazo y todos los que quedan al staging area  
d - No pases este trozo ni ninguno de los que quedan en el fichero al staging area  
g - Selecciona un trozo para situarnos en el  
/ - Busca un trozo que encaje con una expresión regular  
j - Mantener este trozo como "indeciso" y ve al siguiente hunk marcado como "indeciso"  
J - Mantener este trozo como "indeciso" y ve al siguiente  
k - Mantener este trozo como "indeciso" y ve al siguiente hunk marcado como "indeciso"  
K - siguiente hunk marcado como "indeciso" y ve al anterior  
s - rompe este trozo en trozos más pequeños  
e - editar este trozo manualmente  
? - ayuda
```

Añadiendo ficheros

Al terminar, podemos usar la opción “s” para ver el estado y posteriormente “q” para terminar

Cambios subidos al staging area

Cambios fuera del staging area ¡no estarán en el siguiente commit!

```
What now> s
      staged          unstaged path
1:          +2/-2           +4/-0 index.html

*** Commands ***
1: [s]tatus   2: [u]pdate   3: [r]evert   4: [a]dd untracked
5: [p]atch    6: [d]iff     7: [q]uit     8: [h]elp
What now> q
```

Añadir ficheros

Más información:

Sección 6.2 - Pro git

Añadiendo ficheros

Pasamos los ficheros al repositorio

```
git commit -m'Mensaje del commit'
```

Utilizad siempre un mensaje corto y descriptivo
Evitad mensajes como “trabajando” “...” que no
aportan nada al que lo lee

Working
directory

Staging Area

Working
directory



git
aprendegit.com



Añadiendo ficheros

Si nos hemos olvidado algún fichero al hacer el último commit, podemos añadirlo al staging area con

```
git add <fichero>
```

y modificar el commit ejecutando

```
git commit --amend
```

La opción --amend también permite modificar el comentario del commit



Añadiendo ficheros

Recuerda que

```
git commit --amend
```

va a sobreescibir la historia del proyecto

Borrando y moviendo ficheros

Borrar un fichero del índice

```
git rm <fichero>
```



Mover un fichero

```
git mv <ruta actual> <ruta nueva>
```



Etiquetando los commits

Etiquetamos los commits para poder recuperar el código fuente en un estado determinado en el futuro

Etiqutar el último commit de la rama activa

```
git tag <nombre> -m'Mensaje'
```



Etiqatar un commit determinado

```
git tag <nombre> -m'Mensaje' <commit>
```

Etiquetando los commits

Borrar una etiqueta

```
git tag -d <nombre>
```



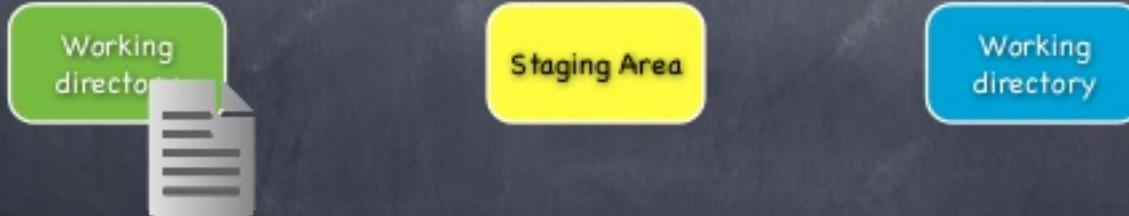
Ver todas las etiquetas

```
git tag -l
```

Sacando ficheros del Staging Area

Deshacer los cambios de un fichero que todavía no está en Staging Area

```
git checkout -- <fichero>
```



Sacando ficheros del Staging Area

Sacar un fichero que está en el Staging Area

```
git reset HEAD <fichero>
```



Sacando ficheros del Staging Area

Sacar un fichero que está en el Staging Area

```
git reset HEAD <fichero>
```



Sacando ficheros del Staging Area

Sacar todos los ficheros que están en el Staging Area

```
git reset HEAD
```





Stashing

Stash: área en el que podemos guardar cambios en ficheros para aplicarlos luego

Útil para:

Poder interrumpir una tarea y trabajar en otra
Mover cambios de un sitio a otro

Stashing

Almacenar en el stash los cambios del working copy
en ese momento

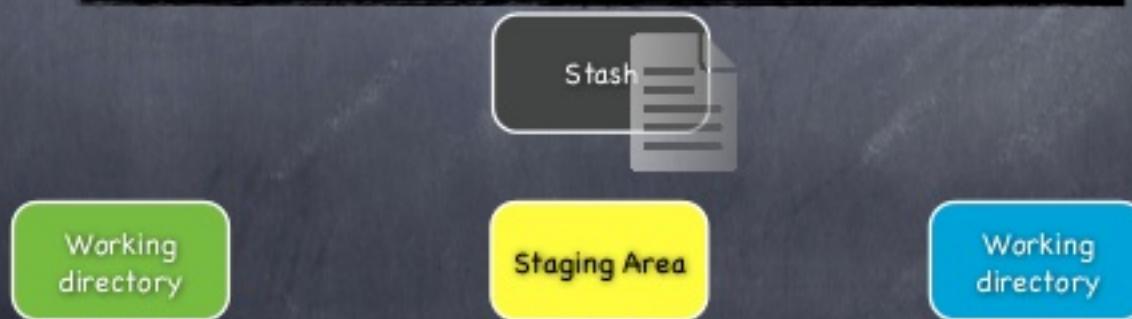
```
git stash save "mensaje"
```



Stashing

Almacenar en el stash los cambios del working copy
en ese momento

```
git stash save "mensaje"
```



Stashing

Muestra el listado de entradas del stash

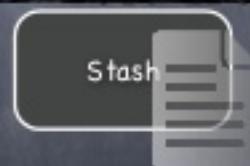
```
git stash list
```



Stashing

Recuperar algo almacenado en el Stash

```
git stash pop <nombre>
```



Working
directory

Staging Area

Working
directory

Stashing

Recuperar algo almacenado en el Stash

```
git stash pop <nombre>
```



Stashing

Recuperar una entrada del stash y guarda una copia

```
git stash apply <nombre>
```



Borrar una entrada del stash sin aplicarla

```
git stash drop <nombre>
```

Ramas

Una rama en git es una etiqueta

Creación de una rama en el último commit de la
rama activa

```
git branch <nombre>
```



Crear una rama en un commit

```
git branch <nombre> <commit>
```

Ramas

Crear una rama y activarla

```
git checkout -b <nombre>
```



Cambiar de rama

```
git checkout <nombre>
```

Ramas

Para incorporar los cambios de feature_x en master cambiamos a la rama master

```
git checkout master
```

e incorporamos los cambios de la rama feature_x

```
git merge feature_x
```



Ramas: rebase

Para incorporar los cambios de feature_x en master cambiamos a la rama master

```
git checkout master
```

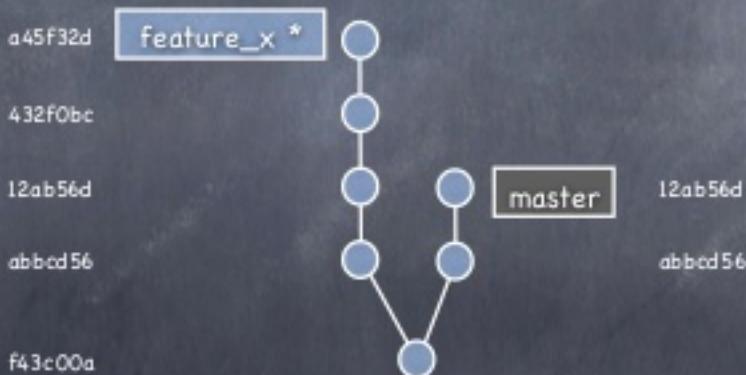
e incorporamos los cambios de la rama feature_x

```
git rebase feature_x
```



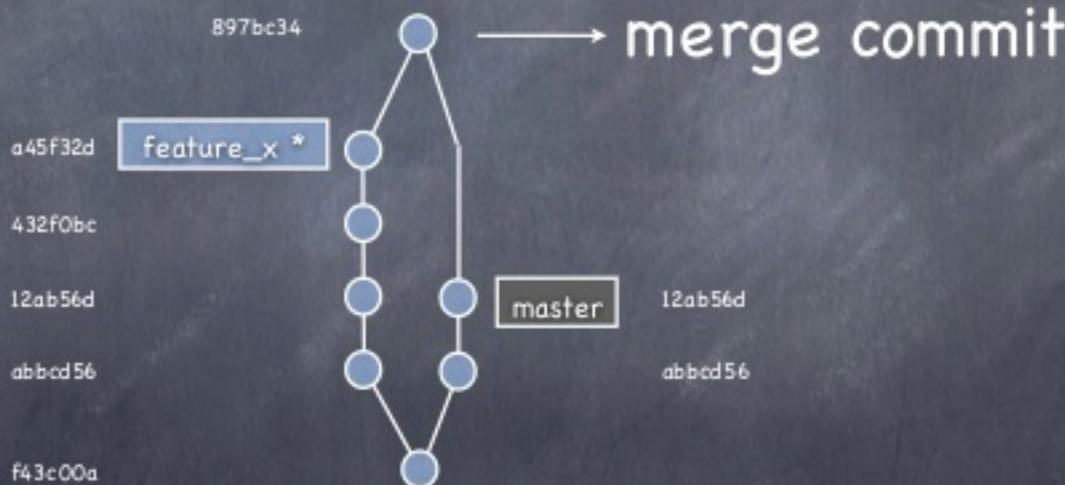
merge vs rebase

git merge master



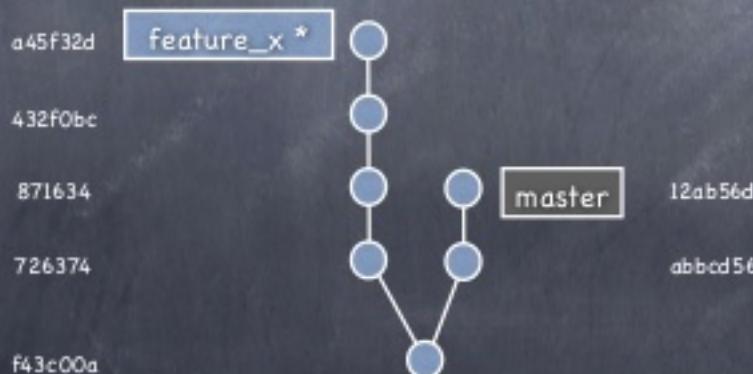
merge vs rebase

git merge master



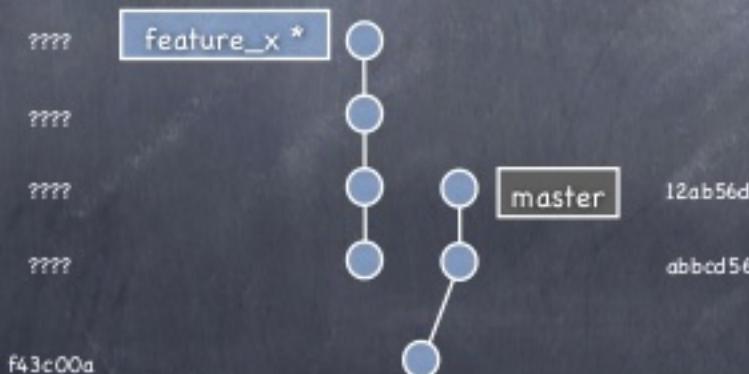
merge vs rebase

git rebase master



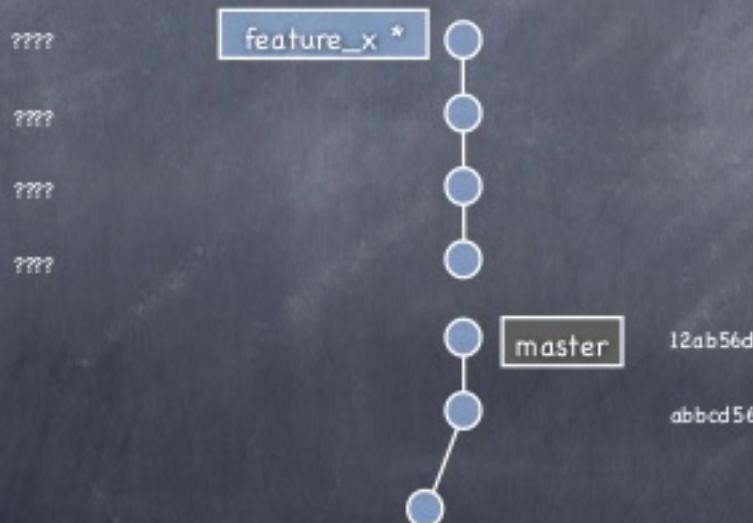
merge vs rebase

git rebase master



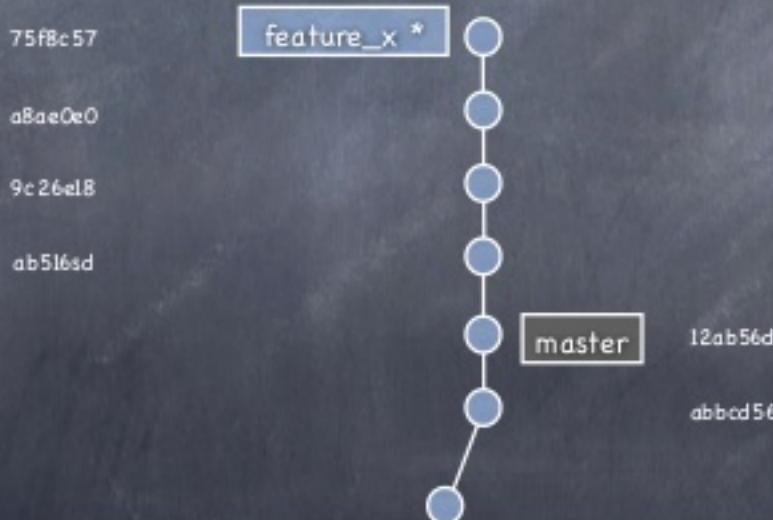
merge vs rebase

git rebase master



merge vs rebase

git rebase master



Repositorios remotos

Para subir nuestro repositorio a un repositorio remoto necesitamos conocer la URL del repositorio

The screenshot shows a GitHub repository page for 'aprendegit / articulo-sobre-gitflow'. At the top, there's a search bar and navigation links for Explore, Gist, Blog, and Help. On the right, there are profile icons for 'aalbagarcia' and standard GitHub navigation buttons for Pull Request, Unwatch, Star, Fork, and Settings.

The main content area displays the repository details: 'aprendegit / articulo-sobre-gitflow', a brief description ('Repositorio usado para escribir la serie de artículos sobre git-flow en aprendegit.com'), and a 'Code' tab. Below the code tab, there are download options: 'Clone in Mac', 'ZIP', 'HTTP' (which is selected), 'SSH', and 'GH Read-Only'. The 'HTTP' option is highlighted with a red box.

Below the download options, there's a note about 'ReadWrite access'. The repository has 1 commit, authored by 'aalbagarcia' 8 days ago. The commit hash is '87ee184e2b'. The bottom of the page shows the 'Files', 'Commits', 'Branches', and 'Tags' tabs, along with the 'git' logo and the URL 'cursodegit.co'.

Repositorios remotos

Para añadir un repositorio remoto ejecutamos

```
git remote add <nombre> <url>
```



Se puede ver la lista de remotos con

```
git remote -v
```

Repositorios remotos

Borrar un repositorio remoto

```
git remote remove <nombre>
```



Push

Para subir todos nuestros commits de la rama activa al repositorio remoto ejecutamos

```
git push <remoto> <rama remota>
```



usando el repositorio que hemos creado durante la clase

```
git push origin master
```

Tracking branches

Una “Tracking branch” es una rama local que está relacionada con una remota.

Si estás en una rama local que es “tracking branch” de una rama remota y ejecutas el comando

```
git push
```



git automáticamente lo convierte en

```
git push <tracking branch> <origin/rama  
remota>
```

Tracking branches

Podemos crear una “tracking branch” de una rama remota ejecutando

```
git branch --track <rama local> <remoto>/<rama remota>
```

que siguiendo con el ejemplo que hacemos en clase sería

```
git branch --track mirama origin/master
```

Tracking branches

En cualquier momento podemos hacer que una tracking branch deje de serlo ejecutando

```
git branch --unset-upstream <rama>
```



En cualquier momento podemos hacer que una rama local sea una tracking branch ejecutando

```
git branch --unset-upstream-to=<remoto>/<rama remota> <rama local>
```

Pull / Fetch

Para descargarnos los commits del repositorio remoto

```
git fetch <remoto>
```



Este comando actualiza los commits únicamente. Si queremos ahora incorporar los cambios de una rama remota en una rama local ejecutamos

```
git checkout <rama local>  
git merge <remoto>/<rama remota>
```

Pull / Fetch

Supongamos que tenemos una rama local “feature_x” que es tracking branch de una rama remota “origin/feature_x”

La rama remota ha sido actualizada por otro miembro del equipo y queremos actualizarnos

```
git checkout feature_x  
git fetch origin  
git merge origin/feature_x
```



Pull / Fetch

Esta operación es tan común que git nos da un atajo

```
git checkout feature_x  
      git fetch origin  
git merge origin/feature_x
```



```
git checkout feature_x  
      git pull
```

Pull / Fetch

git-pull por defecto hace un merge, ¿quieres que en lugar de merge haga un rebase?

```
git checkout feature_x  
git pull --rebase
```



Buenas prácticas

- o Usa ramas locales
- o No dejar los mensajes de los commits vacíos
- o Usar mensajes descriptivos que indiquen qué contiene el commit. ¿Qué quiere decir "trabajando"? ¿Y "...?"
- o Si utilizas un sistema de tareas o bug tracking que se enlaza con git (como Jira o redmine) enlaza tus commits con tus tickets, bugs o tareas
- o Haz commits pequeños. Haz commit a menudo.
- o Usa ramas locales
- o Atomiza los commits todo lo que puedas. Intenta no mezclar en un mismo commit código de cosas que no tengan nada que ver entre sí.
 - o Por ejemplo: si has hecho tres vistas (tabla de registros, detalle de registro y formulario de alta de registro) haz tres commits, no hagas sólo uno.
- o El flujo TDD es escribir test - rojo - escribir código - verde - commit.
 - o Si luego te quedan muchos commits, siempre puedes hacer squashing.
- o Usa ramas locales y utiliza rebase para incorporar los cambios de las otras ramas.
- o Haz squashing de tus commits para eliminar aquellos que sean irrelevantes para el resto del equipo
- o Evita sobreescribir la historia del repositorio (a no ser que sepas lo que estás haciendo)
- o Por cierto, usa ramas locales

¡Gracias!