

Gestión de proyectos software con Git y Github



GIT

1. Proyecto, directorio y versión

GIT

◆ GIT: gestor de proyectos software

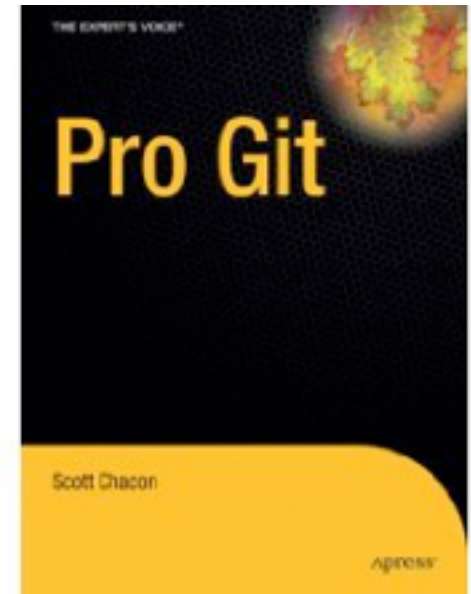
- Desarrollado por Linus Torwalds para Linux

◆ Diseñado para desarrollo distribuido

- Cada desarrollador trabaja de forma independiente en su propio repositorio
 - Sincroniza el repositorio con otro cuando necesita
- Uno de los repositorios puede utilizarse como repositorio de referencia

◆ Tutorial Web y eBook

- <http://git-scm.com/book/es>

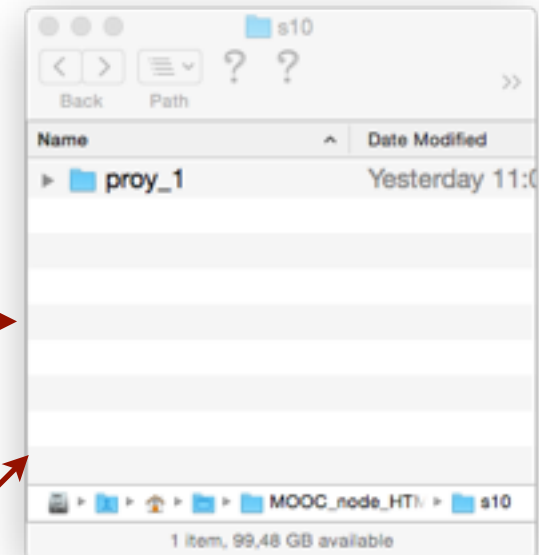


El directorio del proyecto

- ◆ Un **proyecto** se suele gestionar **en un directorio** (o carpeta)
 - El directorio contiene todos los ficheros del proyecto
- ◆ **Explorador de ficheros**: muestra el contenido de un directorio gráficamente
 - Hacer clic sobre un objetos gráfico ejecuta un comando predefinido
- ◆ **Terminal de comandos**: ejecuta comandos en **directorio de trabajo**
 - El **directorio de trabajo** asociado es la base de las rutas (paths) relativas
 - Los objetos se identifican con rutas (paths) absolutas o relativas

```
venus:s jq$ pwd
/Users/jq/Desktop/MOOC_node_HTML5/s10/s
venus:s jq$ ls
venus:s jq$ ls -a
.  ..
venus:s jq$ mkdir proy_1
venus:s jq$ ls
proy_1
venus:s jq$ ls -a
.  ..      proy_1
venus:s jq$ cd proy_1/
venus:proy_1 jq$ pwd
/Users/jq/Desktop/MOOC_node_HTML5/s10/s/proy_1
venus:proy_1 jq$
```

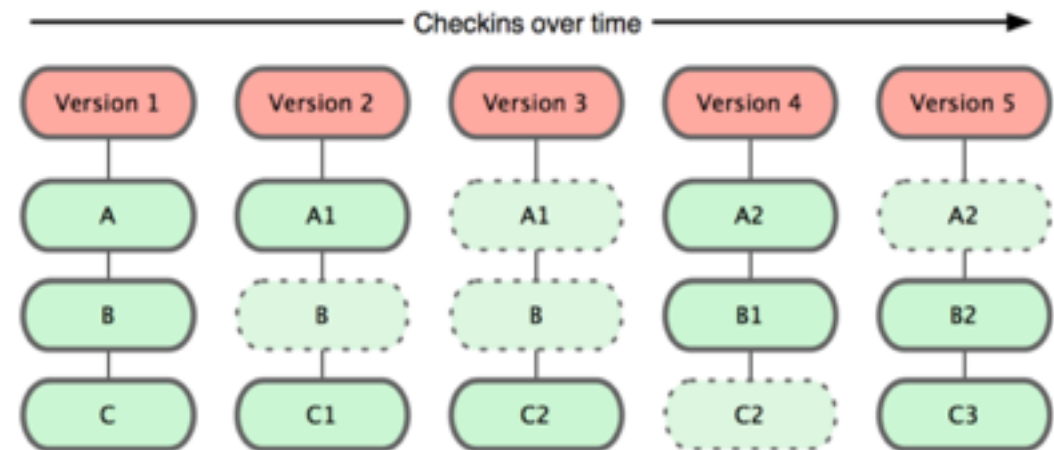
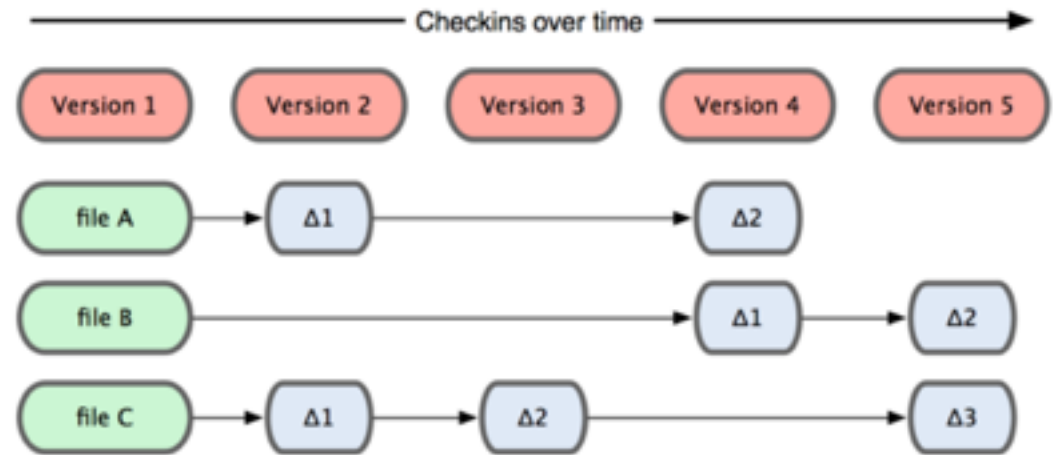
Terminal de comandos



Explorador de ficheros

Historia de un proyecto

- ◆ Historia de un proyecto:
 - ▶ Historia de cambios en el directorio del proyecto
- ◆ Versión (Commit)
 - ▶ Punto de la historia del proyecto que puede ser restaurado (reconstruido)
- ◆ Se debe consolidar versión en los puntos del desarrollo que deseemos poder volver atrás en el futuro
 - ▶ Versiones frecuentes facilitan el mantenimiento y la legibilidad de un programa



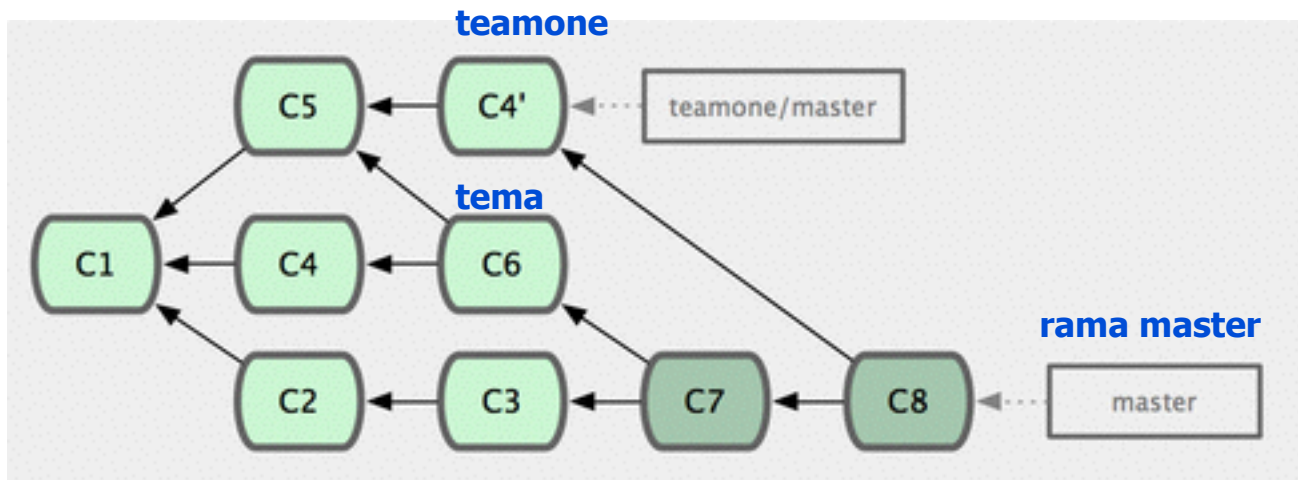
Árbol de versiones

◆ Proyectos software son complejos

- Suelen generar un árbol de versiones

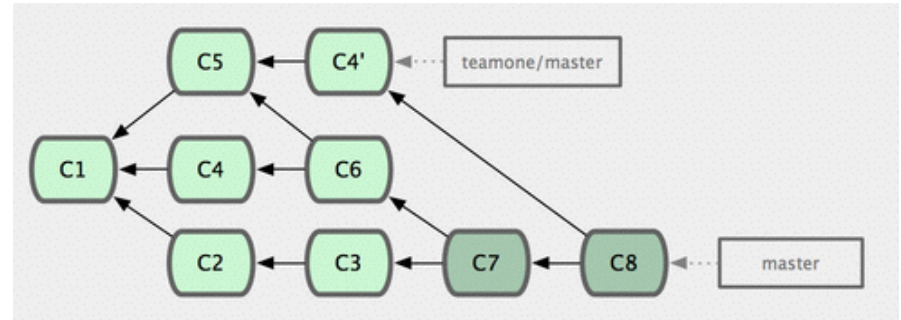
◆ La **rama principal** del proyecto se denomina **master**

- En este árbol hay además 2 ramas: **tema** y **teamone**
 - Una rama suele realizar un desarrollo separado
- **Las ramas se suelen integrar en master**, una vez acabadas
 - P. e., la integración puede realizarse con el comando: **git merge teamone**



*de Scott Chanson: <http://progit.org/book/>

Repositorio y versión



*de Scott Chanson: <http://progit.org/book/>

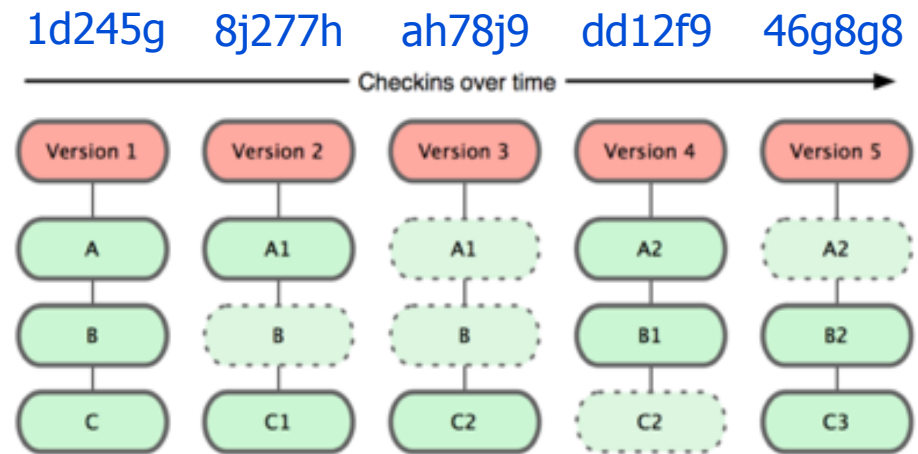
◆ Un **repositorio git** es un “directorio donde gestionar versiones”

- El comando “**git init**” invocado en el directorio
 - Habilita el directorio como un **repositorio git**
 - Puede **guardar o restaurar versiones**
- Las versiones se guardan en el directorio oculto **.git**

◆ **Versión (commit)**

- Directorio (proyecto) congelado en un momento dado
 - Incluyendo todos sus ficheros y subdirectorios
- Punto de sincronización del proyecto que puede restaurarse

Identificador de versión (SHA1)



◆ Cada versión generada por GIT se identifica con

- Número aleatorio único (clave SHA1)
 - ejemplo: **973751d21c4a71f13a2e729ccf77f3a960885682**

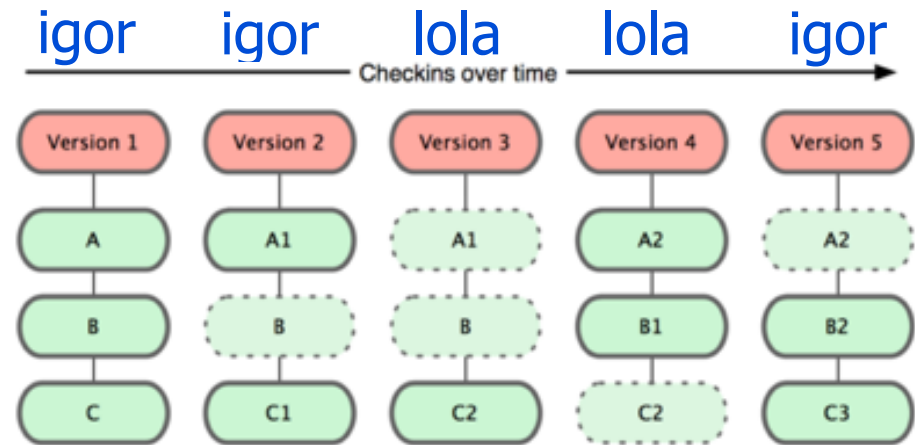
◆ GIT permite equipos de desarrollo distribuidos

- Los repositorios se pueden clonar sin problemas
 - Ninguna versión en ningún otro repositorio utilizará el mismo identificador

◆ El identificador es muy largo y se suelen utilizar

- los 7-8 dígitos iniciales (únicos en un proyecto): **973751d2**
 - Comandos GIT: permiten **identificadores cortos o largos**

Colaboración y Firma



*de Scott Chanson: <http://git-scm.com/book>

- ◆ GIT esta pensado para trabajar en grupo
 - Toda operación va firmada por su autor
 - Al configurar GIT se da el nombre e email del autor
- ◆ Un usuario puede copiar o clonar otro repositorio
 - Y continuar el desarrollo por su cuenta sobre la copia
- ◆ Dos repositorios pueden volver a sincronizarse
 - Aunque integrar las nuevas versiones puede ser complejo

Configurar GIT

El comando **"git config"** permite manejar opciones de configuración.

Las opciones configuradas pueden afectar a distintos ámbitos (proyectos):

- Para todos los proyectos en el sistema.

Usar opción **--system**. La configuración se guarda en **/etc/gitconfig**

- Para todos los proyectos del usuario.

Usar opción **--global**. La configuración se guarda en **~/.gitconfig**

- Sólo para el proyecto actual.

Sin opción. La configuración se guarda en **.git/config**

Consultar todas las opciones existentes: **git help config**

Para firmar correctamente contribuciones y versiones debemos configurar:

```
$ git config --global user.name "Pedro Ramirez"
```

```
$ git config --global user.email pramirez@dit.upm.es
```

Consultar el valor de todas las opciones configuradas:

```
$ git config --list
```

```
user.name=Pedro Ramirez
```

```
user.email=pramirez@dit.upm.es
```

```
color.ui=true
```

Consultar el valor de una opción:

```
$ git config user.name
```

```
Pedro Ramirez
```

Ayuda

Ayuda en línea de comandos:

\$ git help	# Muestra lista con los comandos existentes
\$ git help comando	# Ayuda sobre comando especificado
\$ git help add	# Ayuda sobre el comando add
\$ git add --help	# Equivalente a anterior
\$ man git-add	# Equivalente a anterior

Manual de referencia, chuletas, videos, otros enlaces:

<http://git-scm.com/doc>

<http://ndpsoftware.com/git-cheatsheet.html>

https://na1.salesforce.com/help/doc/en/salesforce_git_developer_cheatsheet.pdf



GIT

2. Proyecto quiz-2015 en GITHUB

Proyecto Quiz

- ◆ **Versión 1:** Esqueleto del proyecto con express-generator
- ◆ **Versión 2:** Primera página y el favicon
- ◆ **Versión 3:** Primera pregunta
- ◆ **Versión 4:** Vistas parciales y marco
- ◆ **Versión 5:** CSS adaptable a móviles y PCs
- ◆ **Versión 6:** Despliegue en la nube (Heroku)
- ◆ **Versión 7:** La base de datos: sequelize.js y SQLite
- ◆ **Versión 8:** Desplegar en Heroku con Postgres
- ◆ **Versión 9:** Lista de preguntas
- ◆ **Versión 10:** Autoload de la DB
- ◆ **Versión 11:** Crear preguntas
- ◆ **Versión 12:** Validación de entradas
- ◆ **Versión 13:** Editar preguntas
- ◆ **Versión 14:** Borrar preguntas
- ◆ **Versión 15:** Crear comentario
- ◆ **Versión 16:** Autenticación y sesión
- ◆ **Versión 17:** Autorización
- ◆ **Versión 18:** Moderación de comentarios
- ◆ **Versión 19:** HTTPS - HTTP Seguro

Objetivo: Crear un pequeño portal Web con un juego de adivinanzas (quizes) usando MVC y vistas adaptables a móvil. Quiz ilustra también el uso de herramientas de gestión de proyectos. El proyecto solo tiene una rama: **master**

El proyecto Quiz en **GITHUB**
<https://github.com/jquemada/quiz-2015>

Proyecto desplegado y operativo en **heroku**
<https://quiz-2015.herokuapp.com/>



GitHub repository page for `jquemada/quiz-2015`. The page shows the repository name, a search bar, and a list of commits. The repository is titled "Proyecto de CORE 2015 y @NodeMOOC". It has 19 commits, 2 branches, 0 releases, and 2 contributors. The current branch is `quiz-2015`. The file explorer on the left shows the directory structure: `bin`, `certs`, `controllers`, `models`, `public`, `routes`, `views`, `.gitignore`, `Procfile`, `app.js`, `commands_to_generate_keys.txt`, and `package.json`.

Quiz en GITHUB

branches (ramas)

Commits (versiones)

código: directorios, ficheros, ..

URL que identifica el repositorio:
<https://github.com/jquemada/quiz-2015>

30 lines (29 sloc) 0.584 kb

```
1 {
2   "name": "quiz",
3   "version": "0.0.0",
4   "private": true,
5   "scripts": {
6     "start": "node ./bin/www"
7   },
8   "engines": {
9     "node": "0.10.x",
10    "npm": "1.4.x"
11  },
12  "devDependencies": {
13    "sqlite3": "^3.0.4"
14  },
15  "dependencies": {
16    "body-parser": "~1.8.1",
17    "cookie-parser": "~1.3.3",
18    "debug": "~2.0.0",
19    "ejs": "~0.8.5",
20    "express": "~4.9.0",
21    "express-partials": "^0.3.0",
22    "express-session": "~1.0.3",
23    "method-override": "^2.3.1",
24    "morgan": "~1.3.0",
25    "pg": "^4.1.1",
26    "sequelize": "^2.0.0-rc4",
27    "serve-favicon": "~2.1.3"
28  }
29 }
```

Commits (versiones)

Último commit (versión)

identificador
"corto" de commit

Diferencias con la versión anterior
del fichero views/index.ejs:
rojo: eliminado, **verde:** añadido

commits
anteriores:
historia de
versiones

343a8ef

4 views/index.ejs

-5,7 +5,7

```
<link rel='stylesheet' href='/stylesheets/style.  
</head>  
<body>  
- <h1><%= title %></h1>  
- <p>Welcome to <%= title %></p>  
+ <h1>Bienvenido a <%= title %></h1>  
+ <p>El portal donde podrá crear sus propios juegos  
</body>  
</html>
```

Clonar e inspeccionar quiz-2015

```
# Un repositorio público en GITHUB, o en otro servidor al que tengamos acceso,  
# puede clonarse en nuestro ordenador con: git clone <URL_repositorio>  
# -> la copia incluye el proyecto completo con toda su historia de versiones  
  
# Podemos clonar el proyecto quiz-2015 de la copia en GITHUB con el comando:  
$ git clone https://github.com/jquemada/quiz-2015  
# El proyecto se copia en un nuevo directorio llamado quiz-2015  
  
# También podemos indicar cual es el nombre del directorio a crear  
$ git clone https://github.com/jquemada/quiz-2015 mi_proyecto  
# El proyecto se copia ahora en un nuevo directorio llamado mi_proyecto  
  
$ cd quiz-2015 # Entramos en el directorio clonado quiz-2015 (o mi-proyecto)  
  
$ git log --oneline # Muestra las versiones del proyecto  
  
$ git checkout <commit_id_SHA1> # descongela las versiones de la historia  
$ git checkout master # vuelve a la rama (última versión: Quiz 19)  
$
```


GITHUB App for MAC & for Windows I

GITHUB App es una herramienta gráfica muy eficaz para gestionar proyectos git localmente en el PC. Solo está soportada para MAC y para Windows:

Descargar (e instalar) GITHUB for MAC: <https://mac.github.com>

Descargar (e instalar) GITHUB for Windows: <https://windows.github.com>

El proyecto Quiz de GITHUB (<https://github.com/jquemada/quiz-2015>), una vez clonado en un directorio local, puede añadirse a GITHUB App tal y como se indica en la figura.

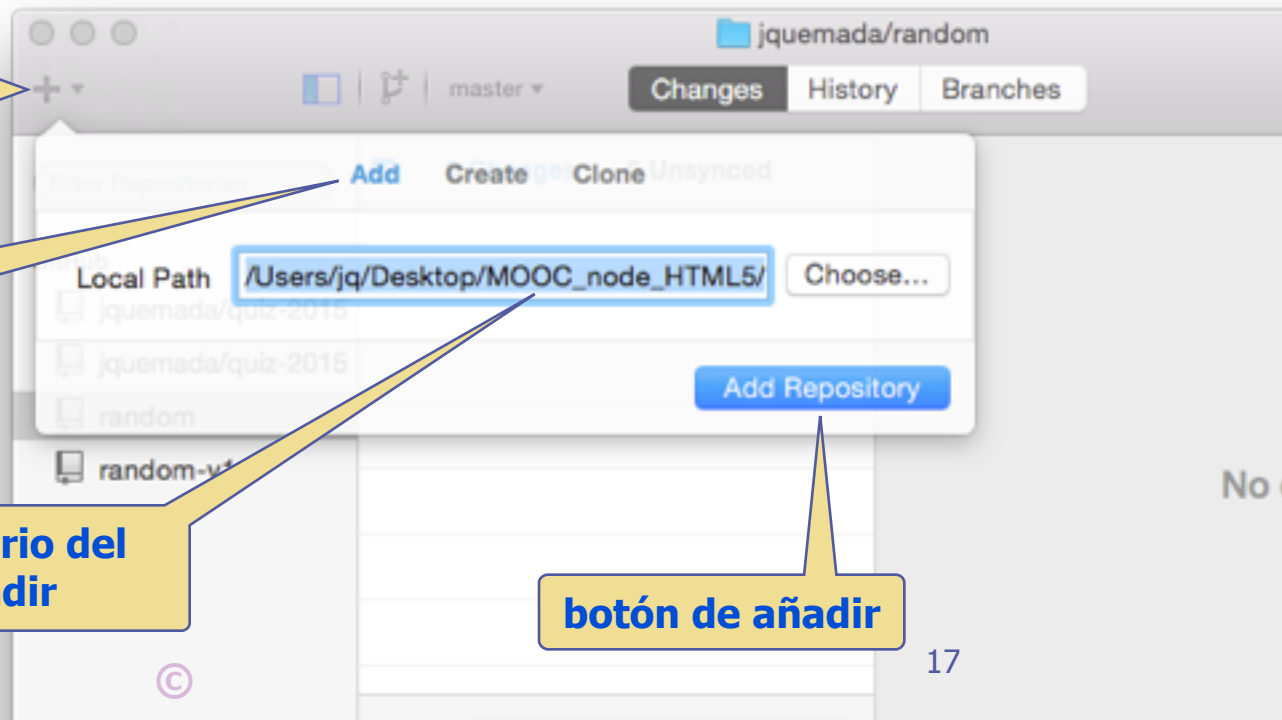
Desplegable para:

- **Añadir (Add) proyecto**
- **Crear (Create) proyecto**
- **Clonar (Clone) proyecto**

**Pestaña de
añadir proyecto**

**ruta al directorio del
proyecto a añadir**

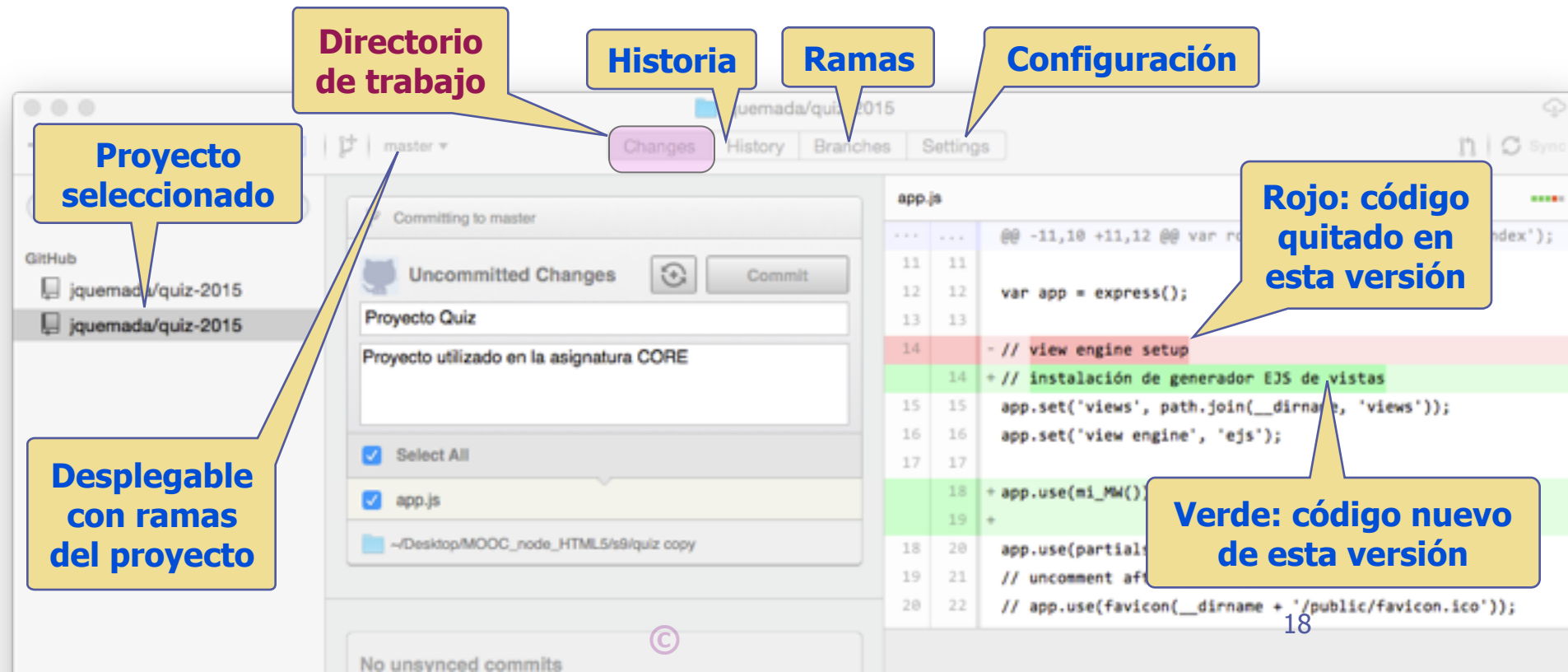
botón de añadir



GITHUB App for MAC & for Windows II

GITHUB App for MAC/Windows **gestiona proyectos git localmente en nuestro PC**. Permite

- Ver y gestionar el **directorio de trabajo** del proyecto y los **cambios realizados**.
- Ver las **versiones de un proyecto (historia)** y los **cambios** realizados en cada versión.
- Ver y gestionar las **ramas de un proyecto** y su sincronización con **repositorios remotos**.



GITHUB App for MAC & Windows III

Seleccionando la historia de una rama del proyecto aparecen todas las versiones (commits) de dicha rama.

Seleccionando una versión podemos ver todas las diferencias con la versión anterior.

El código añadido se resalta en verde y el código eliminado se resalta en rojo.

The screenshot displays the GitHub web interface for a repository named 'jqemada/quiz-2015'. The top navigation bar includes 'Changes', 'History', 'Branches', and 'Settings'. The 'History' tab is selected, showing a list of 19 commits. The commit list on the left includes entries like 'Crear comentario', 'Borrar pregunta', 'Editar pregunta', 'Validación', 'Crear pregunta', 'Autoload', 'Lista de Preguntas', 'Despliegue DB en Heroku', 'Modelo y base de datos', 'Despliegue en Heroku', 'Diseño responsivo', 'express-partials', 'Primera pregunta', 'Primera página y favicon', and 'esqueleto express-generator'. The 'Lista de Preguntas' commit is selected, showing a diff view. The diff view compares the selected commit with the previous one. The code is color-coded: red for removed code and green for added code. Callouts highlight these features: 'Historia' points to the 'History' tab, 'Versión' points to the selected commit, 'Rojo: código quitado en esta versión' points to the red lines in the diff, and 'Verde: código nuevo de esta versión' points to the green lines in the diff.

Historia

Versión

Rojo: código quitado en esta versión

Verde: código nuevo de esta versión

```
... -4,8 +4,6 @@ var app = require('../app');
4 4
5 5 app.set('port', process.env.PORT || 3000);
6 6
7 - console.log("Llega aqui 1")
8 7 var server = app.listen(app.get('port'), function() {
9 8   debug('Express server listening on port ' + server.address().port);
10 9 });
11 - console.log("Llega aqui 2")

controllers/quiz_controller.js
... -1,19 +1,28 @@
1 1 var models = require('../models/models');
2 2
3 - // GET /quizes/question
4 - exports.question = function(req, res) {
5 -   models.Quiz.findAll().then(function(quiz) {
6 -     res.render('quizes/question', { pregunta: quiz[0].pregunta});
7 3 + // GET /quizes
8 4 + exports.index = function(req, res) {
9 5 +   models.Quiz.findAll().then(function(quizes) {
10 6 +     res.render('quizes/index.ejs', { quizes: quizes});
11 7   })
12 8   };
13 9
14 - // GET /quizes/answer
15 10 + // GET /quizes/:id
16 11 + exports.show = function(req, res) {
17 12 +   models.Quiz.find(req.params.quizId).then(function(quiz) {
18 13 +     res.render('quizes/show', { quiz: quiz});
19 14 +   })
20 15 + };
21 16 +
22 - // GET /quizes/:id/answer
23 17 + exports.answer = function(req, res) {
```

GITHUB App for MAC & for Windows IV

Abrir o crear proyectos

Directorio o carpeta de S.O.

Proyectos

- View on GitHub
- Open in Finder
- Open in Terminal
- Open in Atom
- Remove

Click con Botón der. del ratón (^Click) en proyecto despliega esta ventana de acceso

Atom es un editor muy sencillo y eficaz para editar todos los ficheros del proyecto. Si se utiliza GITHUB for MAC/Windows es conveniente instalarlo.

Consola de comando en directorio de trabajo del proyecto

```
Last login: Tue Mar 17 18:33:52 on ttys001
venus:quiz copy jq$
venus:quiz copy jq$
venus:quiz copy jq$ pwd
/Users/jq/Desktop/HOOC_node_HTML5/s9/quiz copy
venus:quiz copy jq$ ls
Procfile      models        quiz.sqlite
app.js        node_modules  routes
bin           package.json  views
controllers   public
venus:quiz copy jq$
```

Name	Date Modified
app.js	Today 07:58
bin	31 Jan 2015 1
controllers	1 Feb 2015 08
models	1 Feb 2015 08
node_modules	2 Feb 2015 13
package.json	31 Jan 2015 1
Procfile	31 Jan 2015 1
public	29 Dec 2014 1
favicon.ico	29 Dec 2014 1
images	25 Nov 2014 1
javascripts	25 Nov 2014 1
stylesheets	1 Feb 2015 08
quiz.sqlite	2 Feb 2015 15
routes	1 Feb 2015 08
index.js	1 Feb 2015 14
views	2 Feb 2015 14
error.ejs	25 Nov 2014 1
index.ejs	31 Jan 2015 1
layout.ejs	2 Feb 2015 14
quizes	2 Feb 2015 14
_form.ejs	1 Feb 2015 07
answer.ejs	31 Jan 2015 1
edit.ejs	1 Feb 2015 12
index.ejs	2 Feb 2015 15
new.ejs	1 Feb 2015 07
show.ejs	1 Feb 2015 08



GIT

3. GITHUB

GITHUB

◆ Portal de repositorios GIT (<https://github.com>)

- Enfoque social y colaborativo -> “social coding”
 - Red social para compartir proyectos software
- Curso necesita cuenta en GITHUB
 - Alberga proyectos de la asignatura

◆ Repositorios **públicos son gratis**, los privados de pago

- Repositorios totales: +20M (Linux, Eclipse, jQuery, RoR, ...)

◆ Gestión de organizaciones y proyectos software

- Soporta equipos de desarrollo distribuidos, abiertos o privados
- Uso y acceso muy sencillo a versiones, tareas, bugs, ...
- Herramientas para desktop (MAC y Windows)
- Incluye muy buenos tutoriales
-

GITHUB: registro y ayuda

- # Lo primero es crear una cuenta y una vez creada, debemos seguir sus instrucciones para
- # -> <https://help.github.com/articles/set-up-git/>
- # 1) Configurar y conectar con GITHUB nuestro GIT local
- # 2) Instrucciones para crear y clonar repositorios
- # 3) Instrucciones para colaborar en proyectos software distribuidos

Click aquí: Instrucciones

GitHub Bootcamp

- 1 Set up Git**
A quick guide to help you get started with Git.
- 2 Create repositories**
Repositories are where you'll work and collaborate on projects.
- 3 Fork repositories**
Forking creates a new, unique project from an existing one.
- 4 Work together**
Send pull requests, follow friends. Star and watch projects.

Repositorio: <https://github.com/jquemada/quiz-2015>
Puede clonarse con:
`..$ git clone https://github.com/jquemada/quiz-2015`

seguir proyecto

clonar en mi cuenta

branches (ramas)

releases

Commits (versiones)

código: directorios, ficheros, ..

colaboradores

contribuciones

clonar o bajar como ZIP

Proyecto de CORE 2015 y @NodeMOOC — Edit

19 commits 2 branches 0 releases 2 contributors

branch: master quiz-2015 / +

Soporte HTTPS

jquemada authored 10 days ago

bin

certs

controllers

models

public

routes

views

.gitignore

Procfile

app.js

commands_to_generate_keys.txt

package.json

Soporte HTTPS 8 days ago

Soporte HTTPS 8 days ago

Moderación de comentarios 10 days ago

Moderación de 10 days ago

Autenticación de usuarios 10 days ago

Moderación de comentarios 10 days ago

Moderación de comentarios 10 days ago

Despliegue DB en Heroku 2 months ago

Despliegue en Heroku ago

Autenticación de usuarios ago

Soporte HTTPS ago

Autenticación de usuarios 10 days ago

Issues 0

Pull Requests 0

Wiki

Pulse

Graphs

HTTPS clone URL

<https://github.com/jquemada/quiz-2015>

You can clone with HTTPS, SSH, or Subversion.

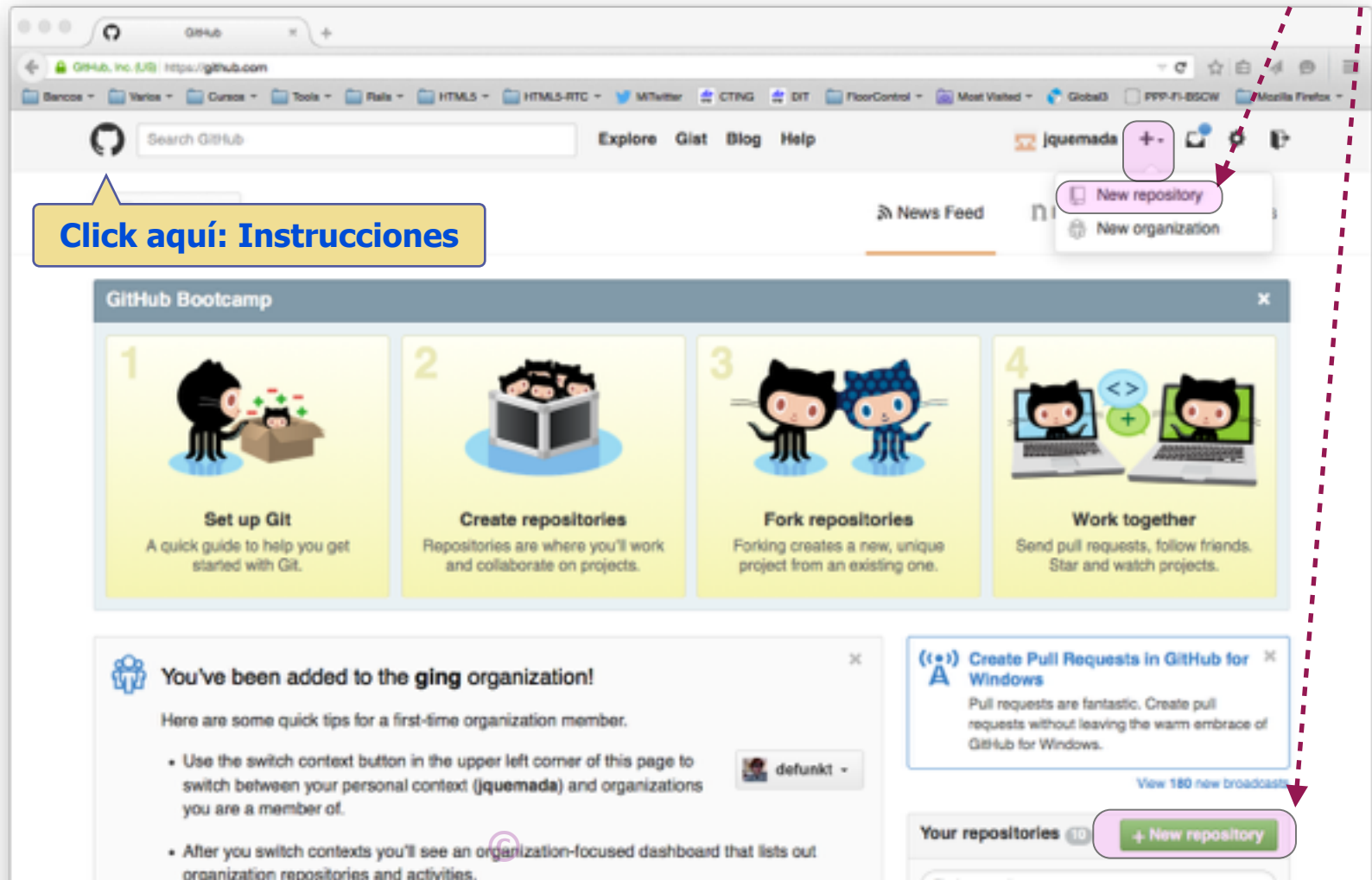
Clone in Desktop

Download ZIP

Subir un repositorio local a GITHUB

- # Para subir un repositorio local a GITHUB debemos
- # 1) Crear un repositorio vacío en GITHUB con **New Repository**
- # 2) Configurar repo. remoto origin con repositorio vacío
- \$ `git remote add origin https://github.com/pepe/proy1`
- # 3) Hacer **push** de rama master local a origin
- \$ `git push -u origin master` # **-u "tracking reference"**

Click aquí: Instrucciones



Fork: Copiar un proyecto en GITHUB

Fork permite copiar un repositorio (proyecto) en nuestra cuenta en GITHUB

-> Una vez copiado (clonado) tenemos acceso a él y podemos evolucionarlo

The screenshot shows the GitHub interface for the jQuery repository. At the top, there's a navigation bar with links like 'Explore', 'Gist', 'Blog', and 'Help'. Below this, the repository name 'jquery / jquery' is displayed. To the right of the name are buttons for 'Watch' (2,892), 'Star' (34,002), and 'Fork' (8,200). A red dashed arrow points from the text in the box above to the 'Fork' button. Below the repository name, there's a section for repository statistics: '5,784 commits', '8 branches', '125 releases', and '216 contributors'. A table of recent commits is visible, with columns for commit type, description, and time ago. On the right side, there's a sidebar with links to 'Code', 'Issues' (101), 'Pull requests' (12), and 'Wiki'. At the bottom right, there's a section for cloning the repository, showing the HTTPS clone URL and buttons for 'Clone in Desktop' and 'Download ZIP'.

jQuery JavaScript Library <http://jquery.com/>

5,784 commits 8 branches 125 releases 216 contributors

branch: master jquery / +

Docs: Add info about Sizzle not being excludable on the compat branch

mzgol authored 8 days ago latest commit @62b5267d0

build	Deferred: Backwards-compatible standards interoperability	18 days ago
external	Build: Update native-promise-only (again)	10 days ago
src	Attributes: revert returning null for non-existant attributes	8 days ago
test	Attributes: revert returning null for non-existant attributes	8 days ago
.editorconfig	Misc: Need for speed removed by 9ad6e7e	6 months ago
.gitattributes	Build: change .gitattributes; use system line ends for non-JS files	a year ago
.gitignore	Build: drop bower; use npm for front-end deps	9 months ago
.jscsrc	Build: Move all external libraries to external directory	10 months ago
.jshintignore	Build: Move all external libraries to external directory	10 months ago

Code

Issues 101

Pull requests 12

Wiki

Pulse

Graphs

HTTPS clone URL

<https://github.com/jquery/jquery>

You can clone with [HTTPS](#), [SSH](#), or [Subversion](#).

Clone in Desktop

Download ZIP

Contribuir a un proyecto GITHUB

La forma habitual de contribuir a un proyecto en GITHUB es seguir estos 4 pasos:

1) Crear una copia del repositorio original en GITHUB con "Fork" en la cuenta propia

2) Clonar la rama creada en nuestra cuenta en nuestro ordenador local

```
p1> git clone https://github.com/pepe/proy1
```

3) Modificar el proyecto local, realizar commit y "push" a nuestra copia en GITHUB

```
p1> .....
```

```
p1> git add ...
```

```
p1> git commit -m `.....`
```

```
p1> git push origin master
```

4) Hacer "Pull Request" desde nuestra cuenta en GITHUB pidiendo al administrador del repositorio original que introduzca nuestros cambios

github SOCIAL CODING

jqemada Dashboard Inbox 1 Account Settings Log Out

Explore GitHub Gist Blog Help Search...

lifo / docrails

Watch Fork Pull Request 585 230

Code Network Pull Requests 0 Wiki 5 Stats & Graphs

PLEASE CHECK <http://github.com/lifo/docrails/wikis> — Read more

<http://weblog.rubyonrails.org/2008/5/2/help-improve-rails-documentation-on-git-branch>

Clone in Mac ZIP SSH HTTP Git Read-Only git@github.com:lifo/docrails.git Read+Write access

Files Commits Branches 1 Tags Downloads

Current branch: master

Latest commit to the master branch



GIT

4. Crear proyecto "random" con GITHUB App

Crear un proyecto con GITHUB App

GITHUB-for-MAC/Windows es una herramienta gráfica muy eficaz para gestionar proyectos git localmente en el PC. Solo está soportada para MAC y para Windows:

Descargar (e instalar) GITHUB for MAC: <https://mac.github.com>

Descargar (e instalar) GITHUB for Windows: <https://windows.github.com>

En este ejemplo vamos a crear desde cero un proyecto, de nombre "random", con 2 versiones de los ejemplos que generan números aleatorios, usados para ilustrar la sentencia if/else. Este proyecto se puede encontrar en GITHUB en: <https://github.com/jquemada/random>

Desplegable para:

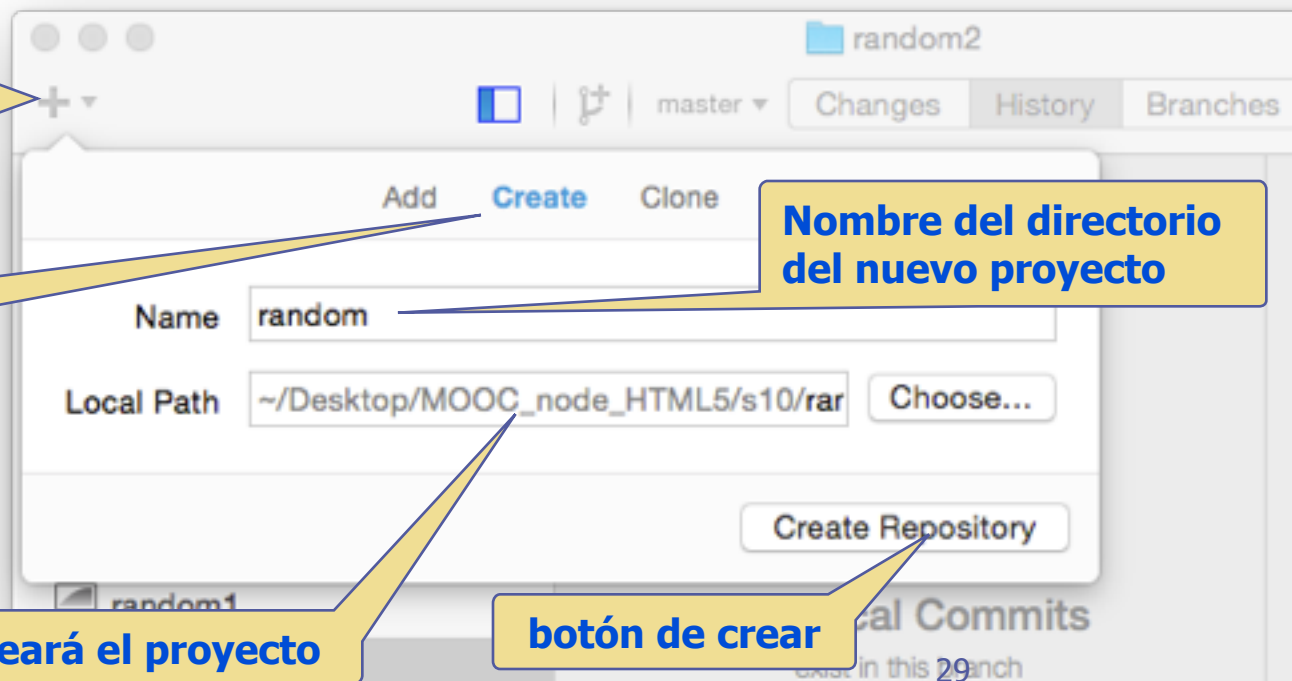
- Añadir (Add) proyecto
- Crear (Create) proyecto
- Clonar (Clone) proyecto

Pestaña de crear nuevo proyecto

Nombre del directorio del nuevo proyecto

directorio donde se creará el proyecto

botón de crear



Crear ficheros del proyecto con Atom

The image shows the Atom editor interface with a project named 'random' open. A context menu is displayed over the project files, and a 'File' menu is open in the top right corner. Red arrows point from text boxes to specific elements in the interface.

Nuevo proyecto creado

Atom es un editor muy sencillo y eficaz para editar los ficheros del proyecto. Una vez instalado Atom, podemos abrirlo y crear nuevos ficheros con "New File".

Click con Botón derecho del ratón (^Click) en proyecto despliega esta ventana de acceso

File	Edit	Selection	Find
New Window		⇧⌘N	
New File		⌘N	
Open...		⌘O	
Reopen Last Item		⇧⌘T	
Save		⌘S	
Save As...		⇧⌘S	
Save All		⌘⌘S	
Close Tab		⌘W	
Close Pane			
Close Window		⇧⌘W	

Crear/Editar fichero random.js

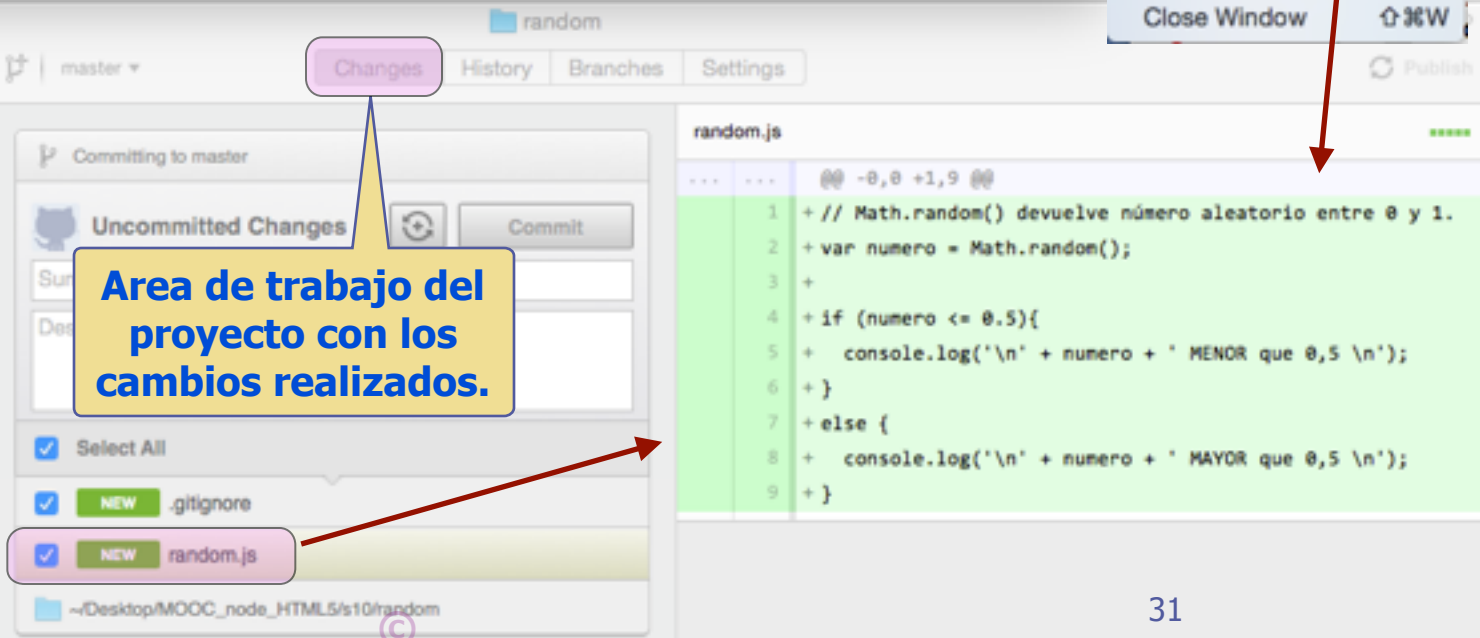


```
1 // Math.random() devuelve número aleatorio entre 0 y 1.
2 var numero = Math.random();
3
4 if (numero <= 0.5){
5   console.log('\n' + numero + ' MENOR que 0,5 \n');
6 }
7 else {
8   console.log('\n' + numero + ' MAYOR que 0,5 \n');
9 }
10
```

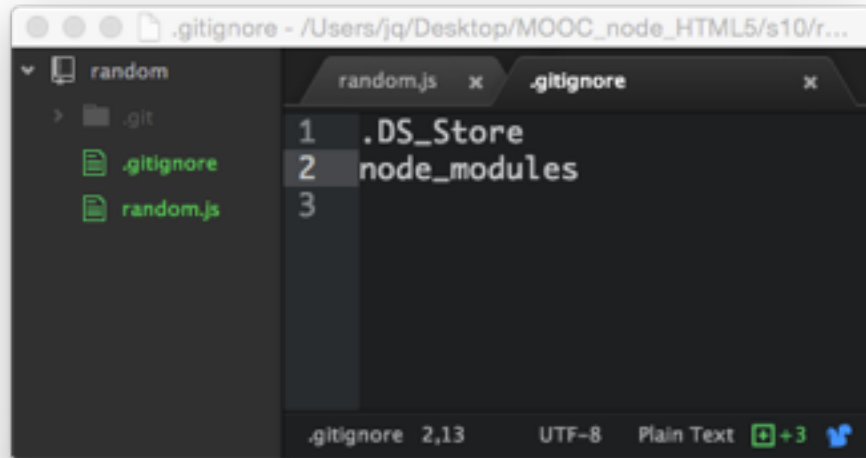
File	Edit	Selection	Find
New Window			⇧⌘N
New File			⌘N
Open...			⌘O
Reopen Last Item			⇧⌘T
Save			⌘S
Save As...			⇧⌘S
Save All			⇧⌘⌘S
Close Tab			⌘W
Close Pane			
Close Window			⇧⌘W

Primer ejemplo de número aleatorio con Math.random() e if/else, que se guarda en la versión 1 del proy random.

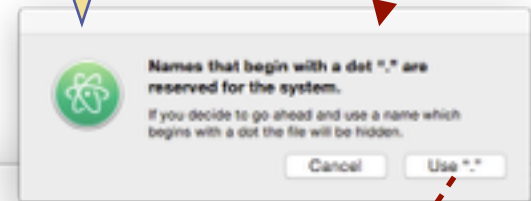
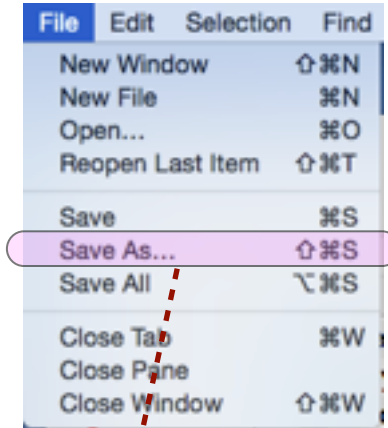
Area de trabajo del proyecto con los cambios realizados.



Crear/Editar fichero .gitignore



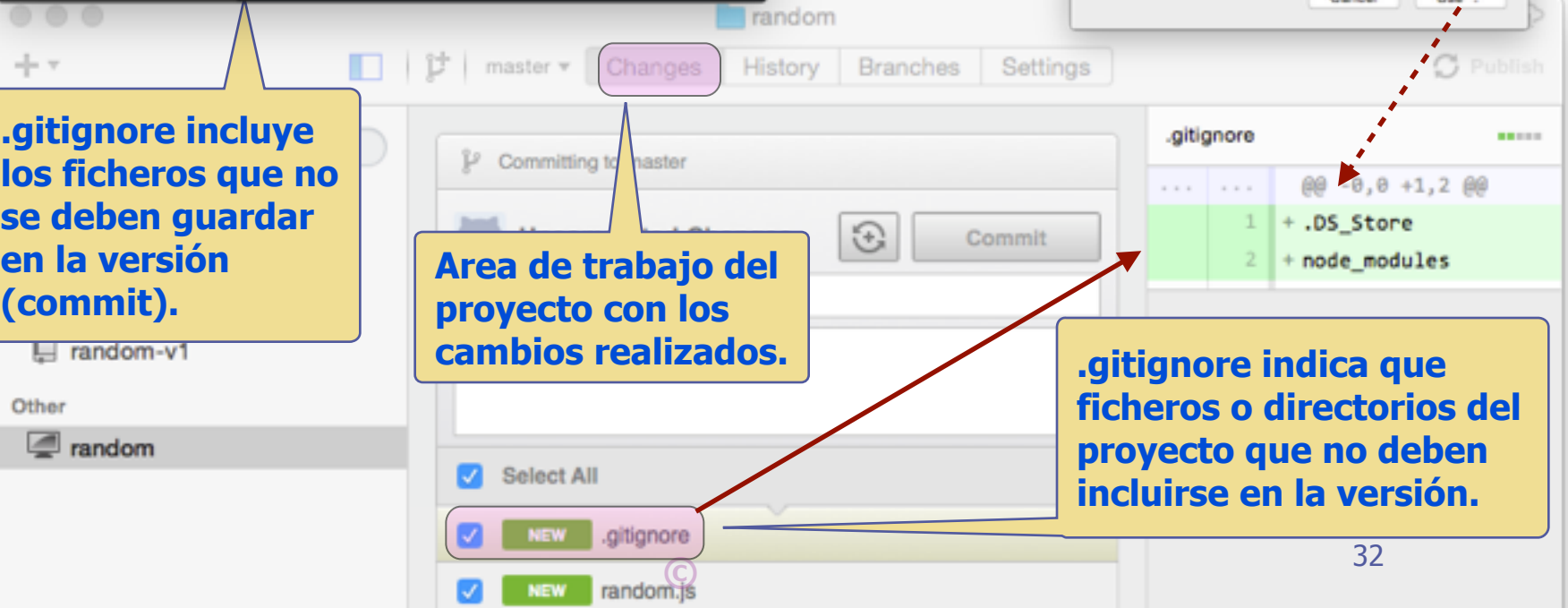
Como .gitignore es un fichero del sistema, al guardarlo nos pide confirmación.



.gitignore incluye los ficheros que no se deben guardar en la versión (commit).

Area de trabajo del proyecto con los cambios realizados.

.gitignore indica que ficheros o directorios del proyecto que no deben incluirse en la versión.



Crear version 1 (commit 1)

La **versión (commit)** se genera en el **proyecto local** guardado en el **directorio de nuestro ordenador** que se creó cuando se creó el proyecto.

El área de cambios se vacía al generar versión. Los cambios de la nueva versión aparecen al modificar ficheros.

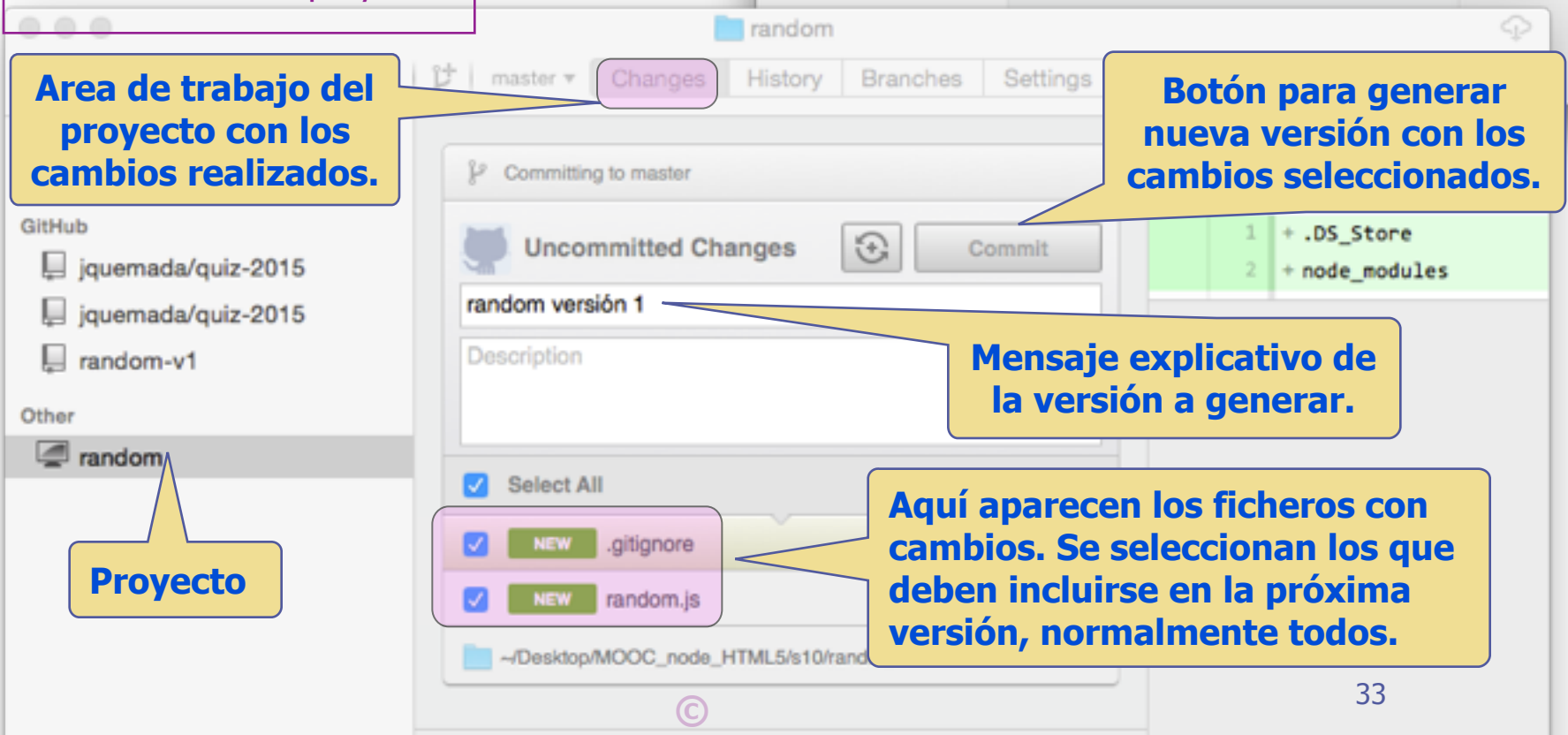
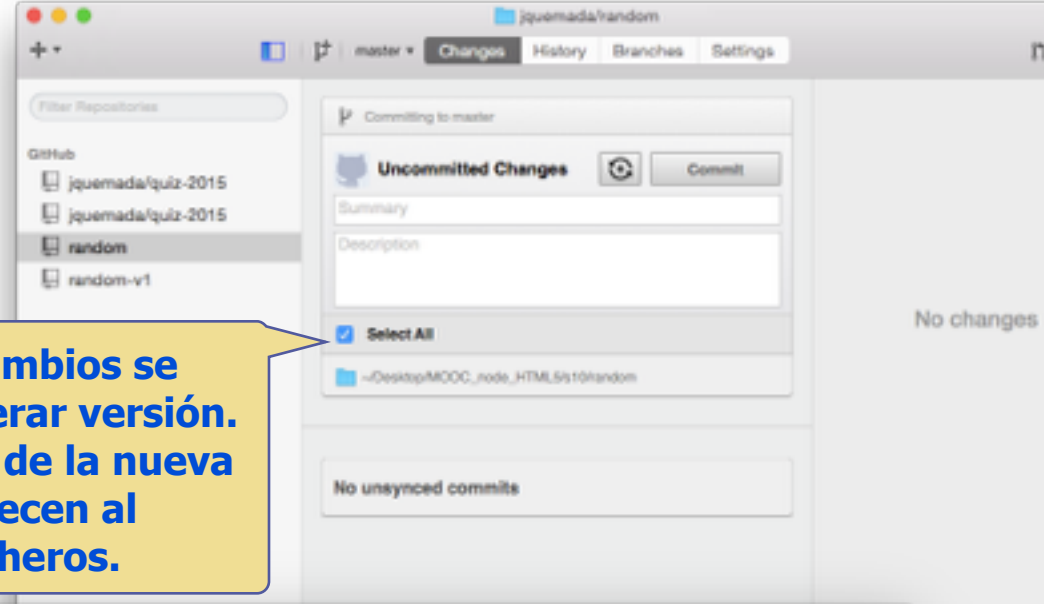
Área de trabajo del proyecto con los cambios realizados.

Botón para generar nueva versión con los cambios seleccionados.

Mensaje explicativo de la versión a generar.

Aquí aparecen los ficheros con cambios. Se seleccionan los que deben incluirse en la próxima versión, normalmente todos.

Proyecto



Definir repositorio remoto "origin"

Una **cuenta en GITHUB** con credenciales de acceso nos permite asignar al **repositorio remoto primario "origin"** un repositorio creado **GITHUB**. Para ello asignamos el **URL de un repositorio vacío creado en GITHUB** al repositorio primario remoto **"origin"**.

Los URLs de repositorios en otras cuentas serán diferentes:
<https://github.com/jquemada/random> pertenece a "jqumada".

Configurar repositorio origin y .gitignore (se configura en pestaña "Repository" también).

Botón para configurar "origin"

Proyecto random

Primary remote repository (origin)

<https://github.com/jquemada/random>

Update Remote

"origin" es un repositorio remoto en GITHUB que identificamos por un URL (<https://github.com/jquemada/random>) donde guardar/publicar el proyecto.

Ignored files (.gitignore)

.DS_Store
node_modules

.gitignore se puede configurar aquí también. Es una forma alternativa a la ya mostrada.

Publicar rama "master" en "origin"

Un proyecto se realiza siempre en una **rama de desarrollo**, donde se **guardan las versiones que se generan**. La **rama "master"** se crea por defecto al crear un proyecto y existe siempre. Este ejemplo **guarda todo en la rama master**. Mas adelante se ve como gestionar otras ramas.

The screenshot shows the GitHub Desktop application window for a repository named 'jqemada/random'. The 'Branches' tab is selected in the top bar. The sidebar on the left shows a list of repositories and branches, with 'random' selected. The main area displays the 'Current branch' section, which shows the 'master' branch with commit hash '31727a2', author 'Juan Quemada', and message 'random versión 1'. A 'Published' button is visible next to the branch information. Callouts provide additional context: one points to the 'Branches' tab, another points to the branch list, and a third points to the 'Published' button.

Area de gestión de ramas

En este área se ve el estado de las ramas del proyecto. En este ejemplo solo tenemos la rama "master" que se creo por defecto al crear el proyecto.

Botón para publicar "origin" en GITHUB

Historia de la rama master del proyecto

La historia de cada rama de un proyecto es la **secuencia de versiones generadas**. Seleccionando una versión se visualizan los **cambios realizados en una versión respecto a la anterior**: el **código añadido se muestra en verde** y el **código eliminado se muestra en rojo**.

Rama master seleccionada

Area de gestión de historia del proyecto

Solo se ha generado la primera versión, identificada por el mensaje asociado.

Cambios de esta versión respecto a la anterior. Al ser la primera versión el código de los 2 ficheros creados es verde.

1 commit

random versión 1
7 minutes ago by jqemada

random versión 1
jqemada · 31727a2 · 7 minutes ago

.gitignore

```
@@ -0,0 +1,2 @@  
1 + .DS_Store  
2 + node_modules
```

random.js

```
@@ -0,0 +1,9 @@  
1 + // Math.random() devuelve número aleatorio entre 0 y 1.  
2 + var numero = Math.random();  
3 +  
4 + if (numero <= 0.5){  
5 +   console.log('\n' + numero + ' MENOR que 0,5 \n');  
6 + }  
7 + else {  
8 +   console.log('\n' + numero + ' MAYOR que 0,5 \n');  
9 + }
```

Crear nueva versión

Segundo ejemplo de número aleatorio con `Math.random()` e if/else.

1. Abrir proyecto en Atom, seleccionar ficheros y hacer los cambios.

Area de trabajo

3. Generar versión

2. Seleccionar cambios e introducir mensaje

Código añadido en verde, código eliminado en rojo y resto en blanco.

File	Edit	Selection	Find
New Window			⇧⌘N
New File			⇧⌘N
Open...			⇧⌘O
Reopen Last Item			⇧⌘T
Save			⌘S
Save As...			⇧⌘S
Save All			⇧⌘S
Close Tab			⌘W
Close Pane			
Close Window			⇧⌘W

random.js - /Users/jq/Desktop/MOOC_node_HTML5/s10/random - Atom

```
1 // Math.random() devuelve número aleatorio entre 0 y 1.
2 var numero = Math.random();
3
4 var str = ' MAYOR que 0,5';
5
6 if (numero <= 0.5){
7   str = ' MENOR que 0,5';
8 }
9
10 console.log('\n' + numero + str + '\n');
11
```

random.js 10,41

UTF-8

JavaScript

master

+5, -4

Changes

Uncommitted Changes

Commit

random versión 2

Description

☒ Select All

☒ random.js

View on GitHub
Open in Finder
Open in Terminal
Open in Atom

Remove

No unsynced commits

Historia de la rama master del proyecto

La historia de la rama master muestra ahora las 2 versiones generadas. Como se ha seleccionado la última, ahora se muestran los cambios respecto a la primera: el **código añadido se muestra en verde** y el **código eliminado se muestra en rojo**.

The screenshot shows a Git client interface with the following elements:

- Top Bar:** Includes a dropdown menu showing 'master', tabs for 'Changes', 'History', 'Branches', and 'Settings', and a 'Sync' button.
- Left Panel:** A sidebar with a 'Filter Repositories' search bar and a list of repositories: 'jqemada/quiz-2015', 'jqemada/quiz-2015', 'random' (selected), and 'random-v1'.
- Commit History:** A list of commits under the heading '2 commits'. The first commit is 'random versión 2' by 'jqemada' (Just now), which is selected with a blue bar and a radio button. The second commit is 'random versión 1' by 'jqemada' (1 hour ago).
- Diff View:** The right pane shows the diff for 'random versión 2'. It displays the file 'random.js' with line-by-line changes. Lines 4, 5, 7, 8, 9, and 10 are highlighted in green, indicating code added. Lines 5, 6, 7, and 8 are highlighted in red, indicating code removed. The diff shows the addition of a string 'MAYOR que 0,5' and the removal of a string 'MENOR que 0,5'.

Annotations:

- Rama master seleccionada:** Points to the 'master' branch dropdown in the top bar.
- historia del proyecto:** Points to the 'History' tab in the top bar.
- Sincronizar "origin":** Points to the 'Sync' button in the top bar.
- Se selecciona la versión 2 para mostrar cambios con versión anterior:** Points to the selected commit 'random versión 2' in the commit history.
- Aparecen 2 versiones en la historia:** Points to the list of commits in the history panel.

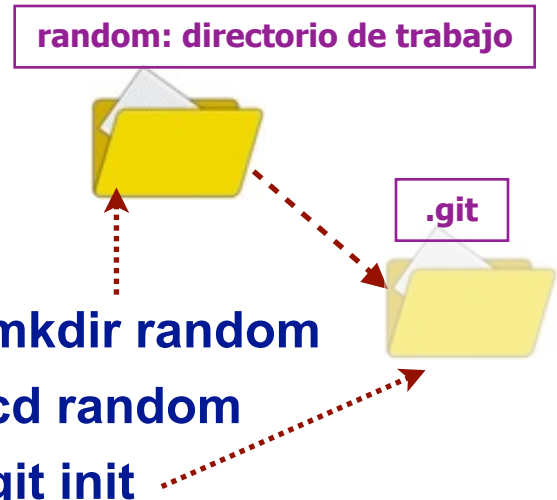


GIT

5. Crear proyecto "random" con comandos

Crear proyecto por comando

- ◆ Paso 1. Crear el directorio del proyecto -> **mkdir random**
- ◆ Paso 2. Entrar en directorio del proyecto -> **cd random**
- ◆ Paso 3. Inicializar repositorio git en el directorio -> **git init**
- ◆ “**git init**” habilita un directorio como repositorio de un proyecto
 - Los comandos git deberán ejecutarse en el directorio
- ◆ “**git init**” crea el subdirectorio oculto **.git** con el repositorio
 - El repositorio contendrá las versiones guardadas en un proyecto



```
random — bash — 62x6
venus:~ jq$
venus:~ jq$ mkdir random
venus:~ jq$ cd random
venus:random jq$ git init
Initialized empty Git repository in /Users/jq/random/.git/
venus:random jq$
```


El directorio de trabajo

◆ Directorio de trabajo (working directory)

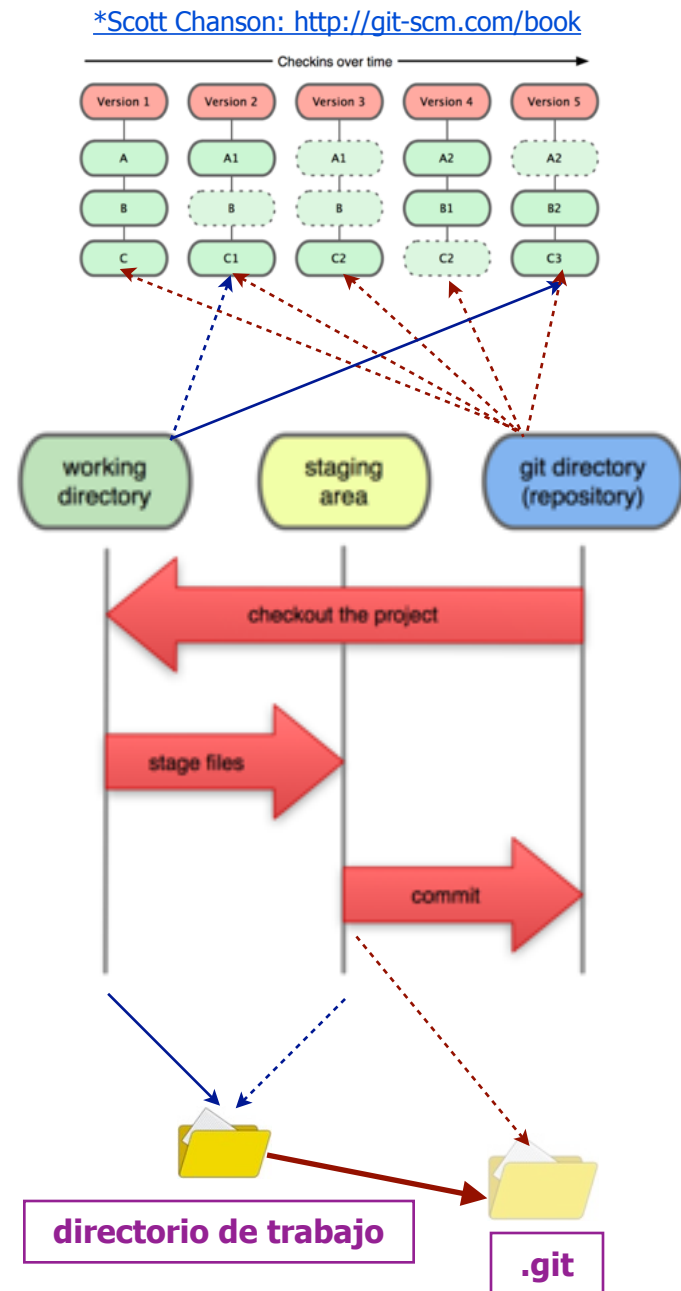
- Contiene todos los ficheros del proyecto
 - El contenido del directorio cambia a medida que el proyecto avanza

◆ Área de cambios o índice (staging area, index)

- Ficheros indexados para la próxima versión
 - Serán **ficheros borrados, nuevos o modificados** respecto a la **versión anterior**
 - "git add" añade al índice
 - "git commit" crea versión
 - **OJO!** Un fichero **modificado pero no indexado** no se incluirá en la versión

◆ Repositorio GIT (Directorio oculto ".git")

- Contiene todas las versiones del proyecto
 - "git checkout"
 - reconstruye (descongela) versiones del proyecto en el directorio de trabajo (working directory)



.gitignore

.gitignore es un fichero que informa de los ficheros que no debe gestionar GIT.
- **git status** no los presentará como ficheros untracked.
- **git add .** no los añadirá al staging area.

Los ficheros **.gitignore** pueden crearse en cualquier directorio del proyecto,
y afectan a ese directorio y a sus subdirectorios.

Su contenido: líneas con patrones de nombres.

- Puede usarse los comodines ***** y **?**

- Patrones terminados en **/** indican directorios

- Un patron que empiece con **!** indica negación

- Se ignoran líneas en blanco y que comiencen con **#**

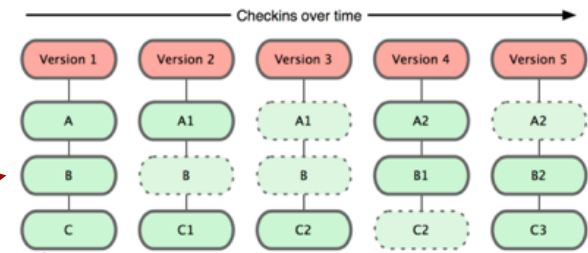
- **[abc]** indica cualquiera de los caracteres entre corchetes

- **[a-z]** indica cualquiera de los caracteres en el rango especificado

Ejemplo

private.txt	# excluir los ficheros con nombre "private.txt"
*.class	# excluir los ficheros acabados en ".class"
*.[oa]	# excluir ficheros acabados en ".o" y ".a"
!lib.a	# no excluir el fichero "lib.a"
*~	# excluir ficheros acabados en "~"
testing/	# excluir directorio "testing"

Creación de un proyecto y sus versiones



```
$ git init # Se inicia proyecto, creando repositorio vacío en .git/
$ .....
$ git add random.js # añade fichero random.js al índice
$ .....
$ git add . # Crear .gitignore con ficheros no indexados
$ git commit -m 'random versión 1' # añade resto de cambios a índice
# congela 1a versión

$ ..... # se modifica random.js
$ git add . # añade cambios a índice
$ git commit -m 'random versión 2' # congela 2a versión

# Creamos una cuenta en GITHUB y un repositorio vacío random para subir el proyecto

# Asociamos "origin" a repositorio remoto en GITHUB https://github.com/jquemada/random
$ git remote add origin https://github.com/jquemada/random
$ git push origin master # subimos la rama "master" a repositorio remoto "origin"
.....
# Clonamos repositorio remoto https://github.com/jquemada/random en directorio random-2
$ git clone https://github.com/jquemada/random random-2

$ cp -r random random-3 # random puede copiarse. random-3 sera otro repo. independiente
```

Crear nuevas versiones: add y commit

◆ git add

- añade fichero(s) al índice para próxima versión
 - **git add .** -> añade todo lo modificado al índice
 - **git add file_1.js file_2.js** -> añade solo ficheros file1.js y file2.js
 - **Ojo:** cambios posteriores a invocar “**git add ..**” no se añaden al índice

◆ git commit -m “mensaje”

- crea nueva versión en la rama actual, incluye lo registrado en el índice
 - **-m “mensaje”** incluye un mensaje que identifica la versión

Ayuda en línea de comandos

\$ git init --help # muestra ayuda en línea (manual) de “git init”

.....

\$ git add --help # muestra ayuda en línea (manual) de “git add”

.....

\$ git commit --help # muestra ayuda en línea (manual) de “git commit”

Modificar el último "commit"

```
# Para modificar el último commit usaremos git commit --amend -m ...  
# Para cambiar el mensaje de log.  
# Para añadir una modificación olvidada  
# ...
```

```
$ git commit -m 'editor acabado' # creamos el commit pero olvidamos  
# añadir un fichero, y el mensaje de  
# log no esta en ingles
```

```
$ ..... # Realizamos los cambios olvidados  
$ git add forgotten_file # y los subimos al índice
```

```
# Repetimos git commit con opción --amend y un mensaje de log (modificado o no)  
$ git commit --amend -m "editor acabado"
```

```
# Se actualiza el commit erróneo con los nuevos cambios introducidos
```

IMPORTANTE: no realizar --amend sobre un commit que se haya hecho público a otros desarrolladores (publicado en otro repositorio).

git log: Historia de versiones

```
# La historia de versiones (commits) de de la rama en la que se está trabajando
# -> se muestra con "git log"
# "git log --stat"           # muestra estadísticas
# "git log --graph"          # muestra árbol
# "git log --since=2.weeks"  # muestra commits últimas 2 semanas
# "git log --oneline"        # muestra resumen de cada commits en 1 línea
# "git log -5"              # muestra 5 últimos commits
```

```
$ git log -2      # Muestra 2 últimos commits
commit b48cd0b84dd71d4314b11a917f2971b26b464d92
Author: Juan Quemada <jquemada@dit.upm.es>
Date: Thu Apr 2 13:13:15 2015 +0200
```

random versión 2

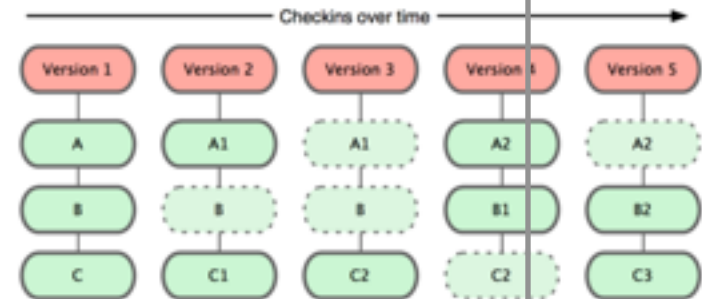
```
commit b66f1fb6c70f3f669b216fc25aac0f5ebe1542f2
Author: Juan Quemada <jquemada@dit.upm.es>
Date: Thu Apr 2 13:08:47 2015 +0200
```

random versión 1

```
$ git log --oneline    # Muestra resumen de 1 línea de commits
```

```
b48cd0b random versión 2
b66f1fb random versión 1
```

```
$
```



La historia de diferencias entre commits se muestra con opción -p, por ejemplo "git log -p -1"

```
$ git log -p -1
commit 188799e21c4a71f13a2e729ccf77f3a960885682
Author: Juan Quemada <jquemada@dit.upm.es>
Date: Mon Nov 21 18:17:13 2011 +0100
```

migración base de datos

Diferencias entre versiones

```
diff --git a/db/schema.rb b/db/schema.rb
index b5e6a79..61dcaab 100644
```

```
--- a/db/schema.rb
```

```
+++ b/db/schema.rb
```

```
@@ -11,6 +11,13 @@
```

```
#
```

```
# It's strongly recommended to check this file into your version control system.
```

```
-ActiveRecord::Schema.define(:version => 0) do
```

```
+ActiveRecord::Schema.define(:version => 20111121171513) do
```

```
+
```

```
+ create_table "types", :force => true do |t|
```

```
+   t.string "name"
```

```
+   t.text "description"
```

```
+   t.datetime "created_at"
```

```
+   t.datetime "updated_at"
```

```
+ end
```

```
end
```

```
$
```

Rama master y puntero HEAD

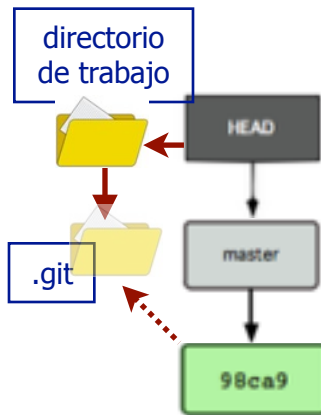
♦ **master** es la rama principal del desarrollo

- “**git init**” inicia el proyecto en la **rama master**
 - Las versiones (commits) se crean en **master** (salvo que se pase a otra rama)
- **master** es un puntero a la última versión de esta rama principal

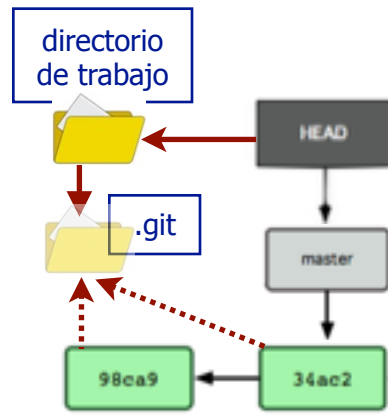
♦ HEAD referencia la versión (commit) actual del directorio de trabajo

♦ Cada “**git commit ...**” crea una nueva versión

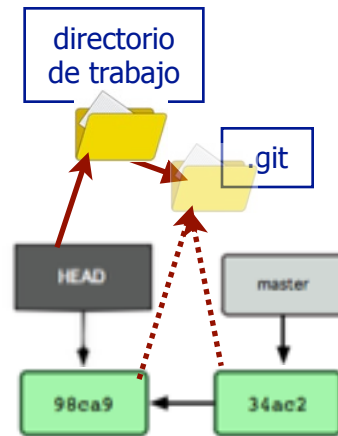
- actualizando los **punteros master y HEAD**



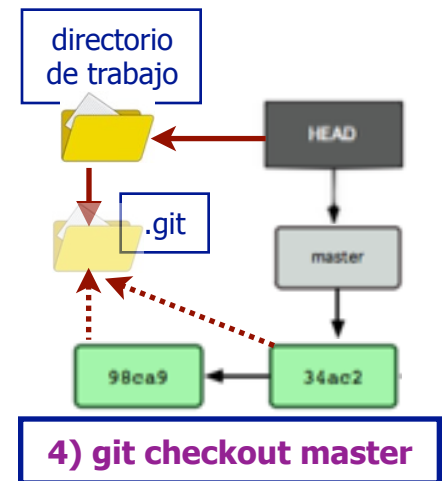
1) git commit -m “..1”



2) git commit -m “..2”



3) git checkout 98ca9



4) git checkout master

OJO! Detached head: peligroso

Reset: Eliminar commits

"git reset <commit_id>"

-> vuelve a <commit_id> eliminando versiones posteriores

\$ git log --oneline # lista commits

c2b9e migración base de datos

f30ab creación de scaffold Type

34ac2 añadir ejemplo

98ca9 vistas index y contact



\$ git reset 34ac2

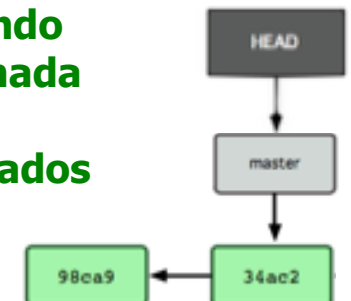
restaura versión 34ac2 'añadir ejemplo' dejando los cambios
realizados en las versiones eliminadas en directorio de trabajo
sin añadir al índice (staging-area)

\$ git reset --hard 34ac2

restaura 34ac2 'añadir ejemplo' eliminando
todos los cambios de las versiones eliminada

¡OJO! con "commit reset --hard ..." se pierden los commits eliminados

\$





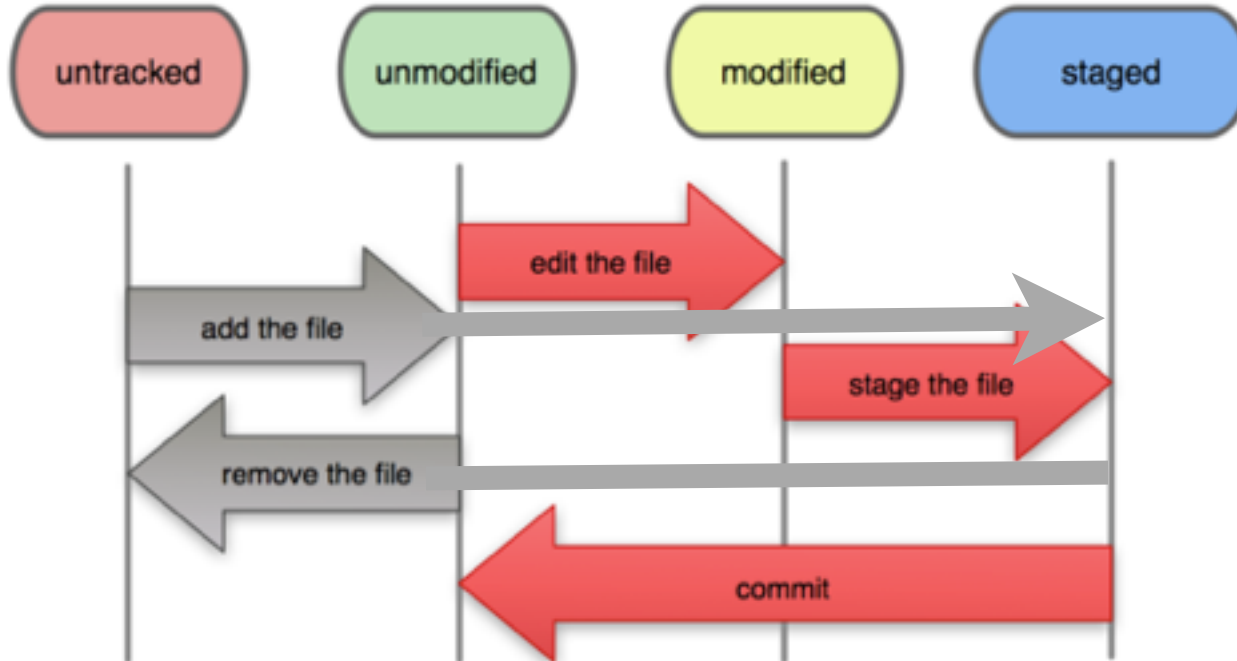
GIT

6. Análisis y gestión del área de trabajo

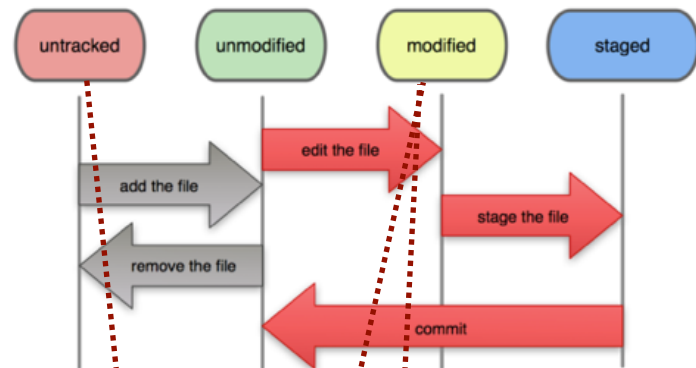
Estado de los ficheros

◆ Los ficheros del directorio de trabajo pueden estar

- **Untracked:** Ficheros que no están bajo el control de versiones
- **Tracked:** Ficheros registrados en versión (con **git add**)
 - **Modified:** ficheros modificados, no incluidos en próximo commit con **git add ...**
 - **Unmodified:** ficheros no modificados, que **siguen en próximo commit**
 - **Staged:** ficheros modificados, incluidos en próximo commit con **git add...**
- **Ignorados:** Ficheros indicados en **.gitignore**



Ánisis del estado del área de trabajo: "git status"



"git status" muestra estado del directorio de trabajo:
1) **Changes to be committed**: ficheros modificados indexados con "git add ..."
2) **Changed but not updated**: ficheros modificados no indexados con "git add ..."
3) **Untracked files**: ficheros nuevos no indexados con "git add ..." o extraídos con "git rm ..."

\$ git status

On branch master

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

#

modified: README
new file: CharIO.java
#

#

Changed but not updated:

(use "git add <file>..." to update what will be committed)

#

modified: benchmarks.rb
#

#

Untracked files:

(use "git add <file>..." to include what will be committed)

#

merge.java
library/lib.js
#

Ficheros modificados
incluidos en próxima versión

Ficheros modificados
no incluidos en
próxima versión

Ficheros excluidos
de versión

Borrar ficheros

Eliminar un fichero en la próxima versión a congelar:

\$ **git rm** CharIO.java # Borra el fichero del directorio de trabajo y del staging area.
Tras el próximo commit dejará de estar tracked.

\$ **git rm --cached** CharIO.java # Borra fichero del staging area.
No lo borra del directorio de trabajo.
Tras el próximo commit dejará de estar tracked.

El comando del S.O. **rm** borra ficheros del directorio de trabajo,
pero no los borra del staging area.
Es como hacer una modificación en el contenido del fichero.
Debe usarse **git add** o **git rm** para meter en el staging area esta modificación.

\$ **rm** CharIO.java # borra el fichero de directorio de trabajo,
pero este cambio aun no ha sido staged.

git rm falla si se intenta borrar un fichero con modificaciones en el directorio
de trabajo o en el staging area (índice).
Para no perder de forma accidental modificaciones realizadas.
Usar la opción **-f** para forzar el borrado.

Renombrar ficheros

Mover o renombrar un fichero:

```
$ git mv filename_old filename_new
```

```
$ git mv index.htm index.html
```

Internamente se implementa ejecutando los comandos git rm y git add

```
$ git mv filename_old filename_new
```

es equivalente a ejecutar:

```
$ mv filename_old filename_new
```

```
$ git rm filename_old
```

```
$ git add filename_new
```

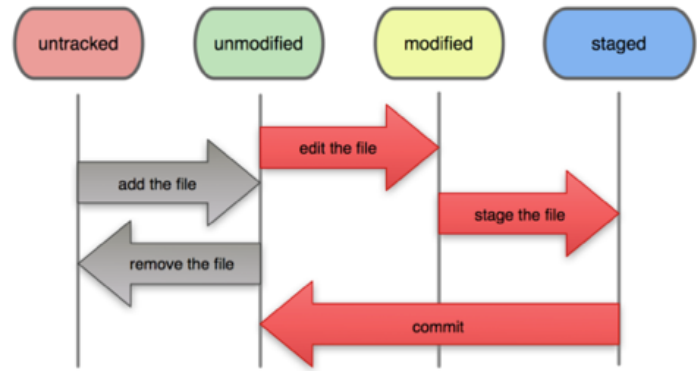
The diagram illustrates the lifecycle of a file in Git across four states: untracked (red), unmodified (green), modified (yellow), and staged (blue). The transitions are as follows:

- add the file:** Moves a file from **untracked** to **unmodified**.
- edit the file:** Moves a file from **unmodified** to **modified**.
- stage the file:** Moves a file from **modified** to **staged**.
- remove the file:** Moves a file from **unmodified** back to **untracked**.
- commit:** Moves a file from **staged** back to **unmodified**.

55

Ánalysis del estado del área de trabajo:

```
"git diff --cached"
```



"git diff --cached" o git diff --staged" muestra diferencias en ficheros modificados e indexados (staged)

\$ git diff --staged

```
diff --git a/benchmarks.rb b/benchmarks.rb
```

index 3cb747f..da65585 100644

--- a/benchmarks.rb

+++ [b/benchmarks.rb](#)

```
@@ -36,6 +36,10 @@ def main
```

rango de líneas con cambios

```
@commit.parents[0].parents[0].parents[0]
```

end

contenido no modificado, enmarca cambios

- **# -> insert new task here**

líneas eliminadas empiezan por "-"

```
+ run_code(x, 'commits 1') do
```

líneas nuevas empiezan por "+"

+ **git.commits.size**

```
+ end
```

+

```
run_code(x, 'commits 2') do
  log = git.commits('master', 15)
  log.size
end
```

contenido no modificado, enmarca cambios

Eliminar Modificaciones en el Directorio de trabajo

Para eliminar las modificaciones realizadas en un fichero del directorio de trabajo, y dejarlo igual que la version del repositorio:
git checkout -- <file>

Ejemplo:

Modificamos un fichero.

\$ vi readme.txt # editamos el contenido del fichero readme.txt

Nos arrepentimos de los cambios realizados.

Para restaurar el fichero a su estado original ejecutamos:

\$ git **checkout --** readme.txt

"git checkout ." deshace todos los cambios staged de area de trabajo

\$ git checkout .

Deshacer operaciones realizadas: Eliminar Modificaciones Staged

```
# Para eliminar del staging area las modificaciones de un fichero:  
#  git reset HEAD <file>
```

```
# Ejemplo:
```

```
# Modificamos un fichero y registramos los cambios en el staging area:
```

```
$ vi readme.txt # editamos el contenido del fichero readme.txt  
$ git add .      # añadimos todos los cambios existentes en todos los  
                  # ficheros tracked al staging area.
```

```
# Pero no queremos añadir los cambios de readme.txt.
```

```
# Para cambiar el estado de readme.txt a unstaged ejecutamos:
```

```
$ git reset HEAD readme.txt
```

```
# readme.txt ya no esta modificado en el staging area.
```

```
# readme.txt conserva sus modificaciones en el directorio de trabajo.
```

GITHUB for MAC & for Windows

GITHUB-for-MAC/Windows permite ver todos los cambios realizados a la última versión en el directorio de trabajo. Además, el editor Atom nos permite hacer cambios con mucha facilidad.

Directorio de trabajo

**Proyecto
seleccionado**

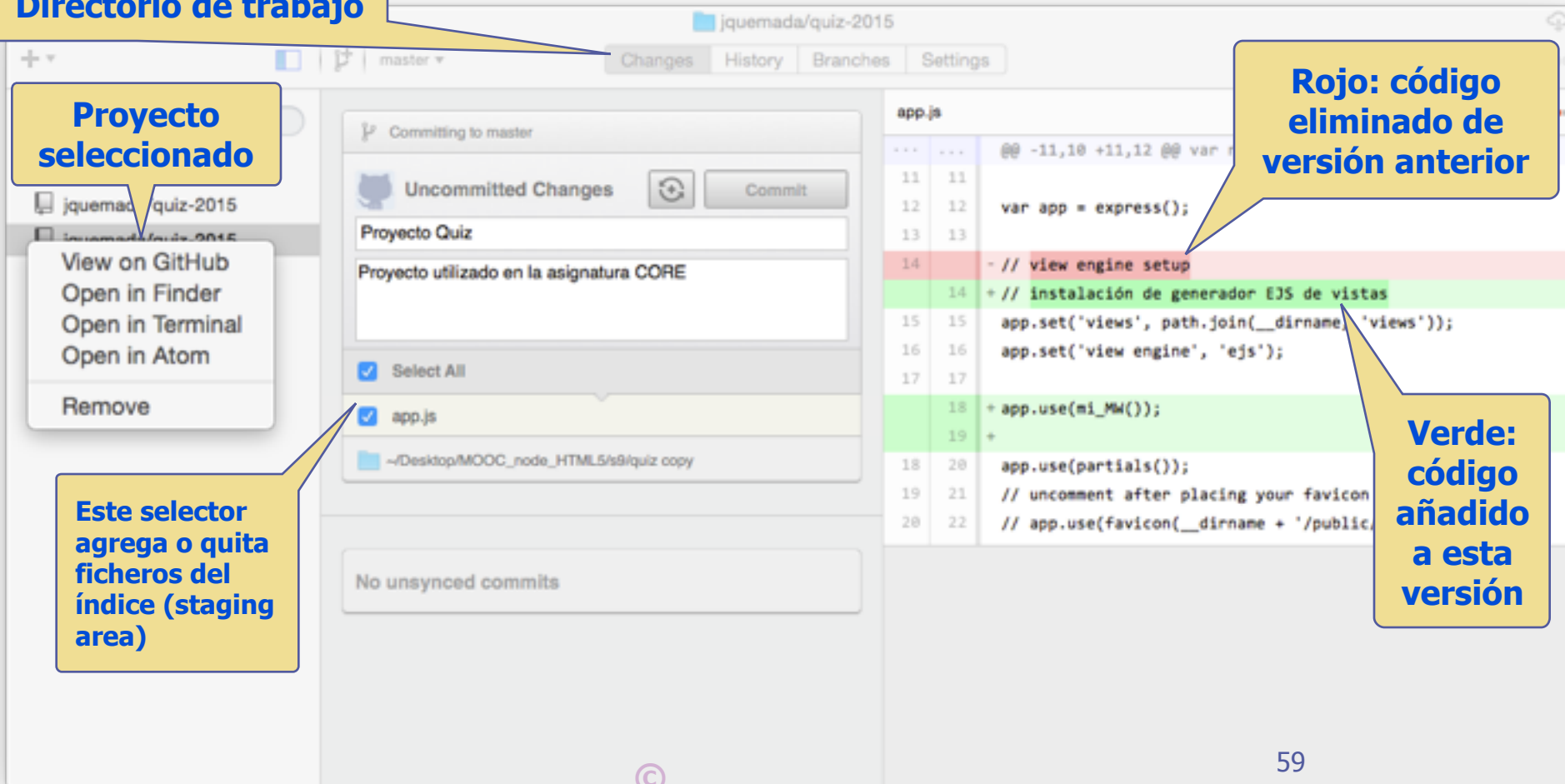
View on GitHub
Open in Finder
Open in Terminal
Open in Atom

Remove

**Este selector
agrega o quita
ficheros del
índice (staging
area)**

**Rojo: código
eliminado de
versión anterior**

**Verde:
código
añadido
a esta
versión**





GIT

7. Ramas

Crear ramas

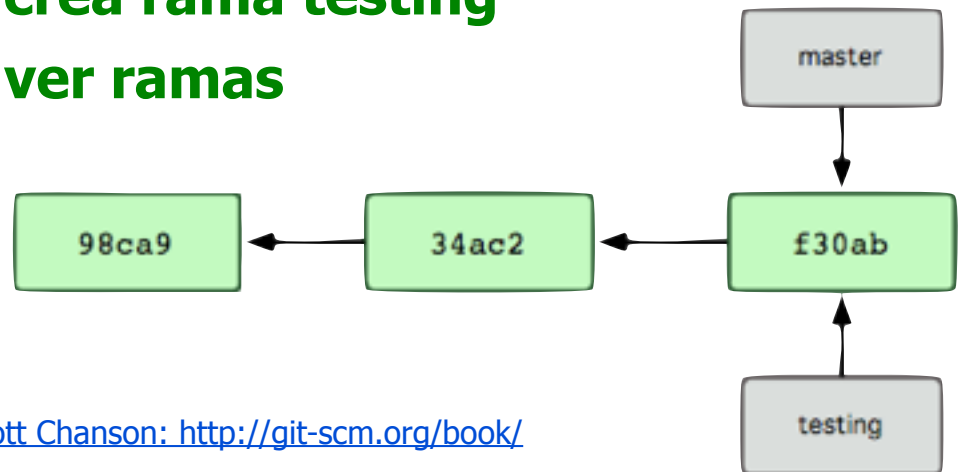
- ♦ **Rama**: desarrollo que diverge de la rama master o de otra rama
 - ★ A partir de alguna versión (commit)
- ♦ Una rama se crea con: **git branch <nombre_de_rama>**
 - ★ Crea un nuevo puntero asociado a la rama
- ♦ **git branch** permite ver las ramas creadas
 - ★ * indica cual esta seleccionada

\$ git branch testing # crea rama testing

\$ git branch # ver ramas

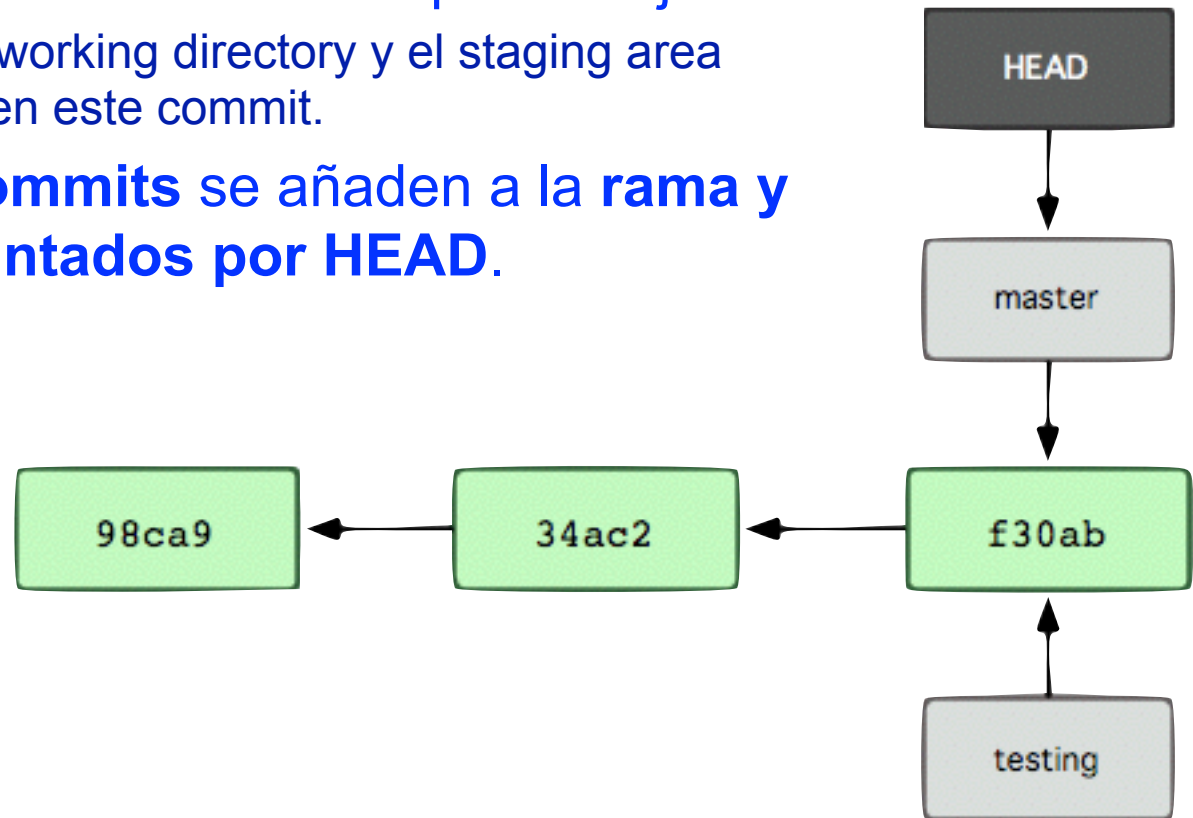
*** testing
master**

\$



**de Scott Chanson: <http://git-scm.org/book/>*

- ◆ En GIT existe una referencia llamada **HEAD** que apunta a la rama actual
 - ★ que apunta al commit sobre el que trabajamos.
 - los ficheros del working directory y el staging area están basados en este commit.
 - ★ Los **nuevos commits** se añaden a la rama y al commit apuntados por HEAD.



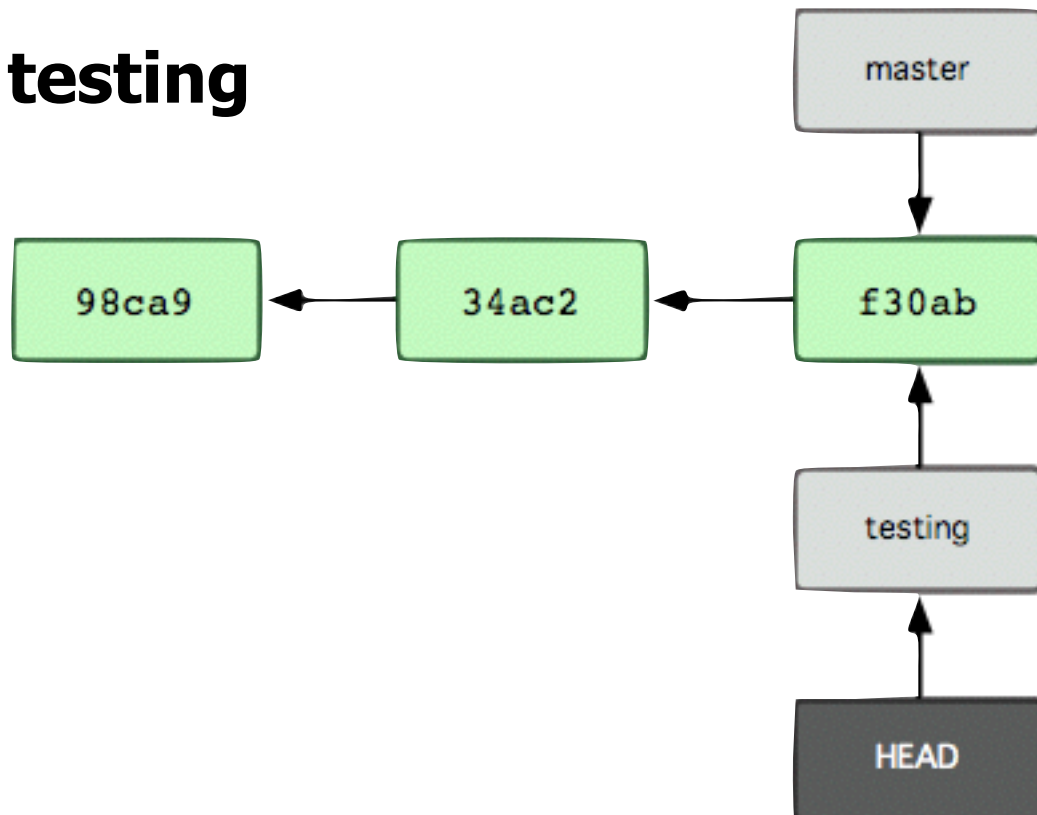
*de Scott Chanson: <http://git-scm.org/book/>

♦ Para cambiar de rama: **git checkout** <nombre_rama>

★ **HEAD** apuntará a la nueva rama.

★ Se actualizan el working directory y el staging area (índice)

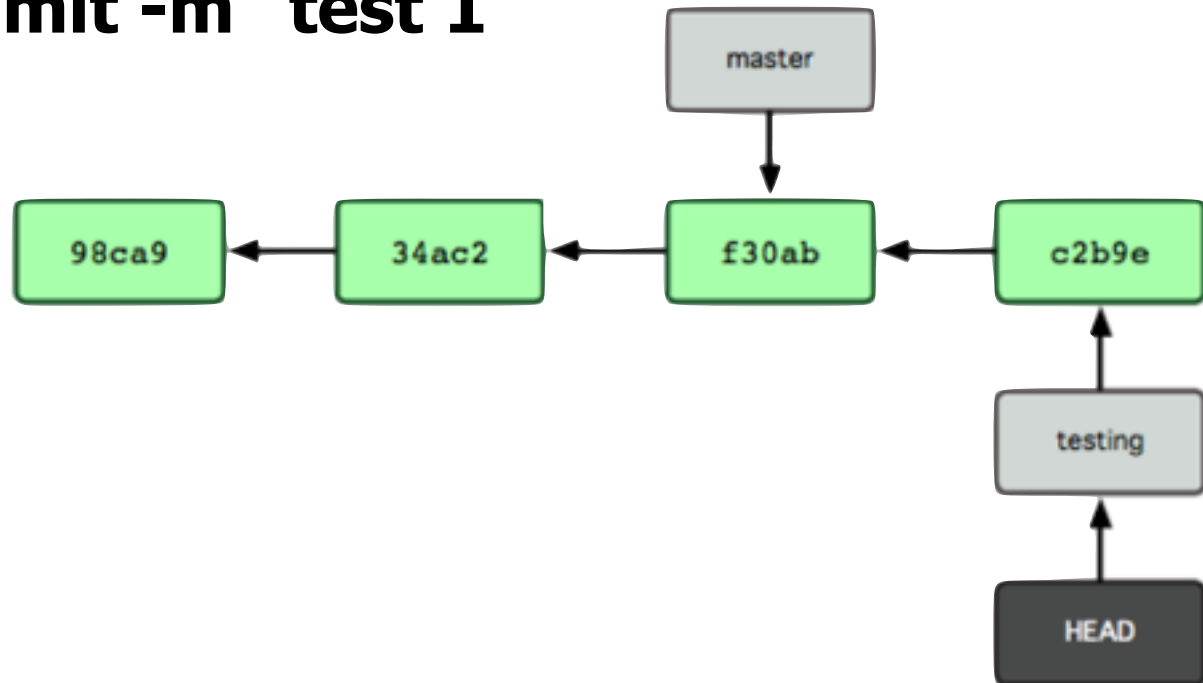
\$ git checkout testing



♦ Al hacer commit avanza la rama apuntada por HEAD.

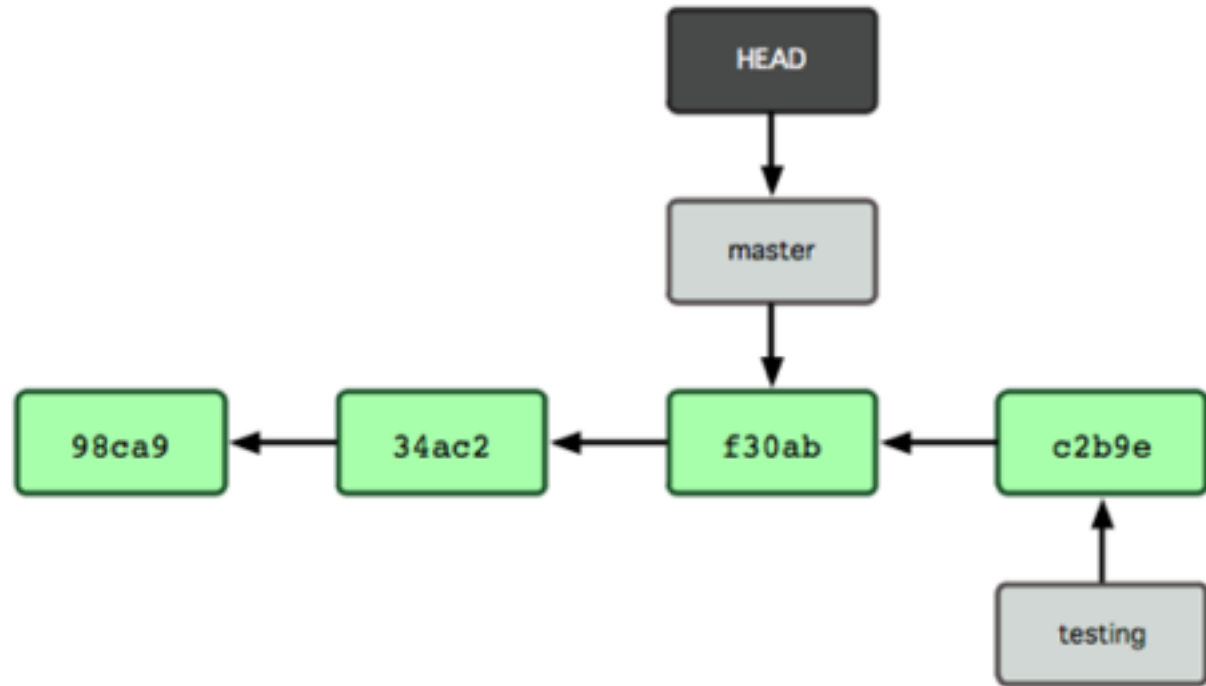
\$ git checkout testing

\$ git commit -m "test 1"



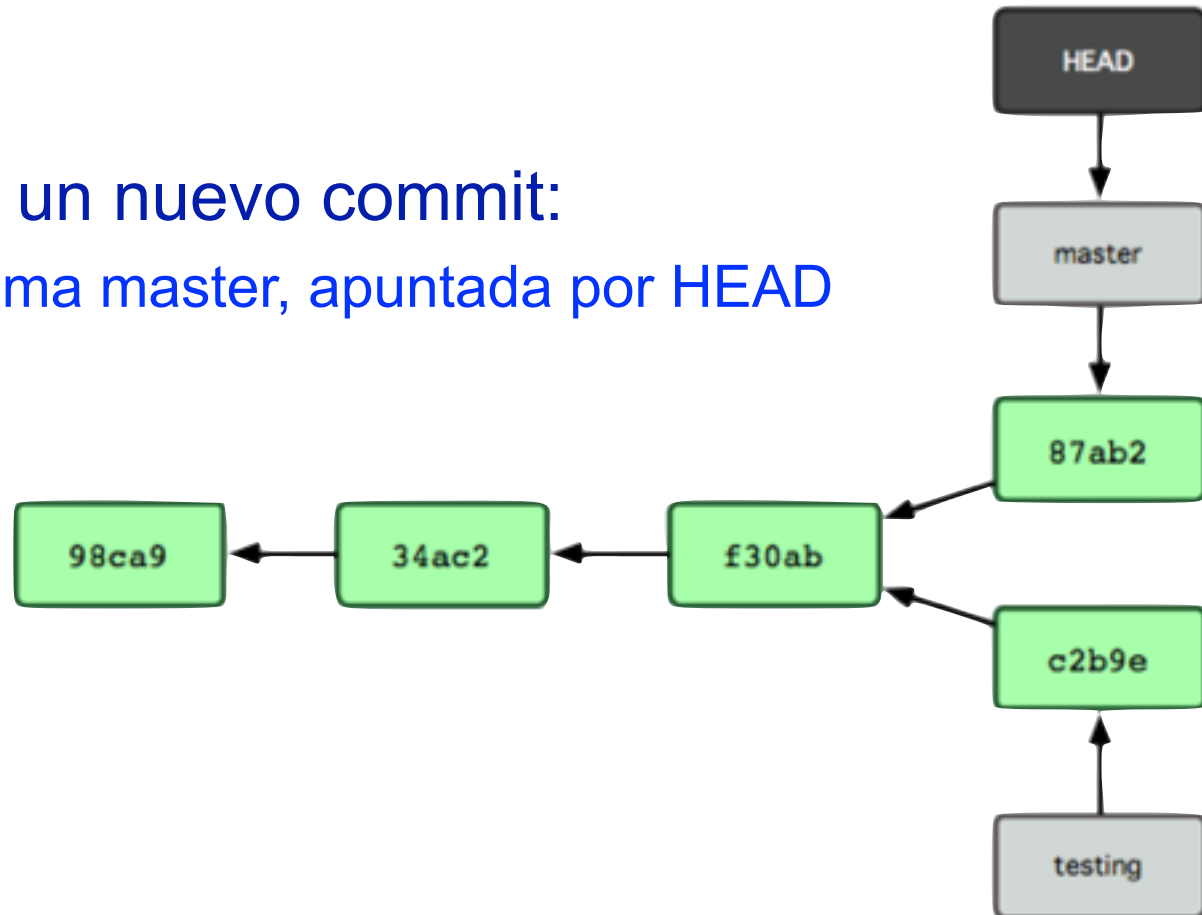
♦ Para volver a la rama master:

\$ git checkout master



◆ Si hacemos un nuevo commit:

★ avanza la rama master, apuntada por HEAD



Resumen ramas

El comando **branch** permite crear ramas

..\$ **git branch testing** # crea rama testing

..\$ **git checkout testing** # HEAD a rama testing

.....

..\$ **git add .**

..\$ **git commit -m 'test 1'** # commit de testing

..\$ **git checkout master**

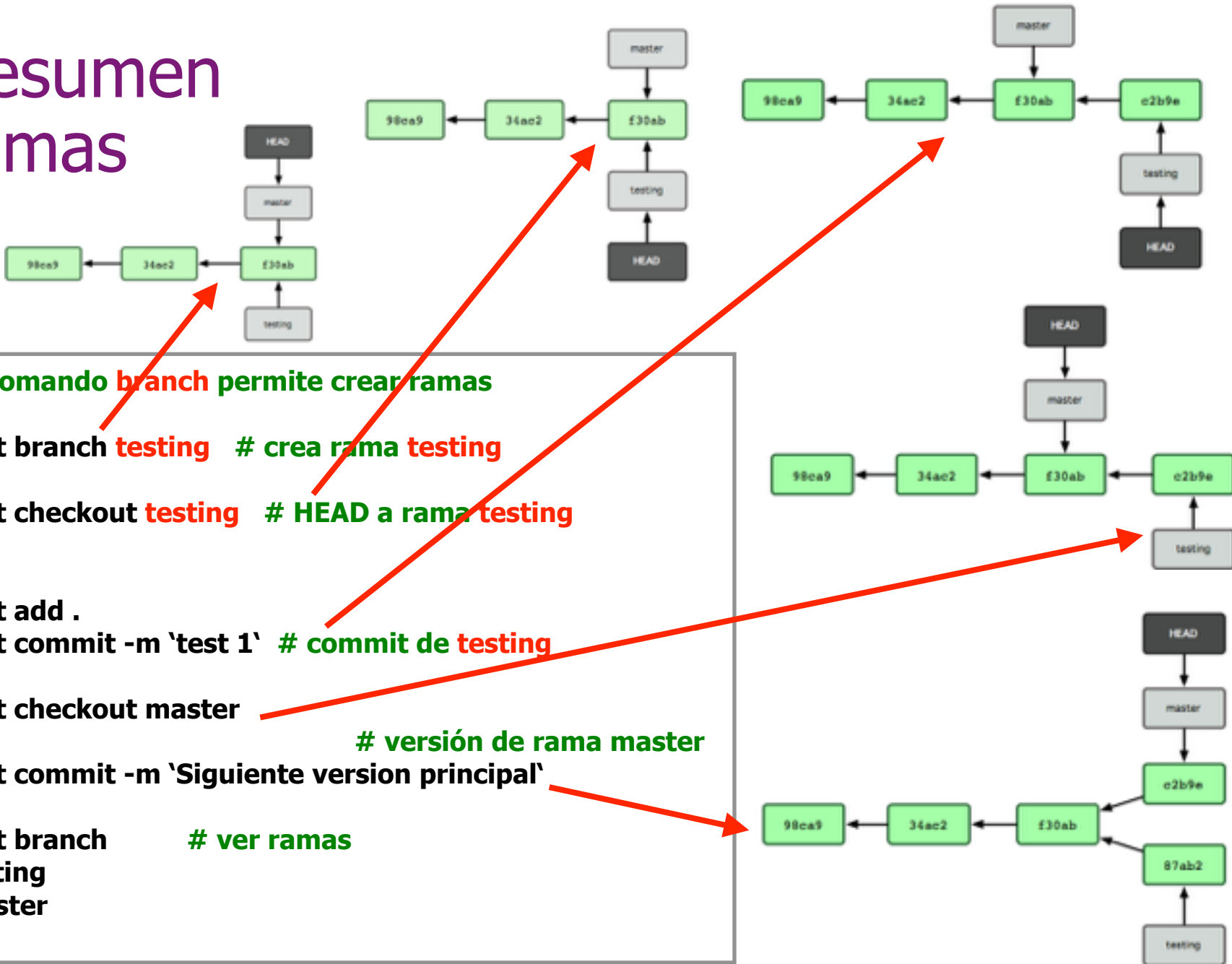
versión de rama master

..\$ **git commit -m 'Siguiete version principal'**

..\$ **git branch testing** # ver ramas

* master

..\$



Crear ramas y cambiar de rama

- ✦ Para crear una rama nueva en el punto de trabajo actual:

git branch <rama>

- ✦ Para crear una rama nueva en un commit dado:

git branch <rama> <commit>

- ✦ El comando checkout con la opción -b también permite crear una rama y cambiarse a ella:

git checkout -b <rama> <commit>

Cambiar de rama

- ◆ Para cambiar de rama:

git checkout <rama>

- ◆ Para crear una rama y cambiarse a ella:

git checkout -b <rama>

- ◆ Checkout falla si hay cambios en los ficheros del directorio de trabajo o en el staging area que no están incluidos en la rama a la que nos queremos cambiar.

- ★ Podemos forzar el cambio usando la opción **-f**.

- perderemos los cambios realizados

- ★ Podemos usar la opción **-m** para que nuestros cambios se mezclen con la rama que queremos sacar

- Si aparecen conflictos, los tendremos que solucionar.

Más usos de "git checkout"

"git checkout ." deshace cambios staged de area de trabajo

..\$ git checkout .

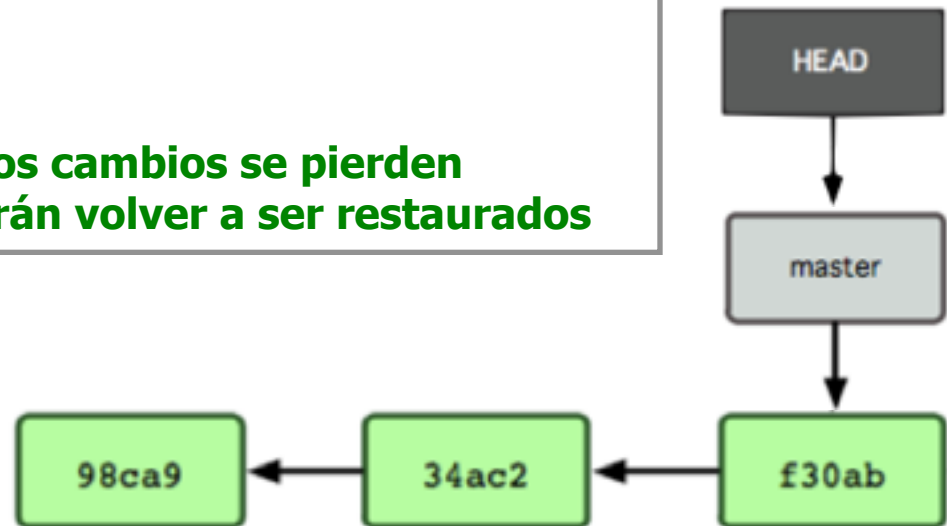
"git checkout -- <fichero>" deshace cambios staged

de <fichero> en area de trabajo

..\$ git checkout -- <fichero>

!!OJO!! Los cambios se pierden

y no podrán volver a ser restaurados

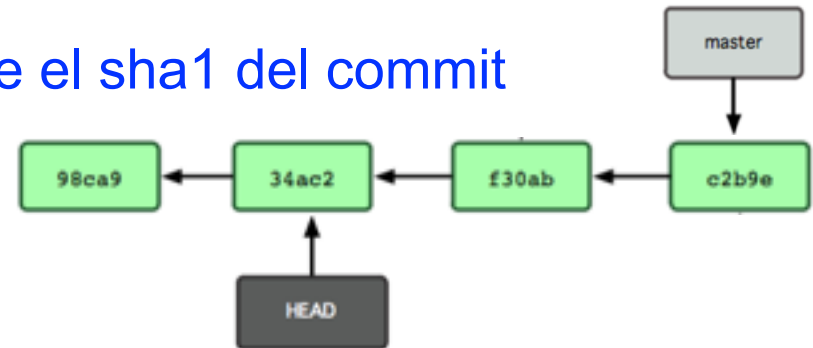


Detached HEAD Branch

- ◆ Es una rama que se crea sobre un commit que no tiene un nombre de rama apuntándole.

\$ git checkout 34ac2

- ★ Se ha realizado el checkout sobre el sha1 del commit



- ◆ Suele hacerse para inspeccionar una versión anterior
- ◆ “Detached HEAD branch” crea una rama sin nombre
 - ★ Para guardar cambios debe crearse rama con: **git branch <xx>**
 - ★ También se podía haber creado la rama al realizar checkout
 - **git checkout -b <nombre_rama> 34ac2**



GIT

8. Unir ramas: merge

Unir ramas

- ◆ Para incorporar en la rama actual los cambios realizados en otra rama:

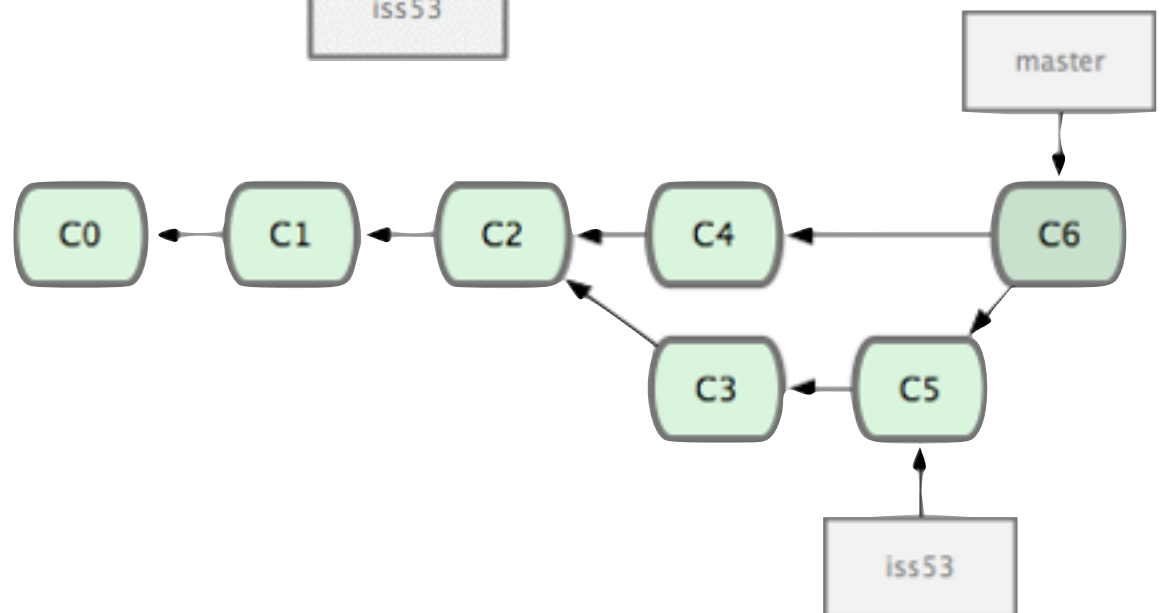
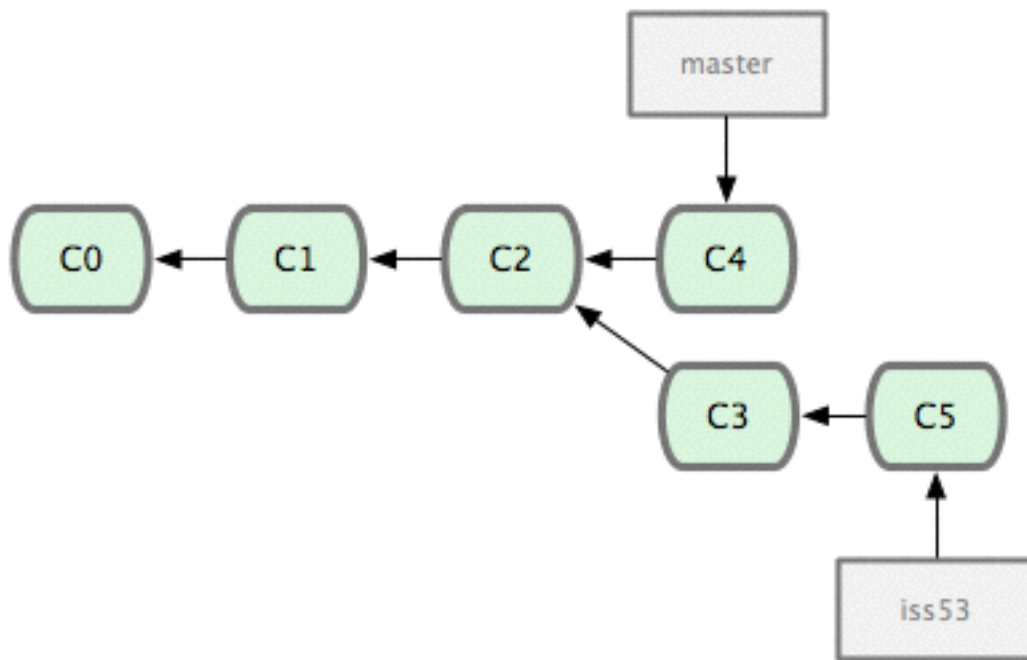
git merge <rama>

- ◆ Internamente GIT analiza la historia de commits para calcular como hacer la mezcla.

★ Puede hacer un fast forward, una mezcla recursiva, ...

- ◆ Ejemplo:

```
$ git checkout master # Estamos en la rama master
$ git merge iss53 # incorporamos los cambios hechos
# en la rama iss53 en la rama master
```



Conflictos

- ♦ Al hacer el merge pueden aparecer conflictos
 - ★ las dos ramas han modificado las mismas líneas de un fichero.
- ♦ Si hay conflictos:
 - ★ no se realiza el commit
 - ★ las zonas conflictivas se marcan

```
<<<<<< HEAD:index.html
<div id="footer">contact : email.support@github.com</div>
=====
<div id="footer">contact us at support@github.com</div>
>>>>>> iss53:index.html
```
 - ★ **git status** lista los ficheros con conflictos como **unmerged**.
- ♦ Para resolver el conflicto hay que:
 - ★ editar el fichero para resolver el conflicto.
 - ★ y ejecutar **git add** y **git commit**.

Borrar ramas

- ♦ Una vez terminado el trabajo con una rama
 - ★ la borraremos con **git branch -d <nombre>**
 - Se elimina el puntero al commit.
- ♦ Si la rama a borrar no ha sido mezclada con la rama actual
 - ★ se muestra un mensaje de error y no se borra.
 - ★ Para borrar la rama independientemente de si ha sido mezclada o no, usar la opción **-D** en vez de **-d**.
git branch -D <nombre>

Listar ramas

♦ Para listar las ramas existentes: **git branch**

\$ git branch

iss53

*** master**

testing

★ Se marca con un asterisco la rama activa.

♦ Opciones:

★ **-r** muestra ramas remotas

★ **-a** muestra todas las ramas (locales y remotas)

★ **-v** muestra el último commit de la rama.

★ **--merged** muestra las ramas que ya se han mezclado con la rama actual.

★ **--no-merged** muestra las ramas que aun no se han



GIT

9. Repositorios remotos

¿Qué es un repositorio remoto?

- ◆ En un proyecto GIT real suelen existir varias copias del repositorio en servidores externos, por ejemplo
 - ★ Un repositorio en GITHUB para colaborar con otros programadores
 - ★ Un repositorio en Heroku con el que hacemos el despliegue en la nube
- ◆ Son repositorios remotos del proyecto que
 - ★ Permite subir (push) o bajar (pull, fetch) contribuciones
 - Compartiendolas con un equipo o una comunidad
- ◆ Un repositorio remoto se puede referenciar por URL o nombre
 - ★ Su URL (largo e incomodo)
 - ★ El nombre (**remote**) es más corto y más cómodo de usar

Repositorios remotos (remote)

El comando **git remote** lista los nombres de los remotes definidos.
origin es el remote que se creó automáticamente al clonar el proyecto.

```
$ git remote  
koke  
origin
```

Usar la opción **-v** para ver las URL de los repositorios remotos.

```
$ git remote -v  
koke      git://github.com/koke/grit.git  
origin    git@github.com:mojombo/grit.git
```

Para crear un remote se usa el comando **git remote add** **shortname** **URL**

```
$ git remote add pepito git://github.com/pepe/planet.git
```

Los repositorios remotos se pueden manejar con 3 tipos de URL

```
# tipo git:      git@github.com:jquemada/swcm2012.git  
# tipo ssh:      ssh://github.com/jquemada/swcm2012  
# tipo https:    https://github.com/jquemada/swcm2012
```


Más comandos sobre remotes

Inspeccionar detalles de un remote: **git remote show** [nombre_del_remote]

Muestra el URL del remote, información sobre las ramas remotas,
las ramas tracking, etc.

Renombrar un remote: **git remote rename** nombre_viejo nombre_nuevo

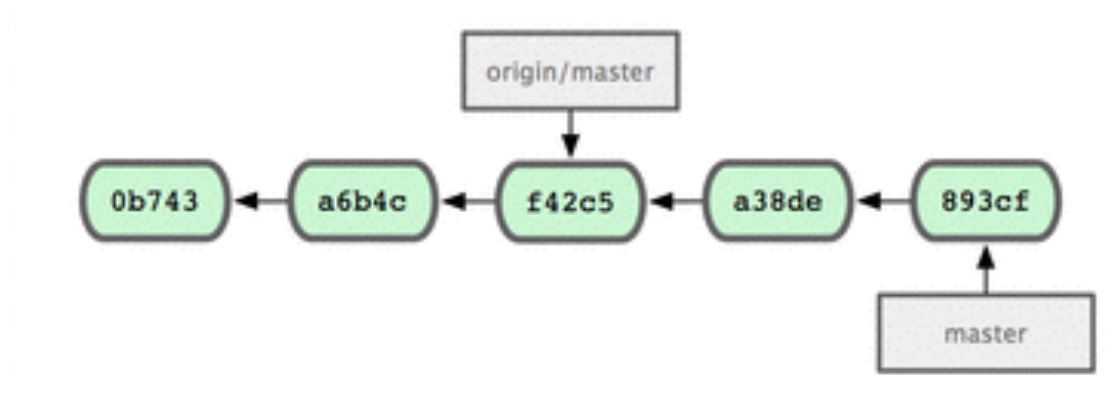
Borrar un remote: **git remote rm** nombre_del_remote

Para actualizar la información sobre los remotes definidos: **git remote update**

Para borrar ramas que ya no existen en el remote: **git remote prune**

Ramas Remotas

- ♦ Una rama remota es un puntero a un commit.
 - ★ Indica cual era la posición de una rama en un repositorio remoto la última vez que nos conectamos con él.
- ♦ Se nombran como: **<remote>/<rama>**.
- ♦ La figura muestra donde estaba la rama **master** en el repositorio **origin** la última vez que nos actualizamos.



- ♦ Este puntero no lo podemos mover manualmente.
 - ★ Se mueve cuando actualizamos con el repositorio remoto.

Tracking Branch

- ◆ Es una rama local emparejada con una rama remota para que estén sincronizadas.
 - ★ Hacer un seguimiento de los cambios realizados en ellas
 - Al hacer **git push** en una tracking branch se suben los cambios locales y se actualiza la rama remota emparejada.
 - Al hacer **git pull** se actualiza la rama local con los cambios existentes de la rama remota emparejada.
- ◆ La rama master que se crea al clonar un repositorio es una tracking branch de origin/master.
- ◆ Para listar las tracking branches existentes:
 - ★ **git branch -vv**
 - ★ **git remote show <remote_name>**

✦ Para crear una tracking branch, ejecutaremos:

git checkout -b <branchname> <remotename>/<branchname>

★ Se crea una rama local que hace el seguimiento de la rama remota indicada.

- Nótese que el nombre local y remoto de la rama puede ser distinto.

✦ También podemos crear una tracking branch ejecutando:

git checkout --track <remotename>/<branchname>

★ Se crea una rama local que hace el seguimiento de la rama remota indicada, usando el mismo nombre.

Descargar datos de un remote

Bajarse los datos de un remoto: **git fetch** [nombre_del_remote [refspec]]

refspec:

- # - indica las ramas local y remota entre las que se hará la bajada de datos.
- # - puede ser un nombre de una rama (tanto local como remota)
- # - Si no es específica este parámetro, se bajan las actualizaciones de todas las ramas locales que también existan en el repositorio remoto.

Este comando actualiza el repositorio con los datos existentes en el remote, pero no modifica los ficheros del directorio de trabajo.

Las operaciones de merge las deberemos invocar explícitamente.

Ejemplo:

Bajarse los datos que aun no tengo del repositorio del que me cloné:

\$ **git fetch** origin

Ahora mezclo mi rama actual con la rama demo de origin:

\$ **git merge** origin/demo

Descargar datos y Mezclar

Bajarse los datos de un remoto y aplicar merge:

git pull [nombre_del_remote [refspec]]

Si la rama actual es una tracking branch:

El comando **git pull** [nombre_del_remote], actualiza la rama actual con los cambios realizados en la rama asociada del repositorio remoto.

\$ **git pull origin** # Actualiza rama actual con los cambios en origin.

\$ **git pull** # Por defecto se realiza un pull de origin.

Este comando ejecuta un fetch con los argumentos dados, y despues realiza un merge en la rama actual con los datos descargados.

\$ **git pull pepito demo** # Mezcla en la rama actual la rama demo de pepito.

Subir datos a un remote

De forma general, el comando para subir cambios a un remote es:

git push [nombre_remote [refspec]]

La operación push sólo puede hacerse sobre repositorios bare.

Son repositorios donde no se desarrolla. Sólo se suben cambios.

No tienen un directorio de trabajo.

Los repositorios bare se crean con init o clone, y usando la opción --bare.

Si la rama actual es una tracking branch, no suele especificarse un refspec:

#

El comando **git push** [nombre_del_remote], sube los cambios desarrollados en

la rama local actual a la rama asociada del repositorio remoto.

\$ **git push origin master** # Subir los cambios en la rama master local a origin.

\$ **git push origin** # Subir los cambios de la rama local actual o origin.

\$ **git push** # Por defecto el remote es origin.

La operación push sólo puede realizarse si:

- se tiene permiso de escritura en el repositorio remoto.

- nadie ha subido nuevos cambios al repositorio remoto, es decir,
estamos actualizados.

* Si en el remote hay actualizaciones que no tenemos, deberemos hacer
un pull antes de poder subir nuestros cambios.

* No pueden subirse actualizaciones que puedan producir conflictos.

Si no es especifica un valor para refspec, se suben las actualizaciones de todas las
ramas locales que también existan con el mismo nombre en el repositorio remoto.

Si se crea una rama local, y se quiere subir al repositorio remoto, debe
ejecutarse el comando push con el nombre de la rama como valor de refspec:

\$ git push origin prueba

refspec permite indicar los nombres distintos para las ramas local y remota:
Subir los cambios de rama local uno a la rama dos del repositorio origin:

\$ git push origin uno:dos

refspec tiene el formato <rama_local>:<rama_remota>

Para borrar una rama remota :

\$ git push origin :dos

Usamos un refspec con un nombre de rama local vacío,
y con el nombre de la rama remota a borrar.



Final del tema