Tecnologías Git y GitHub

Alfredo Abad

ISO-04-21-git.pptx

UA: 26-mar-2019

http://blog.udacity.com/2015/06/a-beginners-git-github-tutorial.html
http://www.htcmania.com/showthread.php?t=674598
Tutorial: http://rogerdudler.github.io/git-guide/index.es.html

1

Sistemas de control de versiones

- Un sistema de control de versiones es un conjunto de herramientas que nos permite registrar los cambios que se producen en los archivos de un proyecto a lo largo del tiempo, y que por tanto nos permite volver a cualquier estado del mismo a nuestra voluntad
- Podemos encontrar distintos tipos de sistemas de control de versiones:
  - **Sistemas locales**, que sólo existen en nuestros propios equipos de trabajo. Un ejemplo de este tipo sería por ejemplo RCS
  - **Sistemas centralizados**, que dependen de un servidor centralizado que almacena toda la información, encargándose de distribuir distintas copias entre los distintos usuarios y guardar todos los cambios que se van produciendo. Algunos ejemplos de este tipo serían CVS, Subversion, o Perforce
  - **Sistemas de control distribuido**, en los que cada participante en el proyecto tiene una copia completa del mismo sin depender de un servidor que almacene toda la información. Algunos ejemplos serían GIT o Mercurial

2

¿Qué es Git y GitHub?

- Son tecnologías que se encargan de los procesos de gestión de versiones de software
- GitHub, además, añade la publicación del software en la sede de GitHub con dos posibilidades:
  - Gratuito: solo para publicar software libre
  - De pago
- Son herramientas bastante sofisticadas y ampliamente utilizadas

3

http://blog.udacity.com/2015/06/a-beginners-git-github-tutorial.html

FUNDAMENTOS BÁSICOS DE GIT

4

¿Por qué GIT?

- GIT es un sistema de control de versiones que surge tras la necesidad de dar soporte a la gran cantidad de cambios que se producían en el Kernel de Linux y tras las diversas discrepancias que se produjeron allá por el año 2005 con BitKeeper, herramienta que se usaba por aquel entonces para mantener todo el código
  - Esta situación impulsó a la comunidad de desarrolladores de Linux y en especial a Linus Torvalds a desarrollar su propio sistema de control de versiones. De esa iniciativa surgió GIT
- En la actualidad GIT es una de las mejores opciones que existen por las siguientes razones:
  - Tiene un diseño sencillo y prácticamente transparente para sus usuarios
  - Es veloz y sobre todo eficiente
  - Cuenta con un increíble sistema de ramificaciones convirtiéndolo en una fabulosa herramienta para el desarrollo no lineal de proyectos
  - Es completamente distribuido y por tanto capaz de manejar grandes proyectos

5

Instalación del software

- Se requiere una copia de **Git** instalada en el sistema que se vaya a utilizar
  - Existen versiones para distintos sistemas operativos: Windows, Linux, etc.
  - El software de instalación se puede descargar a través de los repositorios de las distros o desde la web de Git
- Git se utiliza a través de la línea de comandos, aunque también hay disponibles interfaces gráficos
- Para nuestras pruebas confeccionaremos un repositorio sencillo en el que incluiremos un fichero con código fuente y un README
  - Después se estudiarán y practicarán los comandos típicos de git: init, clone, add, commit, diff, log, etc.

6

Los tres estados de GIT

- Cuando trabajamos con GIT, nuestra información puede estar en una de las siguientes situaciones:
  - **Información modificada (modified)**
    - Este estado implica que hemos modificado nuestra información, pero aún no está siendo trackeada por GIT
  - **Información preparada (staged)**
    - Este estado implica que hemos marcado nuestra información para posteriormente ser confirmada y por tanto trackeada como nueva versión
  - **Información confirmada (committed)**
    - Este estado implica que nuestra información ha sido almacenada en la base de datos local de GIT

7

Inicio de un repositorio Git

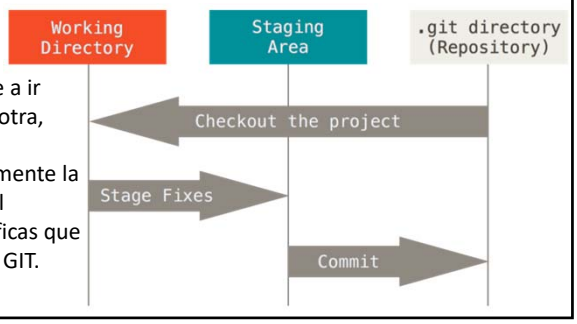
- Preparación:
  - Crear un directorio de trabajo sobre el que crear el repositorio (en nuestro caso: video-lister)
  - Nos posicionamos en ese directorio por defecto
- Creación e inicio del repositorio:
  git init
  - Esto crea un subdirectorio oculto (.git), que es donde Git almacenará sus bases de datos y configuración
  - También crea una rama del proyecto por defecto que es la rama MASTER

8

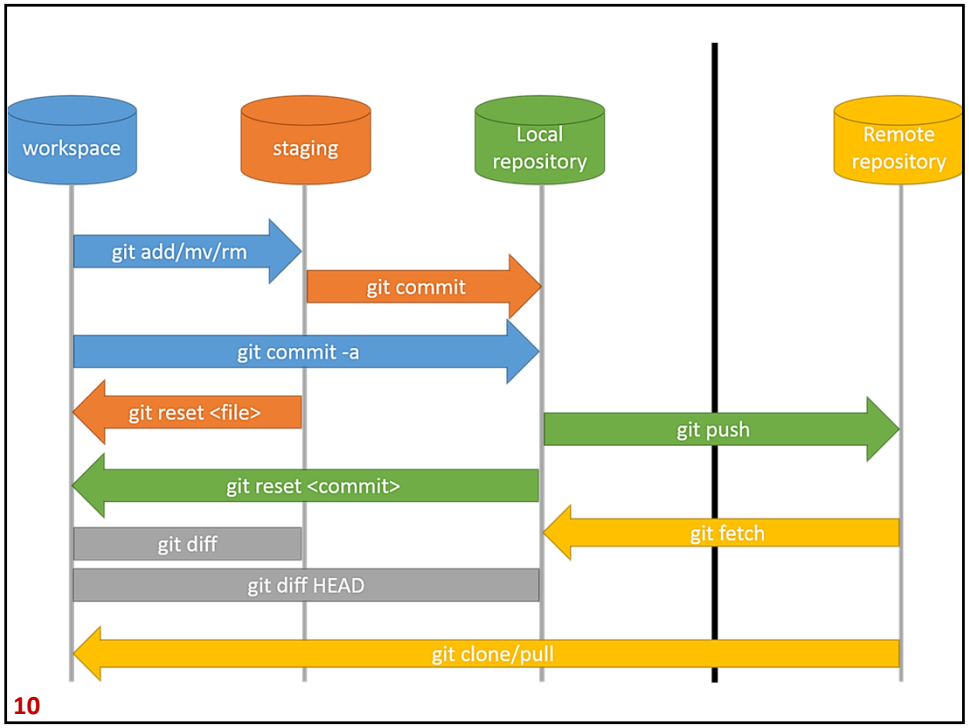
Relaciones entre componentes: los tres estados de GIT

- Estas situaciones dan lugar a lo que se conoce como los tres estados de GIT:
  - El **working directory**, que es nuestra área de trabajo en la que hacemos los cambios en nuestros ficheros
  - El **staging area**, que es el espacio donde colocaremos aquellos ficheros listos para ser colocados en el repositorio
  - Y el **repositorio**, que es el área donde GIT irá guardando las distintas versiones de nuestra información

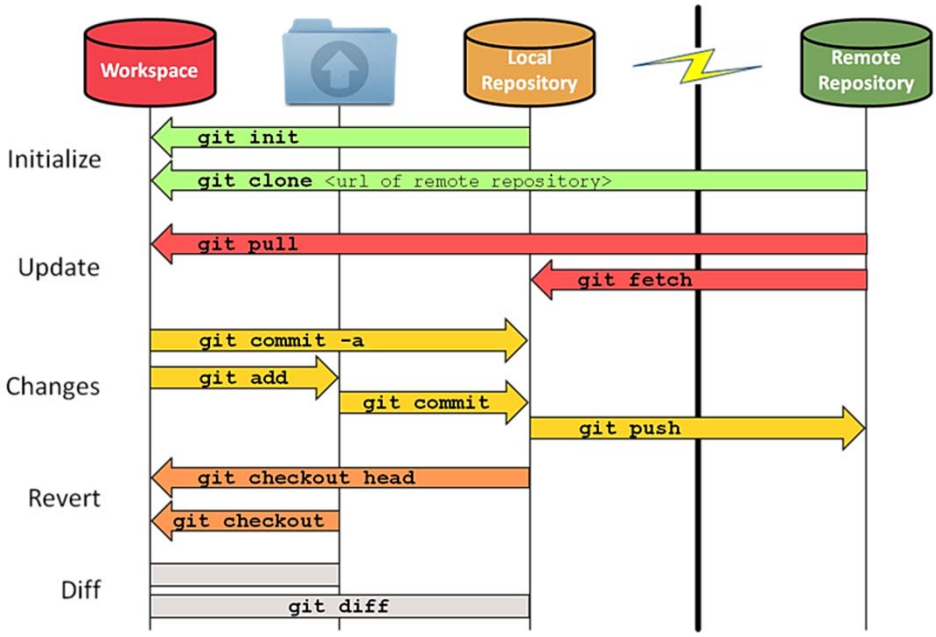
Todo nuestro trabajo con GIT se reduce a ir moviendo la información de un área a otra, gestionando de esa forma sus distintos estados. Para ello usaremos preferiblemente la línea de comandos de nuestro terminal aunque también existen utilidades gráficas que nos facilitan nuestro trabajo diario con GIT.



9



10



11

Configurando GIT

- En primer lugar nos aseguraremos que lo tenemos instalado en nuestro equipo
  - Para verificar si está o no instalado, podemos ejecutar este comando **git version**
  - Si obtenemos respuesta nos indicará la versión de GIT que tenemos instalada, y en caso de que no lo esté, tendrás que instalarlo en tu equipo
  - Aquí hay una guía de instalación:
    - https://git-scm.com/book/es/v2/Inicio---Sobre-el-Control-de-Versiones-Instalaci%C3%B3n-de-Git

12

Configuraciones básicas iniciales

- Todas las operaciones en GIT van firmadas con la información básica del usuario: user.name y user.email
  - Para configurar dicha información en GIT ejecutaremos los siguientes comandos en nuestro terminal:
    - **git config --global user.name "name"**
    - **git config --global user.email "email"**
- El modificador **--global** indica a GIT que esta configuración se usará por defecto siempre que trabajemos con GIT
  - Estos parámetros de configuración podemos personalizarlos a nivel de proyecto tan sólo quitando el modificador **--global** del comando
- Si queremos conocer cuál es nuestra configuración actual podemos ejecutar los siguientes comandos:
  - **git config user.name**, o bien **git config user.email**

13

Otras configuraciones iniciales

- Vamos a configurar un par de parámetros más de forma global:
  - **git config --global color.ui true**
    - Indicamos a GIT que use color para mostrarnos distintos tipos de información
  - **git config --global core.editor "nano"**
    - Configura el editor de texto por defecto que usará GIT cuando necesitemos por ejemplo, anotar información acerca de los cambios que se van a confirmar
- Podríamos hacer más configuraciones, pero en principio para comenzar tenemos suficiente
- Si queremos ver nuestra configuración global, ejecutaremos el comando **git config --list**

14

Clonar un repositorio

- Hay un segundo modo de acceder a un repositorio que es mediante clonación de otro previamente existente
- Se clona con
  git clone <repository URL (remoto)>
  - Que creará una copia exacta del repositorio indicado (remoto) en el repositorio local
  - Después de esto se puede trabajar con la copia local y luego reintegrar los cambios de nuevo al remoto

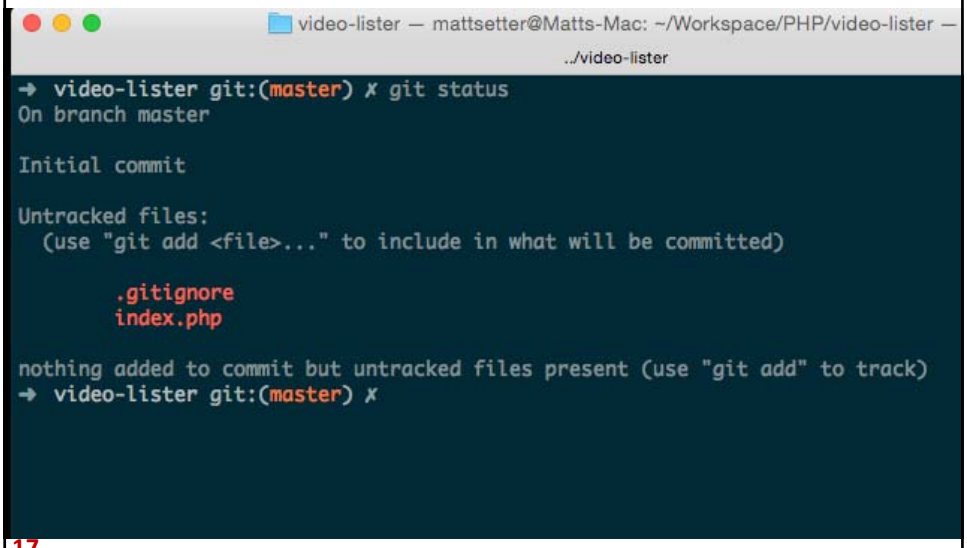
15

Añadir un nuevo fichero a un repositorio

- Crearemos un fichero local en código fuente (index.php, en nuestro caso) en el directorio raíz de nuestro repositorio local
  - El código fuente (no significativo) es:
    <?php print "Hello World“>;
- Después de salvar el código del fichero ejecutaremos
  git status
  - Que mostrará el estado del repositorio local de trabajo (ver diapo siguiente)

16

Ejecución de git status



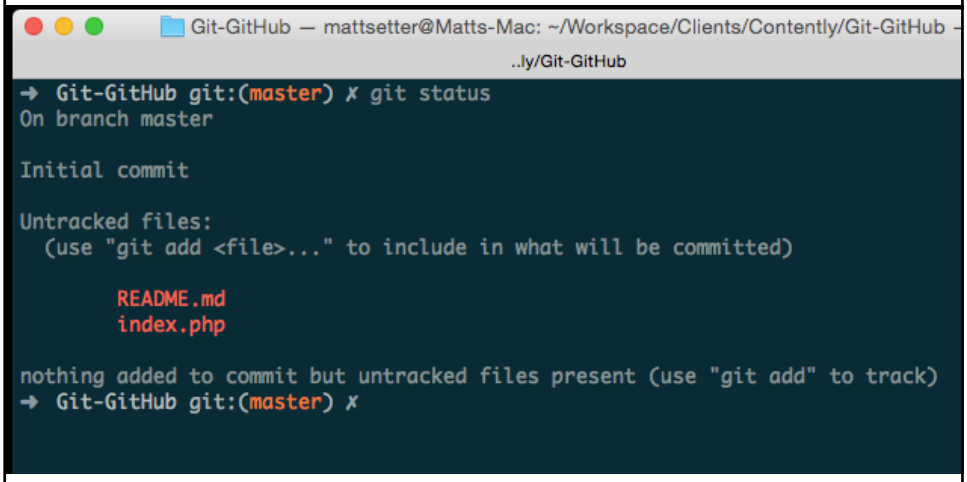
17

Seguimos trabajando sobre el repositorio de trabajo

- Podemos añadir u operar con más ficheros como lo haríamos normalmente en cualquier directorio
  - Añadiremos un nuevo fichero **README.md**
  - (Todo nuevo buen proyecto debería tener un README.md con su descripción)
- Ahora ejecutamos de nuevo **git status** para evaluar el estado del repositorio (diapo siguiente)

18

Estatus de los dos ficheros (untrack)

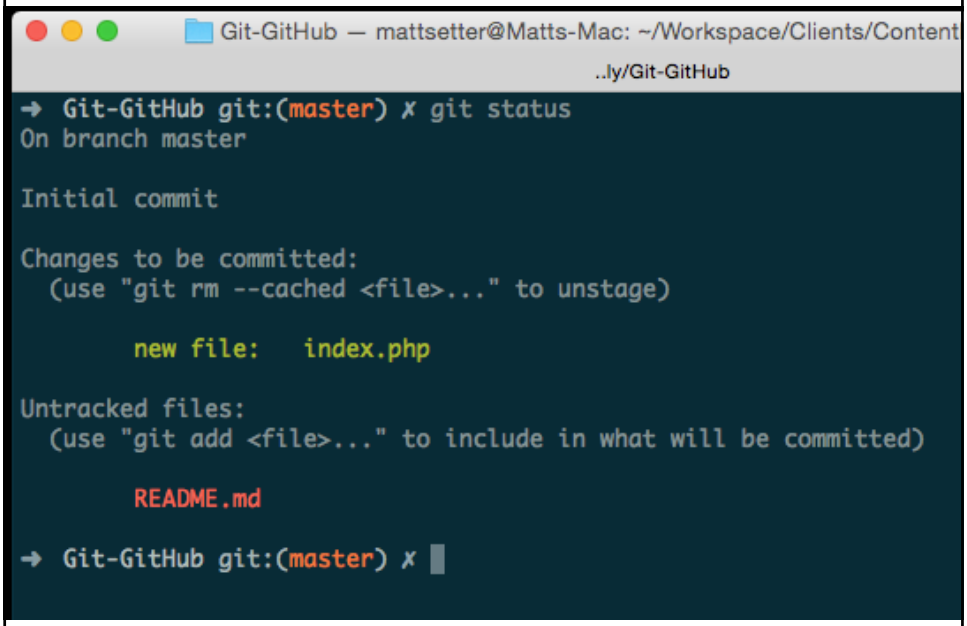


19

Gestión de índice del fichero index.php

- De momento, nos olvidamos de REAME.md y nos fijamos en index.php
- Ejecutamos
  git add index.php
  - Y después un git status para ver el estado, que habrá cambiado (ver diapo siguiente)
  - index.php habrá sido registrado en el control del repositorio (mientras que README.md seguirá alojado pero sin sometimiento a ningún control)

20



21

Actualizando la configuración de Git

- Una vez indexado el fichero index.php, ahora podremos realizar sobre él un **commit** cuyos mensajes se escribirán con un editor por defecto
- Git usa por defecto el editor indicado en las variables $VISUAL o $EDITOR, que normalmente será pico, vi, vim o emacs
- Si queremos especificar un editor concreto (Notepad, TextEdit, Gedit, etc.), lo podemos hacer con:
  - **git config --global core.editor <nombre-de-aplicación>**

22

Realizar el primer commit

- **Commit** es fijar los cambios realizados asociándolos a una versión concreta de la que se gestionará su control
- Se utilizará el comando **git commit**
  - Que abrirá automáticamente el editor seleccionado con una plantilla (diapo siguiente) para rellenar en donde especificaremos los cambios realizados
- El **commit** solo afectará a los ficheros del proyecto que hayan sido indexados previamente (en nuestro caso, index.php; README.md no será afectado)

23

Ejemplo de plantilla de documentación de commit

```
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
#
# Initial commit
#
# Changes to be committed:
#       new file:   index.php
#
# Untracked files:
#       README.md
#
```

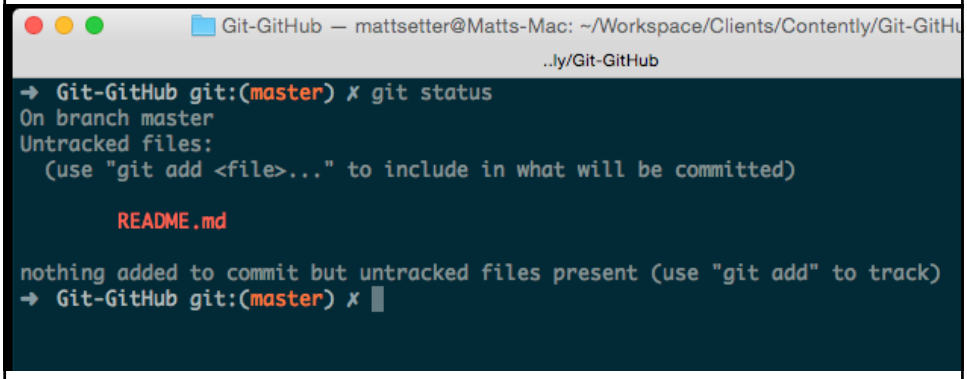
24

El mensaje de commit

- Un buen mensaje de commit consta de dos partes:
  - Un mensaje corto (máximo de 72 caracteres) con una breve descripción de los cambios
  - Un mensaje mucho más largo (opcional) con una descripción detallada de los cambios

25

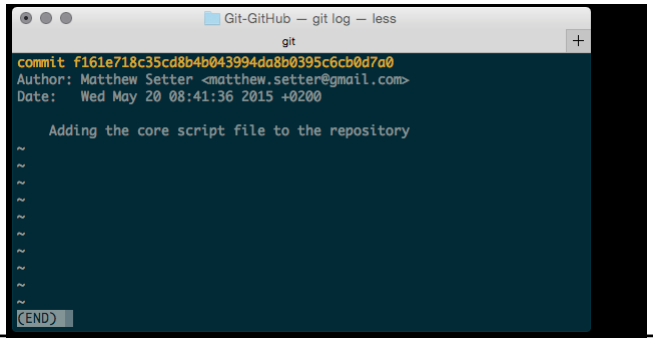
git status después de git commit



26

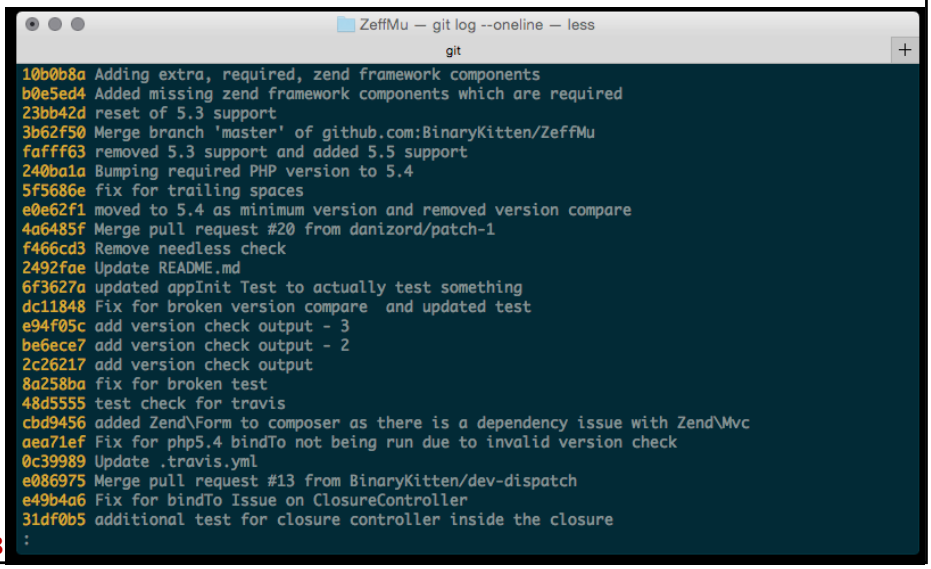
Examinar diferencias y vista de la historia de cambios

- Se pueden examinar las diferencias (local – remota) de un fichero con **git diff**
  - Más información en https://git-scm.com/docs/git-diff
- El histórico de cambios ser irá escribiendo automáticamente y se puede consultar con **git log**
  - Más información con **git help log**



27

Visualizar el hash y el mensaje del comit con git log --oneline

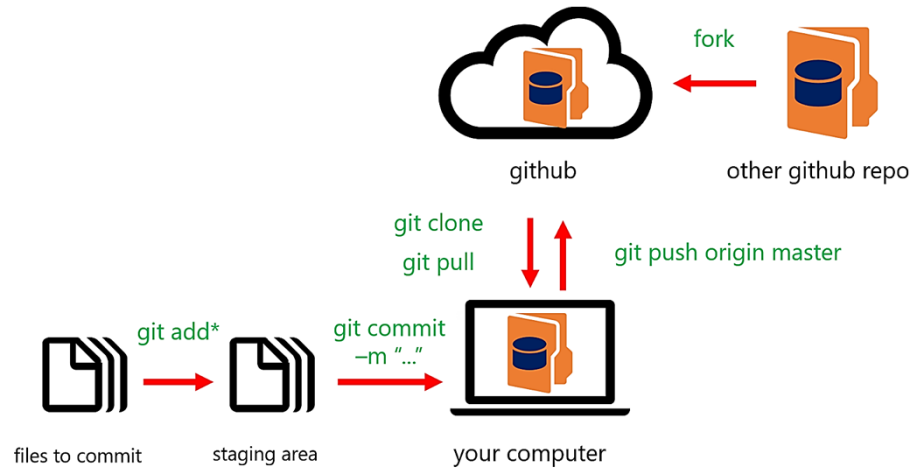


28

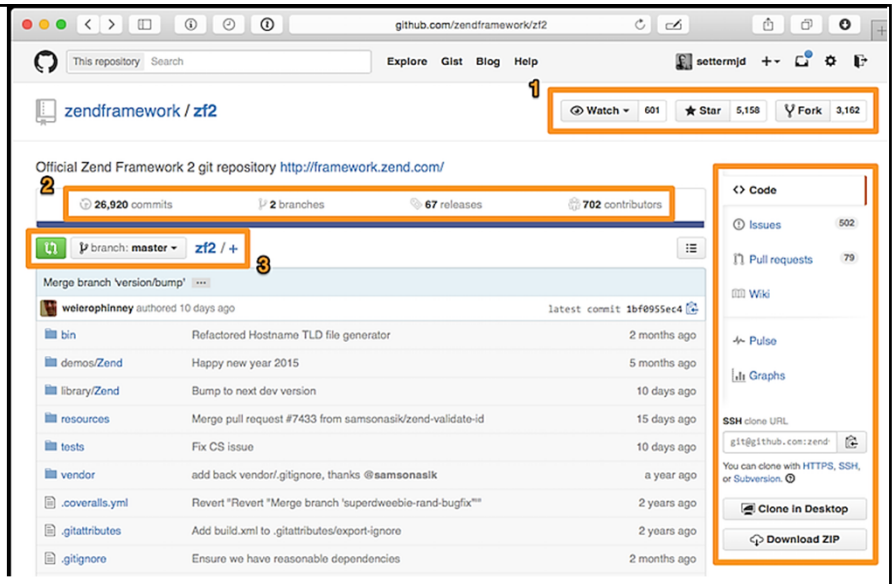
Las ramas de un proyecto (branching) Mainline trunk (en terminología Git)

- Todo proyecto Git tiene necesariamente una rama MASTER, que es la que utiliza por defecto (en todos los anteriores ejemplos)
  - Después se pueden abrir nuevas ramas para hacer derivaciones del proyecto o para hacer pruebas o para dividir el conjunto total
- Para crear una nueva rama (develop) se utiliza el comando **git checkout -b develop**
  - Que en su inicio será una copia del master
- Se puede reintegrar una rama (develop) en la rama de que deriva (master) con el comando **git merge develop**

Resumen gráfico del uso de git



30



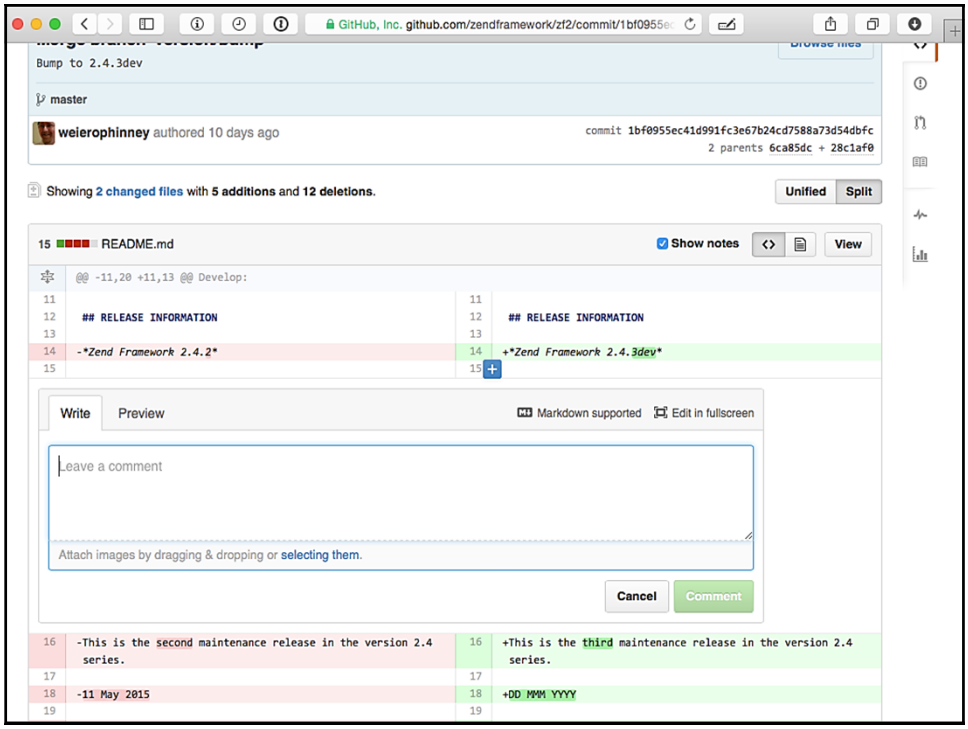
GITHUB

31

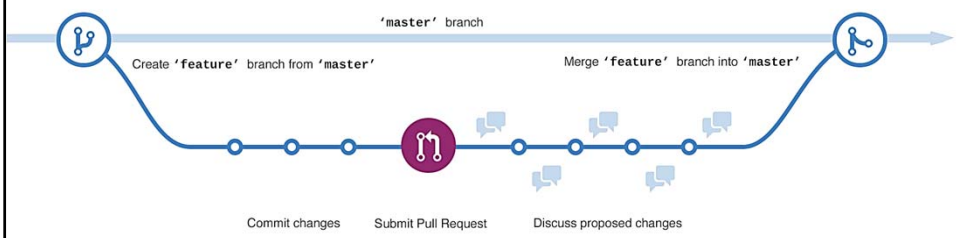
¿Qué es GitHub?

- Es una plataforma de hosting de código que aloja en la nube los proyectos Git de sus usuarios registrados
  - De forma que proporciona una plataforma web para la gestión de un repositorio Git público a través de un homepage de proyecto
- Para clonar un proyecto basta con hacer clic en el botón web correspondiente de la página de proyecto (equivalente a git clone)
  - Hay otras opciones disponibles desde la homepage como commit, diff, visualizar historial, etc.

32



Ejemplo de branch en GitHub



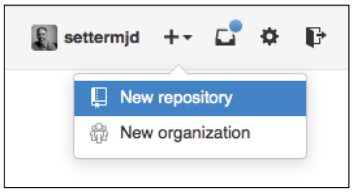
34

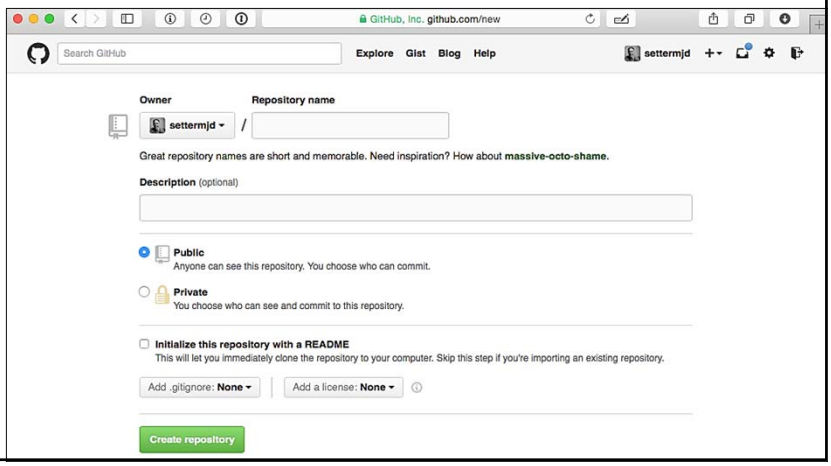
Barra de navegación de GitHub

- **Code**: The view you're on by default, showing the files in the project.
- **Issues**: A simple but effective issue tracker, whether you and the team want to report bugs and problems, make requests for new features, or other such tasks.
- **Wiki**: A simple but effective wiki for documenting the project in more detail than a standard `README` file allows.
- **Pulse**: A summary of statistics about the project, including open and closed issues. Here is where you find out how active the project is.
- **Graphs**: A timeline of commits, followed by a breakdown of commits by individual contributor. You can then use the available tabs to look at the project activity in detail, based on a series of key metrics including code frequency, least to most active days for contributions, and so on.

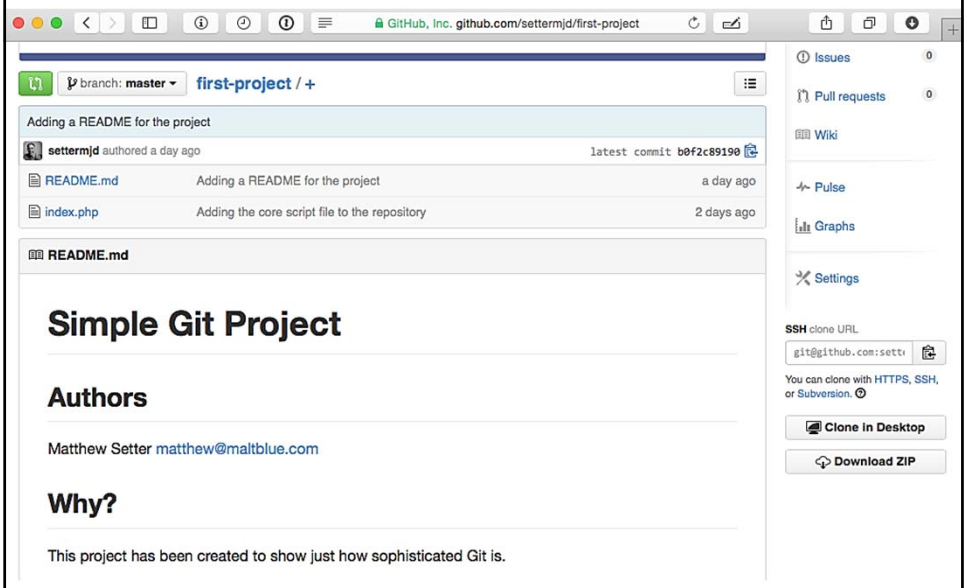
35

Añadir un proyecto a GitHub





36



37

- Más información en:
  - https://guides.github.com/activities/hello-world/

38

http://rogerdudler.github.io/git-guide/index.es.html

Descarga git para OSX
Descarga git para Windows
Descarga git para Linux

RESUMEN DE COMANDOS ÚTILES PARA GESTIONAR GIT

39

crea un repositorio nuevo

Crea un directorio nuevo, ábrelo y ejecuta

```
git init
```

para crear un nuevo repositorio de git.

hacer checkout a un repositorio

Crea una copia local del repositorio ejecutando

```
git clone /path/to/repository
```

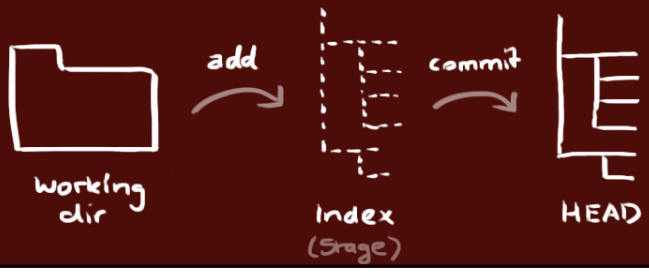
Si utilizas un servidor remoto, ejecuta

```
git clone username@host:/path/to/repository
```

40

flujo de trabajo

Tu repositorio local esta compuesto por tres "árboles" administrados por git. El primero es tu `Directorio de trabajo` que contiene los archivos, el segundo es el `Index` que actua como una zona intermedia, y el último es el `HEAD` que apunta al último commit realizado.



41

add & commit

Puedes registrar cambios (añadirlos al **Index**) usando

```
git add <filename>
```

```
git add .
```

Este es el primer paso en el flujo de trabajo básico. Para hacer commit a estos cambios usa

```
git commit -m "Commit message"
```

Ahora el archivo esta incluído en el **HEAD**, pero aún no en tu repositorio remoto.

42

envío de cambios

Tus cambios están ahora en el **HEAD** de tu copia local. Para enviar estos cambios a tu repositorio remoto ejecuta

```
git push origin master
```

Reemplaza *master* por la rama a la que quieres enviar tus cambios.

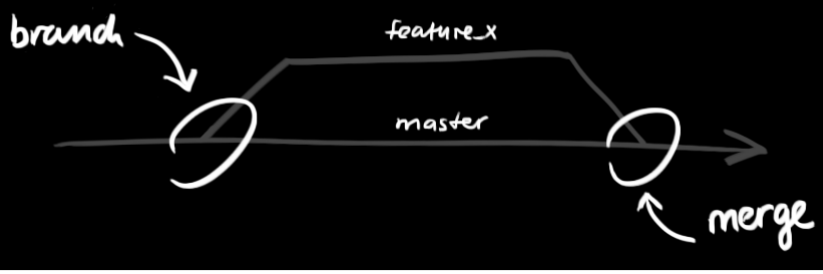
Si no has clonado un repositorio ya existente y quieres conectar tu repositorio local a un repositorio remoto, usa

```
git remote add origin <server>
```

Ahora podrás subir tus cambios al repositorio remoto seleccionado.

ramas

Las ramas son utilizadas para desarrollar funcionalidades aisladas unas de otras. La rama *master* es la rama "por defecto" cuando creas un repositorio. Crea nuevas ramas durante el desarrollo y fusiónalas a la rama principal cuando termines.



Crea una nueva rama llamada "feature_x" y cámbiate a ella usando

```
git checkout -b feature_x
```

vuelve a la rama principal

```
git checkout master
```

y borra la rama

```
git branch -d feature_x
```

Una rama nueva *no estará disponible para los demás* a menos que subas (push) la rama a tu repositorio remoto

```
git push origin <branch>
```

45

actualiza & fusiona

Para actualizar tu repositorio local al commit más nuevo, ejecuta

```
git pull
```

en tu directorio de trabajo para *bajar* y *fusionar* los cambios remotos.

Para fusionar otra rama a tu rama activa (por ejemplo master), utiliza

```
git merge <branch>
```

en ambos casos git intentará fusionar automáticamente los cambios. Desafortunadamente, no siempre será posible y se podrán producir *conflictos*. Tú eres responsable de fusionar esos *conflictos* manualmente al editar los archivos mostrados por git. Después de modificarlos, necesitas marcarlos como fusionados con

```
git add <filename>
```

Antes de fusionar los cambios, puedes revisarlos usando

```
git diff <source_branch> <target_branch>
```

46

etiquetas

Se recomienda crear etiquetas para cada nueva versión publicada de un software. Este concepto no es nuevo, ya que estaba disponible en SVN.

Puedes crear una nueva etiqueta llamada *1.0.0* ejecutando

```
git tag 1.0.0 1b2e1d63ff
```

1b2e1d63ff se refiere a los 10 caracteres del commit id al cual quieres referirte con tu etiqueta. Puedes obtener el commit id con

```
git log
```

también puedes usar menos caracteres que el commit id, pero debe ser un valor único.

47

reemplaza cambios locales

En caso de que hagas algo mal (lo que seguramente nunca suceda ;) puedes reemplazar cambios locales usando el comando

```
git checkout -- <filename>
```

Este comando reemplaza los cambios en tu directorio de trabajo con el último contenido de HEAD. Los cambios que ya han sido agregados al Index, así como también los nuevos archivos, se mantendrán sin cambio.

Por otro lado, si quieres deshacer todos los cambios locales y commits, puedes traer la última versión del servidor y apuntar a tu copia local principal de esta forma

```
git fetch origin
git reset --hard origin/master
```

48

datos útiles

Interfaz gráfica por defecto

```
gitk
```

Colores especiales para la consola

```
git config color.ui true
```

Mostrar sólo una línea por cada commit en la traza

```
git config format.pretty oneline
```

Agregar archivos de forma interactiva

```
git add -i
```

49

Enlaces y recursos

- **Clientes gráficos**
  - GitX (L) (OSX, open source)
  - Tower (OSX)
  - Source Tree (OSX, free)
  - GitHub for Mac (OSX, free)
  - GitBox (OSX)
- **Guías**
  - Git Community Book
  - Pro Git
  - Think like a git
  - GitHub Help
  - A Visual Git Guide
- **Conectar Git y GitHub**
  - http://www.htcmania.com/showthread.php?t=674598

50

Páginas de interés

- Tutorial de GIT – Guía de Uso de GIT 2018
  - https://www.marindelafuente.com.ar/tutorial-de-git-guia-de-uso-de-git-2018/
- Aprendiendo GIT. Primeros pasos
  - https://antoniomasia.com/aprendiendo-git-primeros-pasos/

51