Coding Bootcamp

Sprint 1

Temario

Temario

- Control de Versiones.
- Git.
  - Secciones de un proyecto.
  - Branches.
  - Comandos básicos.
  - Interfaces gráficas.
- Repositorios remotos en la nube.

Control de Versiones

Control de versiones (1/5)

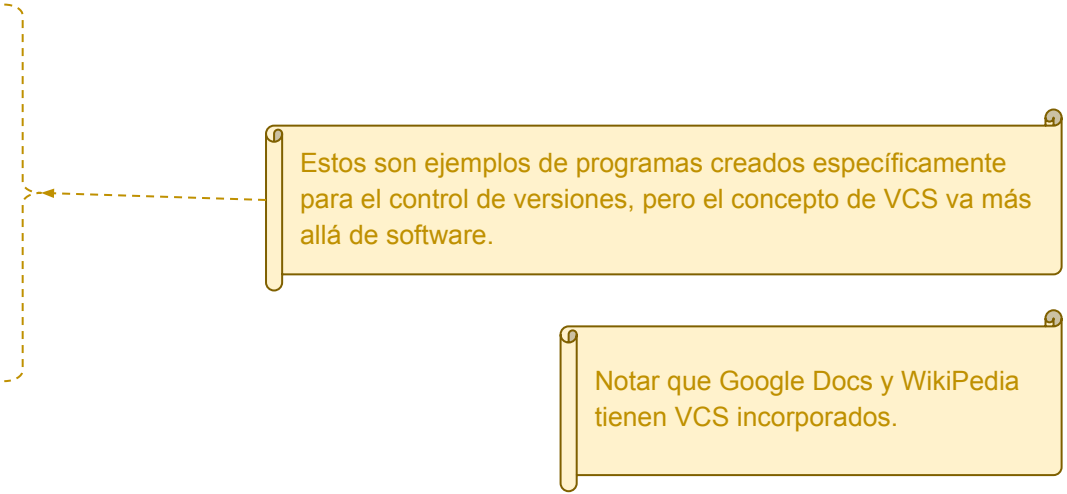
Preguntas que surgen en un proyecto:

- ¿Cuándo se modificó el archivo X la última vez?
- ¿Quién modificó el archivo X el lunes pasado?
- ¿Qué líneas de código se modificaron en el archivo X el lunes pasado?
- ¿El código con el que estoy trabajando es el más nuevo?
- ¿Qué se hace cuando dos desarrolladores modifican el mismo archivo?
- ¿Cómo es posible trabajar en varias versiones paralelas del proyecto?
- ¿Cómo se mantiene el código respaldado?
- Etc.

Control de versiones (2/5)

Para solucionar los problemas anteriores surgen los sistemas de control de versiones (en inglés: VCS). Los más conocidos son:

- Git.
- Subversion (SVN).
- Mercurial.
- Team Foundation Server.



Control de versiones (3/5)

Uno de los conceptos más importantes de los VCS es el de repositorio:

El repositorio es el lugar en el que se almacenan los archivos de los cuales se quieren controlar sus versiones.

Puede ser en un disco duro, en una base de datos, etc..

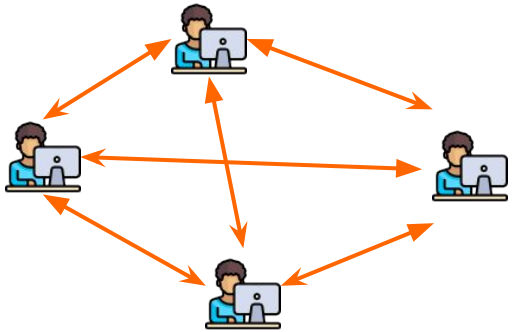
Control de versiones (4/5)

Básicamente, los sistemas de control de versiones se clasifican en:

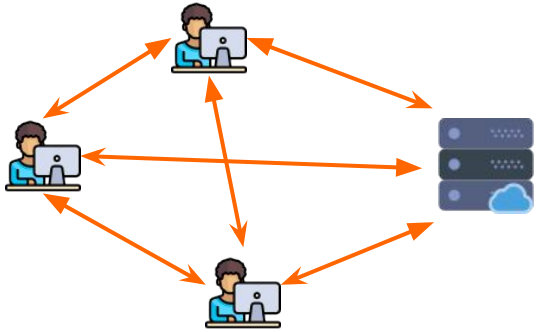
- Distribuidos:
  - Cada usuario tiene su propio repositorio local, por lo cual no se precisa conectividad.
  - Los distintos repositorios pueden intercambiar y mezclar revisiones entre ellos.
  - Es frecuente el uso de un repositorio central (en un servidor) que sirve como punto de sincronización de los distintos repositorios locales. Aquí reside el código "oficial" del proyecto.
- Centralizados:
  - Existe un repositorio centralizado de todo el código, del cual es responsable un único usuario (o conjunto de ellos).
  - Se facilitan las tareas administrativas a cambio de reducir flexibilidad, pues todas las decisiones fuertes (como crear una nueva rama) necesitan la aprobación del responsable.

Control de versiones (5/5)

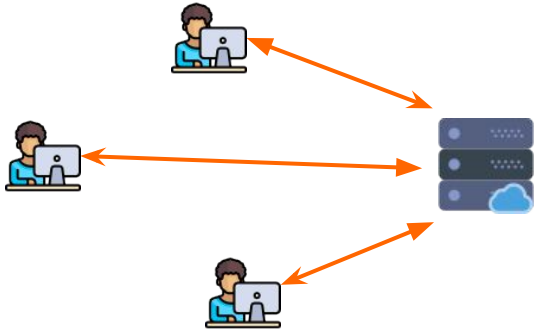
En sistemas distribuidos, cada usuario tiene su propio repositorio, pero lo usual es que una de las máquinas sea un servidor donde el código esté disponible para todo el equipo. De esta forma se emula un sistema centralizado, pero se mantienen las ventajas de los sistemas distribuidos.



Esquema distribuido tradicional (teórico)



Esquema distribuido con repositorio en servidor central



Esquema distribuido con repositorio en servidor central (común en la práctica)

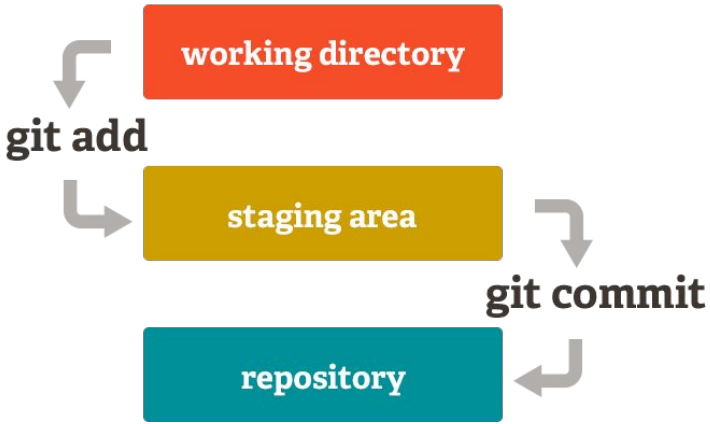
Git

Git

- Es el sistema de control de versiones más utilizado, por lejos.
- Fue creado por Linus Torvalds en 2005 (el creador de Linux).
- Es open source.
- Es distribuido.
- Fue creado pensando en eficiencia, seguridad y flexibilidad.
- Documentación: https://git-scm.com/doc.
- Para practicar: https://try.github.io.

Git – Secciones de un proyecto

En Git, cada archivo se encuentra en alguno de estos estados: `committed`, `modified` o `staged`. Esto conlleva a que existan tres secciones en un proyecto: el directorio de trabajo que contiene los archivos, el staging area (también llamado `Index`) que actúa como una zona intermedia y el repositorio propiamente dicho que apunta al último commit realizado (también llamado `HEAD`).



El *staging area* permite agrupar modificaciones en un mismo `commit`.

Analogía con un fotógrafo:

Stage es preparar la foto, seleccionar qué personas van en la foto y colocar a cada una en su lugar.

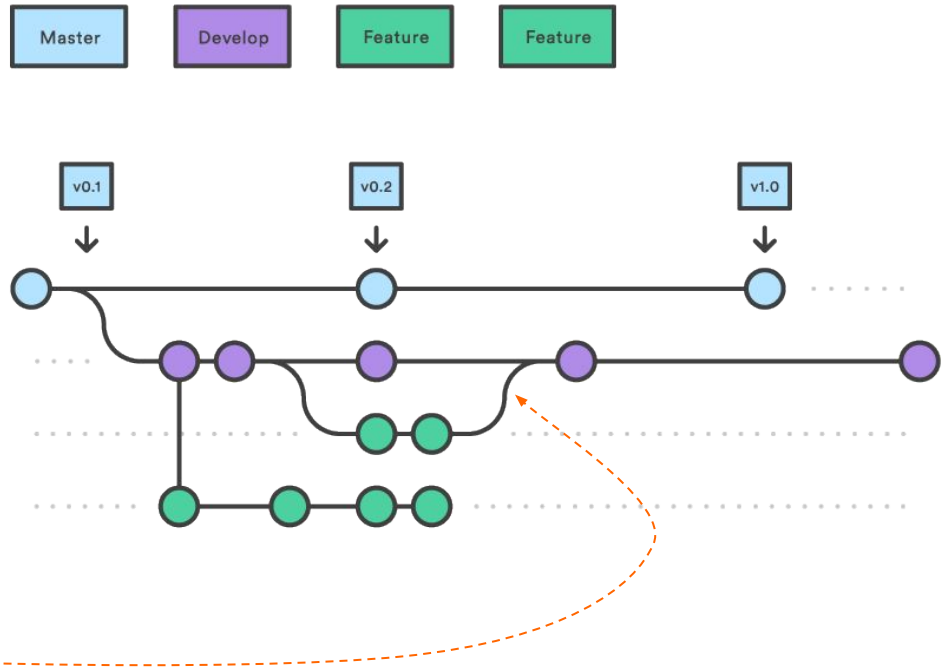
Commit es sacar la foto (snapshot).

Git – Branches

Una de las ventajas de Git es su capacidad para realizar branches.

Cada repositorio cuenta con una branch principal llamada `master`. donde se encuentra el código para producción.

Cada desarrollador puede crear branches paralelas donde desarrollar nuevas funcionalidades. Luego es posible hacer `merge` (combinar) branches.



Git – Comandos Básicos

Crear un proyecto nuevo (crea un directorio para el repositorio):

```
git init mi-proyecto
```

Colocar todos los archivos modificados a un zona de *staging*:

```
git add .
```

Colocar un archivo en la zona de *staging*:

```
git add archivo.txt
```

Un commit.

Grabar una versión histórica de los archivos que están en *staging* (snapshot):

```
git commit -m "Un comentario descriptivo..."
```

Git – Comandos para trabajo colaborativo

Descargar un proyecto existente y toda su historia:

```
git clone [url]
```

Subir todos los commits de cierto branch al repositorio remoto (Ej: Github, Bitbucket):

```
git push origin [branch]
```

Incorporar cambios de repositorio remoto en branch actual:

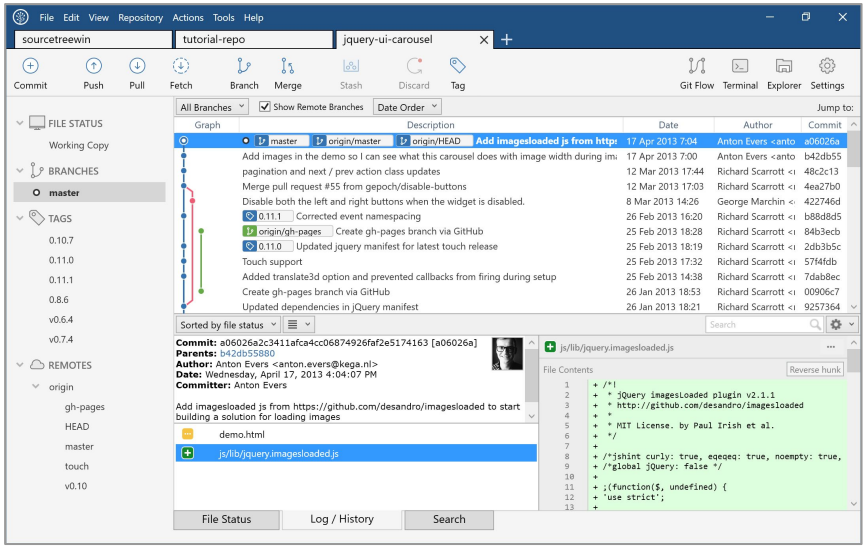
```
git pull
```

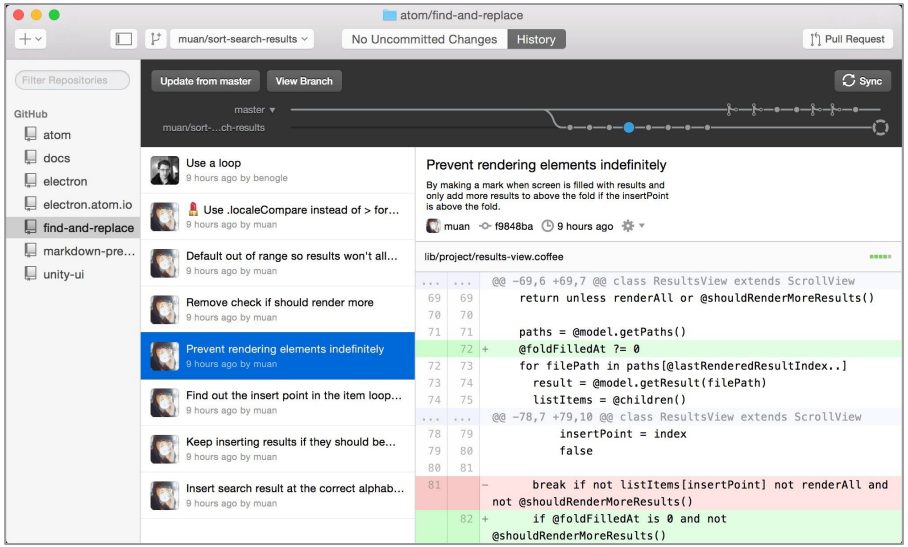
Combinar la historia de cierto branch con el branch actual:

```
git merge [branch]
```

Git – Interfaces Gráficas

Existen interfaces gráficas que facilitan mucho el trabajo con Git, sobre todo para usuarios principiantes. Ejemplos: SourceTree y GitHub Desktop.

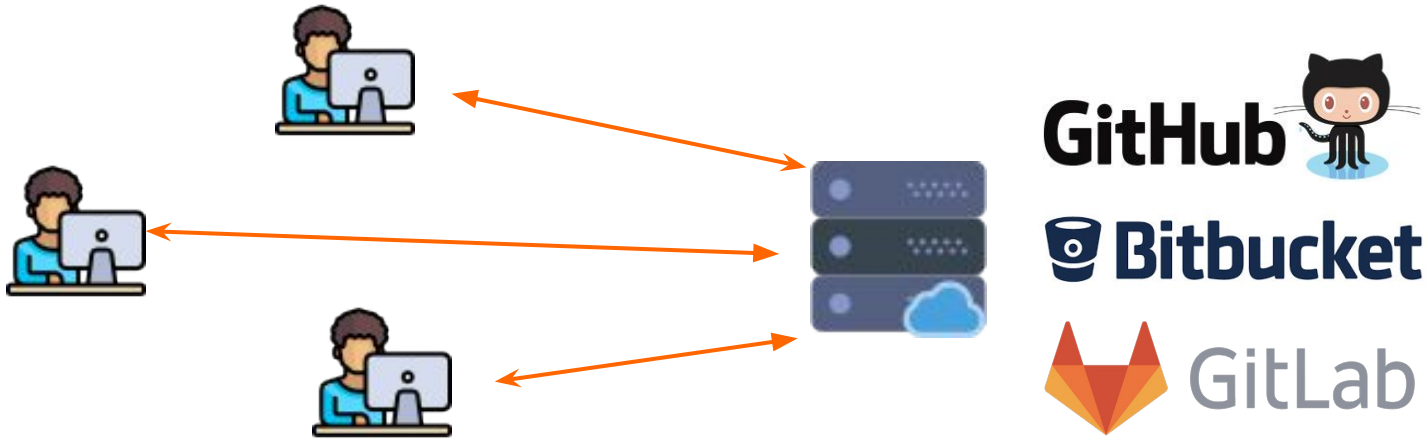




Repositorios remotos en la nube

Repositorios en la nube (1/3)

Como se comentó, por más de que Git sea un VCS distribuido, lo más común es trabajar con un repositorio central. Además, es muy común que dicho repositorio se encuentre alojado en un servicio en la nube como GitHub, Gitlab o Bitbucket.



Este tipo de repositorios también funciona como una gran herramienta de backups.

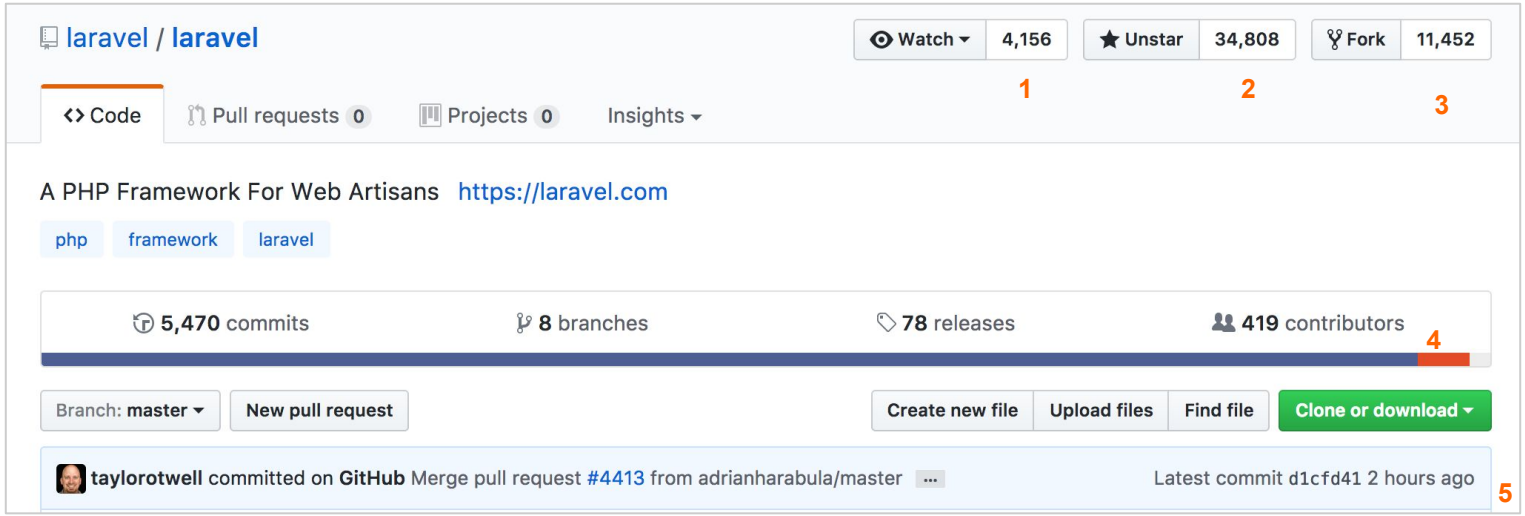
Repositorios en la nube (2/3)

GitHub es el servicio más conocido, sobre todo por la gran cantidad de repositorios de proyectos open source que residen allí. Ejemplos:

- Node.js: https://github.com/nodejs/node.
- React: https://github.com/facebook/react.
- Laravel: https://github.com/laravel/laravel.
- Bootstrap: https://github.com/twbs/bootstrap.
- Vue.js: https://github.com/vuejs/vue.

Crear repositorios en GitHub es gratuito, pero hay ciertas limitaciones.

Repositorios en la nube (3/3) – GitHub – Indicadores



1. Cantidad de personas que están pendientes de cambios en el repositorio.
2. Cantidad de estrellas (*Likes*) del repositorio.
3. Cantidad de veces que otros desarrolladores hicieron su propia versión del repositorio.
4. Cantidad de personas que trabajaron en el repositorio.
5. Tiempo transcurrido desde el último commit.