Hada T1:
Control de versiones

git: Un sistema de control de versiones distribuído.

Departamento de Lenguajes y Sistemas Informáticos Universidad de Alicante

Objetivos del tema

- Conocer los conceptos básicos asociados a los *Sistemas de Control de Versiones*.
- Aprender a usar **git** localmente de manera individual.
- Aprender a usar **git** en grupo.
- Conocer y saber usar **git** con ***github***.

El Control de versiones en la práctica

Supongamos la siguiente situación en la que hemos escrito este código:

```
/*
 * Hola mundo.
 * fecha: 27/12/2009
 */

#include <stdio.h>
int main(int argc, char* argv[]) {
    puts("Hola mundo!");
}
```

El Control de versiones en la práctica

Posteriormente hacemos una serie de cambios en él...

```
/*
 * Hola mundo.
 * fecha:22/01/2010
 */

#include <stdio.h>
int main(int argc, char** argv) {
    printf("Hola mundo!");
}
```

Con los cambios realizados ¿qué podemos hacer?:

- Conservar sólo la última versión del archivo.
- Mantener la versión anterior por si los cambios introducen algún fallo
- Conocer quién los realizó (caso de trabajar en grupo).
- Deshacerlos para recuperar la versión anterior (en caso de haberla perdido).
- Aislarlos para enviarselos a otro desarrollador para que los aplique a la versión del archivo que él tiene (*parche*).

¿En qué consiste el control de versiones?

¿Qué nos aportan entonces los Sistemas de control de versiones? (*SCV*)

- La gestión automática de los cambios que se realizan sobre uno o varios ficheros de un proyecto.
- Restaurar cada uno de los ficheros de un proyecto a un estado de los anteriores por los que ha ido pasando (no solo al inmediatamente anterior).
- Permitir la colaboración de diversos programadores en el desarrollo de un proyecto.

Conceptos generales de los SCV I

- **Repositorio**: Es la copia maestra donde se guardan todas las versiones de los archivos de un proyecto. En el caso de git se trata de un directorio. Cada desarrollador tiene su propia copia local de este directorio.
- **Copia de trabajo**: La copia de los ficheros del proyecto que podemos modificar.

Conceptos generales de los SCV II

Check Out / Clone: La acción empleada para obtener una copia de trabajo desde el repositorio. En los scv distribuidos -como Git- esta operación se conoce como *clonar* el repositorio porque, además de la copia de trabajo, proporciona a cada programador su copia local del repositorio a partir de la *copia maestra* del mismo.

Check In / Commit: La acción empleada para llevar los cambios hechos en la copia de trabajo a la copia local del repositorio. Esto crea una nueva *revisión* de los archivos modificados. Cada commit debe ir acompañado de un “Log Message” el cual es un comentario, una cadena de texto que explica el commit, que añadimos a una revisión cuando hacemos el commit.

Conceptos generales de los SCV III

- **Push**: La acción que traslada los contenidos de la copia local del repositorio de un programador a la copia maestra del mismo.
- **Update/Pull/Fetch+Merge/Rebase**: Acción empleada para actualizar nuestra copia local del repositorio a partir de la copia maestra del mismo, además de actualizar la copia de trabajo con el contenido actual del repositorio local.
- **Conflicto**: Situación que surge cuando dos desarrolladores hacen un *commit* con cambios en la *misma región del mismo fichero*. El **scv** lo detecta, pero es el programador el que debe corregirlo.

Un poco de historia

- SCCS implementación libre de GNU, RCS
- Cvs, Subversion
- BitKeeper
- Bazaar, bzr
- mercurial,
- monotone, darcs, Perforce
- Git es el que usaremos a lo largo de la asignatura.

Git: Historia

- Los desarrolladores de *linux* emplean BitKeeper hasta 2005.
- BitKeeper es un **scv** distribuído. Git también lo es, al igual que Darcs, Mercurial, SVK, Bazaar y Monotone.
- Linus comienza el desarrollo de *git* el 3 de abril de 2005, lo anunció el día 6 de abril.
- Git se auto-hospeda el 7 de abril de 2005.
- El primer núcleo de *linux* gestionado con *git* se libera el 16 de junio de 2005, fue el **2.6.12**.
- ¿Qué significa *git* ?... pues depende, según el propio Linus puede ser:
  1. “I'm an egotistical bastard, and I name all my projects after myself. First Linux, now git.".
  2. *Global Information Tracker*.

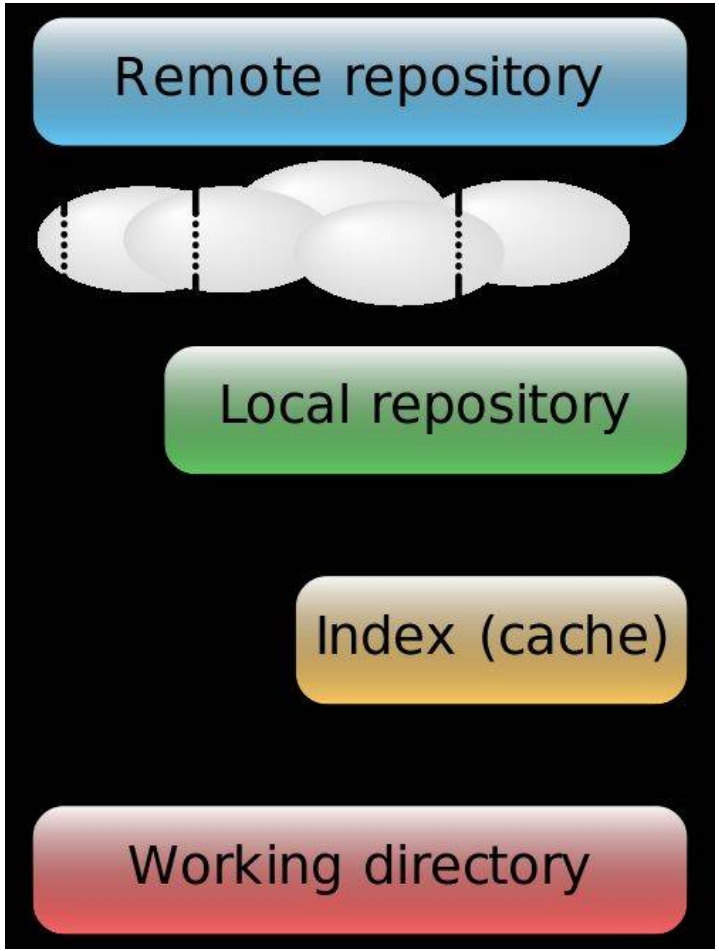
Git: Implementación

- La parte de bajo nivel (plumbing) se puede ver como un sistema de ficheros direccionable por el contenido.
- Por encima incorpora todas las herramientas necesarias que lo convierten en un **scv** más o menos amigable (porcelain).
- Cuenta con aplicaciones escritas en **C** y en **shell-script**. Con el paso del tiempo algunas de estas últimas se han reescrito en **C**.
- Los elementos u objetos en los que *git* almacena su información se identifican por su valor SHA-1.

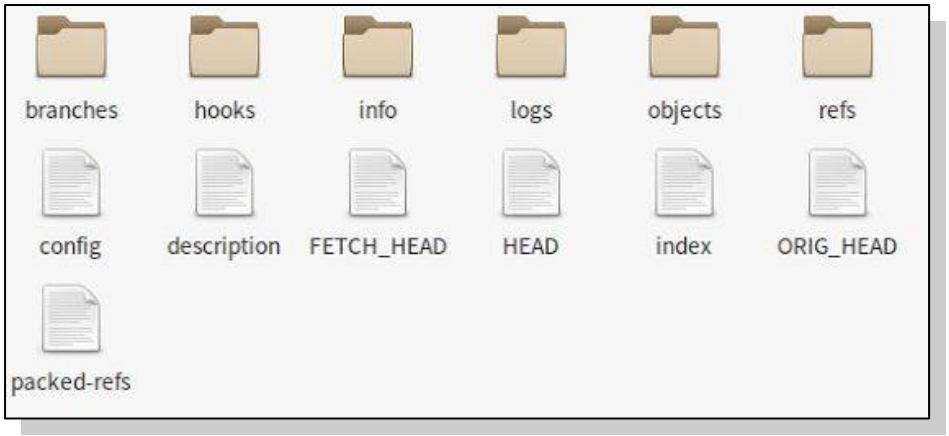
Uso I

- La orden prinicipal es **git**.
  - Comprobamos qué versión tenemos instalada:

    ```
    > git --version
    git version 2.8.0
    ```
- Creamos un repositorio, se puede hacer de dos modos:
  1. `> mkdir Proyecto; cd Proyecto; git init`
  2. `> git init Proyecto`
- Añadimos archivos y guardamos:
  1. `> git add .`
  2. `> git status`
  3. `> git commit -m 'Primer commit.' -m "Descripcion detallada."`
  4. `> git commit -a`


Remote repository
Local repository
Index (cache)
Working directory

Git: Directorio **.git**



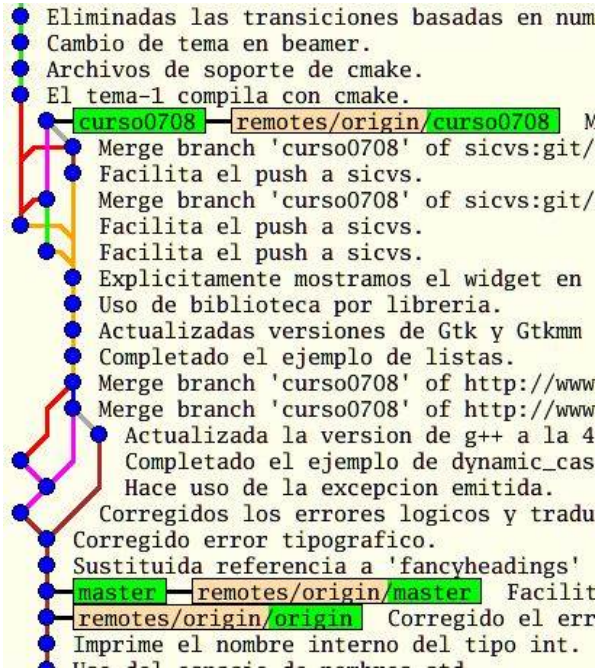
Uso II

- Configuración: archivos “**`.git/config`**” o “**`~/.gitconfig`**”.
- El primero es particular del proyecto actual y el segundo es general para todos los proyectos del usuario.

```
1.  > git config user.name "nombre apellidos"
2.  > git config user.email "usuario@email.com"
3.  > git config --global user.name "nombre apellidos"
4.  > git config --global user.email "usuario@email.com"
```

Uso III

- Ramas
  1. `> git branch [-a] [-r]`
  2. `> git show-branch`
  3. `> git checkout [-b] [rama-de-partida]`
  4. `> git log [-p]`
  5. `> gitk --all`



Uso IV

- Información
  1. `> git status`
  2. `> git log`
  3. `> git show`
  4. `> git diff`
- Descartar cambios
  1. `> git reset --hard`
  2. `> git checkout ruta-archivos o rama`
- Etiquetas
  1. `> git tag tagname commit`
  2. `> git tag -a -m “mensaje” tagname commit`
  3. `> git tag -l`
  4. `> git tag -d tagname`
- Repositorios remotos
  1. `> git remote add nombre protocolo`
  2. `> git remote add origin IP:ruta/hasta/repo`
  3. `> git clone IP:ruta/hasta/repo`
- Operaciones con repositorios remotos
  1. `> git pull [origin] [rama]`
  2. `> git push [repo] [rama]`
  3. `> git checkout -b rama origin/rama-remota`
  4. `> git fetch`
  5. `> git merge`
  6. `> git pull = git fetch + git merge`
  7. `> git rebase otra-rama`

Uso V

- Stash
  1. `> git stash [list | show | drop | ...]`

Echa un vistazo a este tutorial sobre git stash.

- Bisect
  1. `> git bisect [help | start | bad | good | ...]`

Echa un vistazo a este tutorial sobre git bisect.

- Herramientas gráficas
  1. **gitk**
  2. **git gui**
  3. **git view**
  4. **gitg**
  5. **giggle**
  6. **gource**
- Cualquier IDE o editor actual (Atom, Sublime, VisualStudio, VisualStudioCode, etc...) dispone de un plug-in para git.

Interacción con otros SCVs

- Git puede interactuar con otros scvs. Además puede hacerlo en ambos sentidos (recuperar y guardar información).
- Por ejemplo, en los casos de CVS y Subversion echa un vistazo a:
  1. `> git cvsimport --help`
  2. `> git svn --help`

Casos de uso I

- ¿Cómo creo una rama local que *siga* los cambios en una remota al hacer `pull'?
  - **`git branch --track ramalocal origin/ramaremota`**
- ¿Se puede crear una rama que no parta del último commit de otra?...**sí**
  - **`git branch --no-track feature3 HEAD~4`**
- ¿Quién hizo *qué commit* en un fichero del proyecto?:
  - **`git blame fichero`**
- ¿Cómo creo una rama para resolver un bug y lo integro de nuevo en la rama principal?:
  - **`git checkout -b fixes`**
    `hack...hack...hack`
  - **`git commit -a -m "Crashing bug solved."`**
  - **`git checkout master`**
  - **`git merge fixes`**

Casos de uso II

- He modificado localmente el fichero “src/main.cs” y no me gustan los cambios hechos. ¿Cómo lo devuelvo a la última versión bajo control de versiones?:
  - `git checkout -- src/main.cs`
- ¿Y un directorio completo, p.e. a la *penúltima versión* de la rama “test”?:
  - `git checkout test~1 -- src/`
- ¿Y si he modificado varios ficheros y quiero dejar todo como estaba antes de la modificación?...tenemos varias maneras:

1. `git checkout -f`

o también:

2. `git reset --HARD`

Casos de uso III

- ¿Se puede deshacer un `commit' que es una mezcla (*merge*) de varios “commits”?...**sí**, hay que elegir cuál o cuáles de los commits que forman la mezcla así:
  1. **`git revert HEAD~1 -m 1`**
     En este ejemplo estaríamos deshaciendo sólo el primero de los “commits” que formaban este “merge”.
- ¿Cómo puedo obtener un archivo tal y como se encontraba en una versión determinada del proyecto?... de varias maneras:
  1. **`git show HEAD~4:index.html > oldIndex.html`**
     o también así:
  2. **`git checkout HEAD~4 -- index.html`**

Casos de uso IV

- ¿De qué maneras distintas puedo ver los cambios que ha habido en el repositorio?:
  1. `git diff`
  2. `git log --stat`
  3. `git whatchanged`
- ¿Cómo puedo saber cuántos commits ha hecho cada miembro del proyecto en la rama actual?:
  1. `git shortlog -s -n`
- ¿Y en todas las ramas?:
  1. `git shortlog -s -n --all`
- ¿Cómo puedo corregir el mensaje de explicación del último commit que he hecho?:
  1. `git commit --amend`
     Abre el editor por defecto y nos permite modificarlo.

Tutorial interactivo

- En la web de github dispones de uno, se llama Try Git.
- Está pensado para que en unos 15min. pruebes las tareas básicas de git.
- Funciona en el propio navegador.

Ejercicio sencillo

Lo mejor es que pruebes git con algún código tuyo, p.e. de alguna práctica de otra asignatura que ya tengas hecha. Sigue estos pasos:

1. Elige un directorio que contenga el código de esa práctica. Cambiate a él.
2. Inicia el repositorio en este directorio.
3. Añade los archivos que haya previamente en él.
4. Haz el primer "*commit*" de los archivos recién importados.
5. Haz una modificación a uno o varios de ellos. Comprueba cuáles han cambiado, cómo lo han hecho. Añádelos al siguiente commit.
6. Contribuye los cambios creando el "commit".
7. Crea una rama en el proyecto y cámbiate a ella automáticamente (mira las opciones de `checkout`).
8. Haz cambios y commits en esta rama.
9. Vuelve a la rama "master".

Git y Github I

- Github es una plataforma de desarrollo colaborativo para alojar proyectos software haciendo uso del SCV Git.
- Podemos crear proyectos en ella y almacenarlos de manera gratuita si son “públicos” (*todo el mundo puede clonarlos*) pero también permite crear proyectos “privados” haciendo uso de una *cuenta de pago*.
- Desde hace un tiempo Github permite hacer uso gratuito de ciertos recursos que normalmente forman parte de la *cuenta de pago*, p.e. poder crear proyectos “privados”.
- Sólo necesitas una cuenta de email institucional de la UA (@alu.ua.es) asociada a la cuenta de Github empleada y darte de alta en Github-Education.

Git y Github II

- El código de la práctica en grupo deberá estar almacenado en una cuenta de github como un proyecto privado.
- Deberás suministrar a tu profesor de prácticas la URL y clave para clonarlo.
- No se tendrán en cuenta las modificaciones que se hagan al proyecto pasada la fecha de entrega final.
- En la práctica en grupo cada componente deberá tener una rama en ese repositorio llamada como su DNI, y trabajará sólo en su rama.
- El coordinador del grupo (**y sólo él/ella**) será el/la encargado/a de integrar los *commits* correspondientes de cada integrante del proyecto a la rama *master*.
- En este enlace dispones de la ayuda de github y en este otro una serie de guías sobre cómo usarlo, entenderlo mejor y sacarle el máximo partido.
- El **acceso por ssh a github** se configura según esta guía.

Webs de interés

1. git
2. git guide
3. Carl's Worth tutorial
4. git-for-computer-scientists
5. gitmagic
6. freedesktop
7. gitready
8. progit
9. winehq
10. Presentación en vídeo de git hecha por *Linus Torvalds*
11. Vídeo de uso de git/github desde Windows con el intérprete de órdenes y con Visual Studio Code y Visual Studio 2015.