Proyecto Software

Índice

Capítulo 1

Gestión de configuraciones y *Git*

Git es un sistema de control de versiones (VCS). Permite seguir cambios en el proyecto y permite trabajar con diferentes versiones de una aplicación. También permite controlar quién tiene permitido modificar qué partes del proyecto, y automatizar la construcción del software.

1.1 Repositorios

Un proyecto de git tiene:

- Un repositorio (`.git`, que contiene la historia del proyecto).
- El directorio/árbol de trabajo (donde está la copia del repositorio).
- El área de preparación o índice (dentro de `.git`, con los cambios del proyecto a comprometer).

```
# Inicializar nuevo repositorio en el directorio actual
git init

# Clonar repositorio existente, ya sea local, desde una URL acabada en .git o
# desde un servidor SSH
git clone <DIRECCIÓN>
```

1.2 Rastrear ficheros

Los ficheros del directorio de trabajo pueden estar rastreados (*tracked*) o no (*untracked*). Los ficheros se pueden ignorar (indicados en el fichero `.gitignore`).

Los ficheros rastreados pueden estar comprometidos (*committed*, sin cambios desde último *commit*), modificados (*modified*) o preparados (*staged*, listos para ser comprometidos)

```
# Añadir fichero a rastrear.
git add <FICHERO>

# Dejar de rastrear un fichero (sin eliminarlo).
git rm --cached <FICHERO>
```

1.3 Comprometer modificaciones

En el repositorio, se almacenan instantáneas del proyecto (formada por *commits*). Git contiene el estado de todos los ficheros en dicho momento. Los commits están identificados por un *hash* que sirve para verificar su integridad.

```
# Comprometer ficheros preparados.
git commit -m <MENSAJE>

# Con -a añade ficheros rastreados y modificados al commit automáticamente
git commit -a ...
```

```
# Añadir ficheros preparados a commit anterior.
git commit --amend ...
```

Si se modifica un fichero pero se quiere desistir de estos cambios:

```
# Descartar cambios de fichero preparado sin modificarlo.
git restore --staged <FICHERO>

# Restaurar fichero a estado desde último commit (se pierden datos).
git checkout -- <FICHERO>
```

Alternativamente, se puede deshacer cambios entre commits:

```
# Realiza un commit que deshace los cambios del anterior
git revert HEAD

# Vuelve a un commit anterior eliminando posteriores (se pierden datos).
git reset --hard <HASH>
```

1.4 Ramas

Las ramas son bifurcaciones del historial de *commits* a partir de uno. Cada rama apunta a un *commit* (normalmente el más actualizado), y la rama de trabajo se llama HEAD. Al crear un repositorio, la rama inicial se llama normalmente *master/main*.

```
# Crear una nueva rama a partir del commit del HEAD.
git branch <NOMBRE>

# Eliminar rama
git branch -d <RAMA>

# Cambiar de rama.
git checkout <RAMA>

# Crear rama y cambiar.
git checkout -b ...

# Cambiar de rama modifica el directorio de trabajo. Si hay cambios en el
# directorio, o hay ficheros preparados, se puede forzar (se pierden datos).
git checkout --force ...
```

Las ramas se pueden volver a unir de varias formas:

- Si la rama actual se fusiona con un descendiente, se realiza un *fast-forward*.
- Si las ramas son paralelas pero no hay conflictos, se realiza un *merge commit* (combina recursivamente los cambios).
- Si hay conflictos, lo indicará y no realizará el *merge*. Además añadirá marcadores dentro de los ficheros afectados para facilitar encontrarlos.

```
# Mezclar los cambios de una rama con otra
git merge <RAMA A FUSIONAR>
```

1.5 Consultar repositorio

Es recomendable utilizar herramientas gráficas para realizar consultas del estado del repositorio, pues normalmente se muestra

```
# Consultar estado del repositorio (rama, commit, cambios preparados, ficheros
# sin rastrear...).
git status

# Consultar diferencias entre versiones (preparadas o comprometidas).
git diff [<HASH 1> <HASH 2>]
```

```
# Consultar historial de commits.
git log

# Consultar cambios entre ramas
git diverge
```

1.6 Repositorios remotos