

LECCIÓN 2. PRÁCTICA 2

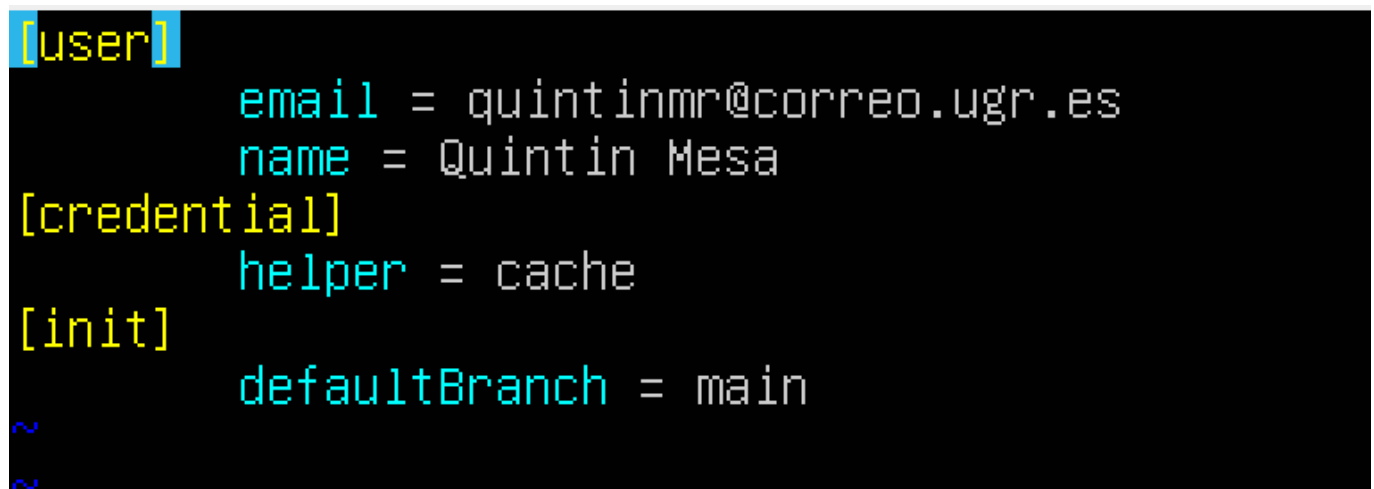
Copias de seguridad y control de versiones

Vamos a centrarnos en el control de versiones con **Git** en local.

1. Abrimos una terminal y creamos un repositorio:

```
git init repositorio
```

2. Vamos a editar el fichero de configuración `~/.gitconfig` (es un archivo que está en el directorio home).



```
[user]
    email = quintinmr@correo.ugr.es
    name = Quintin Mesa
[credential]
    helper = cache
[init]
    defaultBranch = main
~
~
```

3. Creamos nuestro primer archivo:

```
touch libro.txt
```

4. El comando para tener información sobre el estado del repositorio es:

```
git status
```

Nos informará que estamos en la rama main y que no tenemos ningún commit todavía.

5. Para "echar en carrito de la compra" el archivo que hemos creado:

```
git add libro.txt
```

6. Para poder guardar lo que le hemos dicho al sistema que gestione (libro.txt), hacemos:

```
git commit -m "mensaje significativo"
```

El commit lo que hace tomar una fotografía de la situación actual del repositorio y guardarla.

7. Con `git log` se nos muestra una bitácora de los commits que vamos haciendo (la opción `graph` nos muestra un `graph`):

```
git log
```

8. Si queremos evitar que al editar un archivo, al hacer git status, se nos muestre como editado en el estado, tenemos que editar el archivo **.gitignore**, especificando los archivos que no queremos que se tengan en cuenta para ser commiteables.

```
vi .gitignore
```

```
Dentro: nombre_archivo *.o otros
```

9. Si volvemos a hacer git status, ya no nos sale el archivo que hemos dicho que ignore, pero sí el propio archivo .gitignore. Podemos decirle que se ignore a sí mismo.

10. Editemos el archivo libro.txt. Si hacemos

```
git diff
```

Nos indica que tenemos la diferencia con respecto a lo que hay en el repositorio. Si añadimos la modificación al working directory,

```
git add libro.txt
```

Y volvemos a hacer git diff, ¿con quién compara? compara el working directory con lo que hay en la zona de seguimiento, y si no hay nada en la zona de seguimiento, lo compara con el Head.

11. Si hacemos

```
git diff HEAD
```

Nos compara el working directory con el repositorio; con el puntero al último commit.

12. Si lo que queremos es comparar la zona de seguimiento con el repositorio:

```
git diff --staged
```

13. Hagamos el seguimiento de estas modificaciones:

```
git add libro.txt
```

Hacemos commit para confirmar los cambios:

```
git commit -m "añado titulo y subtítulo"
```

14. Para trabajar con ramas:

1. Creación de ramas:

```
git branch rama o git checkout -b rama (crea rama y te cambia a ella)
```

2. Listar ramas

```
git branch (locales) o git branch -a (locales y remotas)
```

3. Para cambiar de rama:

```
git checkout rama
```

4. Para borrar una rama:

```
git branch -D rama
```

Cuando creamos ramas, se crean espacios de trabajo distintos. De esta forma, modificaciones en los archivos de un repositorio, desde una rama, no implican modificaciones en el resto de ramas. A menos que se haga un merge.

15. Para unir ramas:

El comando rebase se actualiza; se cambia la base de la rama y se mantiene de forma lineal el historico de los commit, pero en caso de ramas remotas se desaconseja su uso porque cuando se hacen cambios en el branch, cuando se integran, pueden verse descartados ciertos commits.

```
git rebase
```

Con git merge hay que irse a la rama principal y hacer

```
git merge rama
```

Si hay conflictos, resolver conflictos de archivos.

16. El rebase tiene que hacerse desde la rama que vamos a utilizar para cambiar la base. Tenemos que decirle que se lleve el Head del main a la rama en cuestión. Luego, ir al main y hacer merge:

```
git rebase main (desde rama tuya) git merge rama (desde main)
```

17. Deshacer cambios:

1.

```
git restore --source =identificador_commit archivo
```

Deshace los cambios y vuelve al commit que le hemos pasado. El working directory ha cambiado, pero a nivel de historia no ha cambiado nada.

2. Si queremos volver al último commit:

```
git restore
```

git restore no deja huella.

3. Si queremos dejar constancia de esa vuelta atrás, lo más recomendable es utilizar git revert (crea un nuevo commit a partir de los cambios anteriores):

```
git revert identificador_commit
```

4. Si queremos ser más agresivos, usamos git reset, eliminando la historia:

```
git reset --hard identificador_commit
```

Elimina todos los commits posteriores al especificado y además nos hemos ido atrás en la historia.