Práctica 1: Entorno de Desarrollo

Íñigo de Oñate Cruz

Git y Github

Una vez configurada la cuenta procedemos a utilizar los siguientes comandos sobre el repositorio indicado (https://github.com/gitt-3-pat/p1):

- ***git clone***

Este comando copia un repositorio remoto completo a nuestro local, incluyendo su historial de versiones y es útil para trabajar sobre proyectos ya comenzados. De tal forma que ejecutando:

```
git clone https://github.com/gitt-3-pat/p1
```

Tenemos por salida:

```
Cloning into 'p1'...
remote: Enumerating objects: 15, done.
remote: Counting objects: 100% (15/15), done.
remote: Compressing objects: 100% (10/10), done.
remote: Total 15 (delta 2), reused 12 (delta 1), pack-reused 0
Receiving objects: 100% (15/15), 4.56 KiB | 4.56 MiB/s, done.
Resolving deltas: 100% (2/2), done.
```

- ***git status***

Este comando muestra el estado de nuestro repositorio de Github, es útil para comprobar la rama actual o si hay archivos modificados pero no preparados para confirmar. De tal forma que al ejecutar:

```
git status
```

Observamos:

```
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
```

- ***git add***

Este comando añade los cambios realizados en los archivos especificados al área de preparación (staging área), preparándolos para ser confirmados. Podemos especificar qué archivos incluir para el próximo *commit* para ello podemos hacerlo con el código:

```
git add <archive>
```

Al querer incluir todos los archivos ejecutaremos:

```
git add .
```

A diferencia de los anteriores, no genera una salida visible si se ejecuta correctamente.

- ***git commit***

Este comando guarda los cambios añadidos al área de preparación (staging area) en el historial del repositorio, permitiendo, mediante un mensaje descriptivo, mantener un registro claro de las modificaciones realizadas.

A modo de ejemplo, modificamos el README.md, (ya que sino no aparecerá ningún cambio) ejecutando así:

```
git commit -m “Update README.md”
```

Obteniendo:

```
[main 2813d98] Update README.md
 1 files changed, 92 insertions(+), 16 deletions(-)
```

A modo de buena praxis se recomienda que el mensaje del *commit* sea claro con el objetivo de reflejar los cambios realizados y, así, en caso de ser revisado el historial del repositorio todo sea más ágil.

Cabe mencionar que si no usáramos -m Git abriría el editor de texto predeterminado para escribir el mensaje del *commit*. En definitiva, es un atajo a usar por comodidad.

- ***git push***

Este comando nos permite enviar los *commits* desde el repositorio local al remoto, sincronizando ambos. Antes de realizar el push, Git verifica que los cambios locales no entren en conflicto con los del repositorio remoto.

Si hay conflictos, será necesario (antes del push) sincronizar con:

```
git pull
```

Si fuera necesario sobreescribir cambios, se puede usar:

```
git push --force
```

Esto debe hacerse con precaución para evitar la pérdida de código. Conociendo esto, al enviar los *commits* desde la rama main del repositorio local al repositorio remoto (configurado como origin), ejecutamos:

```
git push origin main
```

Tras esto, podemos observar:

```
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
```

```
Writing objects: 100% (3/3), 1.54 KiB | 1.54 MiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/ionatecrz/Practica1.git
   07720b5..2813d98  main -> main
```

- ***git checkout***

Este comando permite cambiar de una rama a otra, restaurar archivos específicos o retroceder a una versión anterior del repositorio. Al ejecutar:

```
git checkout -b development
```

Podemos observar la salida:

```
Switched to a new branch 'development'
```

De esta forma Podemos crear otras ramas como la llamada *development* para, por ejemplo, trabajar en nuevas funcionalidades del código sin afectar la rama principal *main*. Esto facilita el desarrollo en paralelo de versiones y el control de las mismas.