**Actividad: Comandos de Git**

Para cada paso de la práctica, indicar el comando empleado.

Tras cada comando ejecutado que produzca alguna modificación en el repositorio, comprobar el estado del repositorio, explicando el resultado de la ejecución de dicho comando.

Siglas:

* SA: Staging area o Área de preparación/montaje.
* WD: Working directory o Directorio de trabajo.

Toda la práctica se realizará en un directorio **reposgit**, que no es un repositorio Git en sí mismo, donde a su vez se creará algún directorio que sí es un repositorio Git.

Al final de la práctica, en la sección de ANEXOS, aparecen listados los comandos, tanto de Git, como de Linux, que se usan en esta práctica.

\* NOTA: En varios de los apartados se pide que se obtenga un cierto resultado “con un solo comando”. Habitualmente esto quiere decir que hay que concatenar en una misma línea varios comandos para obtener el resultado. En Linux existen operadores para ello como tubería (*pipe*) **|** o redireccionamiento **>** , **>>**, **<** o **<<**.

**Trabajo con la clonación de un repositorio ya existente**

1. Clonar el repositorio ubicado en https://github.com/grayghostvisuals/practice-git.

|  |
| --- |
| **$ git clone https://github.com/grayghostvisuals/practice-git.git**  Cloning into '**practice-git**'...  remote: Enumerating objects: 639, done.  remote: Total 639 (delta 0), reused 0 (delta 0), pack-reused 639  Receiving objects: 100% (639/639), 565.61 KiB | 2.49 MiB/s, done.  Resolving deltas: 100% (239/239), done. |

1. Mediante la ejecución de un solo comando (una sola línea), compuesto de un comando Git para listar archivos, usado conjuntamente con el comando de wc de Linux, indicar cuántos archivos contiene el proyecto en su WD. Se deben incluir aquellos archivos que haya en subdirectorios. El directorio .git no se considera un subdirectorio del proyecto.

Hacerlo de diferentes maneras, al menos de dos.

|  |
| --- |
| **$ git log --oneline | wc -l**  271  **$ git rev-list --count HEAD**  271 |

1. Mostrar la lista de contribuyentes/colaboradores del repositorio, así como el número de commits que ha hecho cada uno.

|  |
| --- |
|  |

1. Mostrar, con un comando, los nombres de los contribuyentes/colaboradores que hay en el repositorio.

|  |
| --- |
| git shortlog -sc |

1. Mostrar, con un comando, el número de contribuyentes/colaboradores que hay en el repositorio.

|  |
| --- |
| **git ls-remote | wc -l** |

1. Mediante un comando de Linux, usado conjuntamente con el comando usado anteriormente, mostrar el número de commits existentes en el repositorio. Se trata de hacer la suma de los números que hay dentro de un archivo o flujo de caracteres.

|  |
| --- |
| git rev-list --count HEAD |

1. Mostrar el histórico de commits en el repositorio, en modo de un commit por línea, con salida gráfica y para todas las ramas del repositorio.

|  |
| --- |
| git log --oneline --all |

1. Mostrar los nombres de ramas que existen.

|  |
| --- |
| git branch |

1. Mostrar el estado del repositorio y explicar la información que se obtiene.

|  |
| --- |
| git status |

1. Mostrar las direcciones de los servidores en la nube donde se almacena el repositorio, tanto para subir contenidos (push), como para descargarlos (fetch o pull).

|  |
| --- |
| git remote -v |

1. Mostrar los commits realizados por el contribuyente grayghostvisuals durante el año 2019.

|  |
| --- |
| git log --author=grayghostvisuals --since=2019-01-01 –-until=2019-12-31 |
|  |

**Trabajo con un nuevo repositorio actividad**

1. Crear, dentro del directorio reposgit, un repositorio de nombre actividad.

|  |
| --- |
| git init actividad |

1. Mostrar el histórico de commits del repositorio. Explicar el resultado obtenido.

|  |
| --- |
| git log |

1. Mostrar los nombres de los contribuyentes al repositorio con el número de commits que ha realizado cada uno. Explicar el resultado obtenido.

|  |
| --- |
| git shortlog -s |

1. Mostrar la información disponible del último *commit*. Explicar el resultado obtenido.

|  |
| --- |
| git show |

1. Mostrar el nombre de las ramas existentes. Explicar el resultado obtenido.

|  |
| --- |
| git branch |

1. Crear una nueva rama de nombre bifurca. Explicar el resultado obtenido.

|  |
| --- |
| git branch bifurca |

1. Mostrar las direcciones de los servidores en la nube donde se almacena el repositorio, tanto para subir contenidos (push), como para descargarlos (fetch o pull).

|  |
| --- |
| git remote -v |

1. Verificar la conectividad y validez de los objetos de la base de datos de Git. Explicar el resultado obtenido.

|  |
| --- |
|  |

**A partir de aquí, se irán realizando operaciones, tras cada una de las cuáles se irá comprobando el estado del repositorio.**

1. Crear un archivo de nombre **letras** que contenga la letra **a**.

|  |
| --- |
| echo a > letras  git status |

1. Añadir el archivo letras al SA. ¿Ha cambiado el contenido del archivo?

|  |
| --- |
| git add letras  git status |

1. Ejecutar el comando git rm letras. Explicar la respuesta obtenida.

|  |
| --- |
| git rm letras  git status |

1. Sacar el archivo del SA (*unstage*).

|  |
| --- |
| git rm -f letras  git status |

1. Añadir de nuevo el archivo al SA. Realizar un *commit* con el mensaje *Creado el archivo* ***letras***.

|  |
| --- |
| git add letras  git commit -m “Creado el archivo letras.”  git status |

1. Mostrar los nombres de los archivos contenidos en este *commit* recién creado. Únicamente los nombres.

|  |
| --- |
| git ls-tree --name-only HEAD |

1. Mostrar los nombres de los archivos contenidos en el *commit* recién creado, con sus SHA y su tipo.

|  |
| --- |
| git ls-tree -r HEAD |

1. ¿Cuál es el SHA del archivo **letras** en el WD?

|  |
| --- |
| git hash-object letras |

1. ¿Cuál es el SHA del archivo **letras** en el SA?

|  |
| --- |
| git ls-files --stage letras |

1. Añadir la letra **b** al archivo **letras**. Subir el archivo al SA. Sacar el archivo del SA. ¿Ha cambiado el contenido del archivo?

|  |
| --- |
| echo b >> letras |

1. Añadir de nuevo el archivo **letras** al SA. Añadir la letra **c** al archivo **letras**. Comprobar el estado del repositorio. Hacerlo de nuevo con el comando **git status -s**. Explicar la salida devuelta por ambos comandos de comprobación del estado del repositorio.

|  |
| --- |
| git add letras  git status  echo c >> letras  git status |

1. ¿Qué sucede si se ejecuta el comando **git restore letras**?

Si ejecutas el comando git restore letras, se descartarán los cambios realizados en el archivo "letras" y se restaurará al estado del commit más reciente.

1. ¿Qué sucede si se ejecuta el comando **git restore --staged letras**?

Si ejecutas el comando git restore --staged letras, el archivo "letras" volverá al estado en el Working Directory, lo que significa que los cambios que habías añadido al Staging Area se descartarán.

1. Para este apartado, volver al estado en el que se encuentra el repositorio justo detrás del punto 27.

Mostrar un listado del histórico de *commits*.

|  |
| --- |
| git log --oneline |

1. Mostrar los nombres de los contribuyentes al repositorio con el número de *commits* que ha realizado cada uno. Explicar el resultado obtenido.

|  |
| --- |
| git shortlog -sne |

1. Mostrar la información disponible del último *commit*. Explicar el resultado obtenido.

|  |
| --- |
| git show HEAD |

1. Mostrar el nombre de las ramas existentes. Explicar el resultado obtenido.

|  |
| --- |
| git branch |

1. Verificar la conectividad y validez de los objetos de la base de datos de Git. Explicar el resultado obtenido.

|  |
| --- |
| git fsck |

1. Realizar un *commit* con el mensaje *Modificado el directorio* ***letras***.

|  |
| --- |
| git commit -m “Modificado el directorio letras.”  git show |

1. Enmendar el *commit* para que el mensaje sea *Modificado el archivo* ***letras***.

|  |
| --- |
| git commit --amend |

1. Mostrar el contenido del archivo **letras** en el *commit* actual y en el anterior.

|  |
| --- |
| git show HEAD:letras  git show HEAD^:letras |

1. Modificar el contenido del archivo **letras** para que contenga las letras a y c, cada una en una línea, dejando una línea en blanco entre la a y la c.

Crear un archivo **numeros**, insertando una línea con los números impares 1, 3 y 5 (separados por comas), y otra con los pares 2, 4 y 6 (separados por comas).

Añadir ambos archivos al SA. Comprobar el estado del repositorio, en el formato largo y en el corto.

|  |
| --- |
| echo “a\n\nc” > letras  echo “1,3,5\n2,4,6” > numeros  git add letras numeros  git show |

1. Confirmarlos con el mensaje *Modificado el archivo leras y añadido el archivo números*. Explicar el resultado del comando.

|  |
| --- |
| git commit -m “Modificado el archivo letras y añadido el archivo números.”  Git show |

1. Mostrar el contenido del archivo números en el commit actual y en el anterior. Explicar el resultado obtenido.

|  |
| --- |
| Git show HEAD:numeros |

1. Mostrar las diferencias del archivo letras en sus versiones del *commit* anterior y la del actual. Utilizar el operado ~.

|  |
| --- |
| git diff HEAD HEAD~1 |

1. Mostrar las diferencias entre el *commit* anterior y el actual. Utilizar el operado ^.

|  |
| --- |
| Git diff HEAD HEAD^ |

1. Modificar el archivo **numeos** para que tenga: en la primera línea, los números impares del 1 al 9, separados por comas; en la segunda, los números pares 2 y 4 separados por comas; la tercera línea en blanco y la cuarta línea, los números 30, 20 y 10 separados por comas. En un solo comando, añadir el archivo al SA y confirmarlo con el mensaje *Retocado el archivo numeros*.

|  |
| --- |
| git |

1. Mostrar el histórico de confirmaciones en formato de una línea por *commi*t y gráfico.

|  |
| --- |
|  |

1. Eliminar completamente el último *commit* y mostrar el histórico de confirmaciones en formato de una línea por *commit* y gráfico.

|  |
| --- |
|  |

1. Arreglar el árbol no referenciado del apartado anterior (su id es **4b825dc**) y volver a probar el comando **git fsck**.

|  |
| --- |
|  |

1. Verificar la conectividad y validez de los objetos de la base de datos de Git. Explicar el resultado obtenido.

|  |
| --- |
|  |

1. Crear una rama de nombre **ramifica**. Cambiarle el nombre a **bifurca**.

|  |
| --- |
|  |

1. Crear un archivo llamado **simbolos** en **main**. Añadirle, en líneas separadas, los caracteres **!**, **·** y **$**. Cambiar a **bifurca**, ¿existe el archivo símbolos? Explicar la respuesta.

|  |
| --- |
|  |

1. En **bifurca**, añadir al SA el archivo **simbolos**. Comprobar el estado del repositorio. Listar el contenido del WD. Cambiar a **main**. Comprobar el estado del repositorio. Comparar los resultados obtenidos en una y otra ramas. Explicar la respuesta.

|  |
| --- |
|  |

1. En **main**, confirmar con el mensaje *Añadido el archivo simbolos*. Comprobar el estado del repositorio. Comparar los resultados obtenidos en una y otra ramas. Explicar la respuesta. Listar el contenido del WD.

|  |
| --- |
|  |

1. Cambiar a **bifurca**. Comprobar el estado del repositorio. Listar el contenido del WD. ¿Coincide el contenido del WD en una y otra rama? Explicar la respuesta.

|  |
| --- |
|  |

1. En la rama **bifurca**., crear el archivo simbolos. Añadirle, en líneas separadas, los caracteres **%**, **/** y **=**. Comprobar el estado del repositorio.

|  |
| --- |
|  |

1. Añadir **simbolos** al **SA**. Comprobar el estado del repositorio. Confirmar con el mensaje *Añadido el archivo símbolos en bifurca*.

|  |
| --- |
|  |

1. En **bifurca**, mostrar el histórico de *commits*. Cambiar a **main**. Mostrar el histórico de *commits*, ¿coinciden ambos históricos? Explicar la respuesta.

|  |
| --- |
|  |

1. En **main**, mostrar el histórico de *commits* de todas las ramas. Cambiar a **bifurca**. Mostrar el histórico de *commits*, de todas las ramas.

|  |
| --- |
|  |

1. Cambar a **main**. Crear un archivo de nombre **nuevo\_en\_main** que contenga una línea con el texto *una línea*. Añadirlo al SA. Añadir al archivo recién creado el texto *otra\_línea*. Mostrar el estado el simplificado repositorio. Mostrar el contenido del archivo **nuevo\_en\_main**.

|  |
| --- |
|  |

1. En **main**, listar el contenido del WD. Reservar el trabajo en curso. Listar el contenido del WD. Mostrar el estado del repositorio. Mostrar la lista de reservas. Sacar el trabajo reservado al WD. Listar el contenido del WD. Mostrar el estado del repositorio. ¿Coincide el contenido del WD y esl estado del repositorio cuando se ha hecho la reserva y cuando se deshace? Explicar la respuesta.

|  |
| --- |
|  |

1. Fusionar la rama **bifurca** en la rama **main**. ¿Qué sucede?, ¿qué hay que hacer ahora? Ejecutar los comandos necesarios para que no se pierda el trabajo que hubiera en curso, sino que se guarde en el repositorio.

|  |
| --- |
|  |

1. Si la rama bifurca no se fusionó a **main** anteriormente, hacerlo ahora. ¿Qué sucede? Explicar el resultado del comando. Muestra el estado del repositorio. ¿Hay que hacer algo para que el repositorio no tenga “nada pendiente”? En caso afirmativo, hacerlo.

|  |
| --- |
|  |

1. Mostrar el histórico de confirmaciones del repositorio, en formato de un *commit* por línea, gráfico y para todas las ramas.

|  |
| --- |
|  |

1. Mostrar el histórico de confirmaciones del repositorio en los que ha sido modificado el archivo **simbolos**, en formato de un *commit* por línea, gráfico y para todas las ramas.

|  |
| --- |
|  |

1. El archivo **letras** ha sido confirmado tres veces. Mostrar en contenido en la versión que hay en el WD. Recuperar en el WD la versión que del mismo hay en el segundo *commit*. Mostrar el contenido del archivo letras. Mostrar el estado del repositorio.

|  |
| --- |
|  |

1. Descartar completamente la versión recuperada del archivo letras, es decir, que el archivo vuelva a estar como antes de recuperar la versión anterior y no quede rastro de la versión recuperada. Mostrar el contenido del archivo letras. Explicar los pasos que se dan, es decir, los comandos ejecutados.

|  |
| --- |
|  |

1. Crear un nuevo archivo de nombre **mar\_de\_dudas**, que contenga una línea con el texto *No sé qué será de mi vida*. Mostrar el estado del repositorio. Crear un archivo **.gitignore** que contenga como texto únicamente el nombre del archivo recién creado. Mostrar el estado del repositorio. Explicar qué ha sucedido. Confirmar el trabajo que deba ser confirmado. Mostrar el histórico de confirmaciones del repositorio, en formato de un *commit* por línea, gráfico y para todas las ramas.

|  |
| --- |
|  |

**Manejo de Git con GitHub**

De las acciones que se realicen en GitHub se debe hacer una captura.

1. En GitHub, crear un repositorio público y sin archivos, de nombre **actividad**.

|  |
| --- |
|  |

1. Comprobar las conexiones que tiene el repositorio local con repositos remoto. Conectar el repositorio local con el remoto antes creado. Volver a comprobar las conexiones que tiene el repositorio local con repositos remotos.

|  |
| --- |
|  |

1. Descargar el contenido de la rama **main** del repositorio remoto a local. ¿Se descarga? Explicar la respuesta. Si no se pudo realizar la descarga, arrglarlo para que se pueda.

|  |
| --- |
|  |

1. Mostrar el histórico de confirmaciones del repositorio (un *commit* por línea, en modo gráfico y para todas las ramas). Mostrar las referencias existentes en el repositorio remoto.

|  |
| --- |
|  |

1. Subir la rama **bifurca** a remoto. Explicar el resultado devuelto por el comando y la acción realizada. Mostrar las referencias existentes en el repositorio remoto.

|  |
| --- |
|  |

1. Crear un archivo de nombre **comun**, tanto en local como en remoto. En local contendrá el texto *¿Y ahora qué, remoto?* En remoto contendrá el texto *¿Y ahora qué, local?* Añadir cada archivo al repositorio donde fue creado. Sincronizar ambos repositorios para que contengan la misma información. Explicar todos los pasos y el resultado de cada acción usando para esto los comandos que se vienen usando anteriormente: **git log**, **git status**…

En GitHub:

|  |
| --- |
|  |

En local:

|  |
| --- |
|  |

1. Mostrar el histórico de *commits* del repositorio local (una línea por commit, modo gráfico, todas las ramas) y las referencias existentes en el repositorio remoto.

|  |
| --- |
|  |

**Trabajo con un nuevo repositorio clonaactividad**

1. Crear un nuevo repositorio de nombre **clonactividad**, como una clonación del repositorio **LOCAL** de nombre **actividad** que se está usando en la práctica. Entra en el WD del nuevo repositorio **clonactividad**, muestra el estado del repositorio, el histórico de confirmaciones (una línea por commit, modo gráfico, todas las ramas) y los servidores remotos a los que está conectado.

|  |
| --- |
|  |

1. Crear un nuevo archivo **me\_revierto** con el texto *Me voy a revertir*. Confirmarlo. Cambiarle el nombre a **revertiendo**. Confirmarlo. A partir de aquí, comprobar el estado del repositorio tras la ejecución de cada comando. Cambiar el nombre del archivo **me\_revierto** por **revertiendo**. Añadir el cambio al SA. Confirmarlo. Explicar el resultado de las operaciones.

|  |
| --- |
|  |

1. Revertir el último *commit* y mostrar el histórico de *commits*. Explicar las operaciones usando para esto los comandos que se vienen usando anteriormente: **git log**, **git status**…

|  |
| --- |
|  |

**ANEXO I: Listado de comandos de Git que se usan en la práctica** (algunos no es imprescindible usarlos)

|  |  |  |  |
| --- | --- | --- | --- |
| **git init** | **git add** | **git remote** | **git ls-files** |
| **git clone** | **git commit** | **git rm** | **git ls-tree** |
| **git help** | **git diff** | **git mv** | **git ls-remote** |
| **git config** | **git branch** | **git restore** | **git rev-list** |
| **git status** | **git checkout** | **git reset** | **git hash-object** |
| **git log** | **git switch** | **git push** | **git fsck** |
| **git shortlog** | **git stash** | **git pull** |  |
| **git show** | **git merge** | **git fetch** |  |

**ANEXO II: Listado de comandos de Linux que se usan en la práctica**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **ls** | **cd** | **echo** | **pwd** | **rm** | **mv** | **cut** | **wc** | **awk** |