El Shell

El Shell es un intérprete de comandos o aplicación que permite a los usuarios interactuar con el sistema operativo de una computadora y ejecutar comandos para realizar diversas tareas.

El Bash es el Shell más utilizado y se ejecuta en la mayoría de los sistemas operativos basados en Unix, incluidos Linux y macOS. Permite la automatización de tareas y la creación de secuencias de comandos para ejecutar varias operaciones de manera secuencial o condicional.

Los científicos de datos aprovechan el shell y el Bash en su flujo de trabajo para automatizar tareas repetitivas, gestionar entornos y dependencias, procesar grandes volúmenes de datos en lotes y crear pipelines de datos eficientes. Estas capacidades les permiten ahorrar tiempo y esfuerzo, mejorar la eficiencia y automatizar tareas complejas en su trabajo de análisis de datos.

Comandos básicos de Shell

A continuación se presenta un instructivo de los comandos más básicos de Shell que un aspirante a científico de datos debería conocer:

1. `cd` (Change Directory): Cambia el directorio actual a uno especificado.
    - Uso: `cd ruta_del_directorio`
    - Ejemplo: `cd Documentos` (cambia al directorio "Documentos")
2. `pwd` (Print Working Directory): Muestra la ruta del directorio actual.
    - Uso: `pwd`
    - Ejemplo: `pwd` (muestra la ruta del directorio actual)
3. `mkdir` (Make Directory): Crea un nuevo directorio.
    - Uso: `mkdir nombre_del_directorio`
    - Ejemplo: `mkdir Proyecto` (crea un directorio llamado "Proyecto")
4. `rm` (Remove): Elimina un archivo o directorio.
    - Uso: `rm nombre_del_archivo` (para eliminar un archivo)
    - Uso: `rm -r nombre_del_directorio` (para eliminar un directorio y su contenido de forma recursiva)
    - Ejemplo: `rm archivo.txt` (elimina el archivo "archivo.txt")
5. `ls` (List): Lista los archivos y directorios en el directorio actual.
    - Uso: `ls`
    - Ejemplo: `ls` (muestra una lista de archivos y directorios en el directorio actual)
6. `cp` (Copy): Copia un archivo o directorio a una ubicación especificada.
    - Uso: `cp origen destino`

    - Ejemplo: `cp archivo.txt CarpetaDestino` (copia el archivo "archivo.txt" a la carpeta "CarpetaDestino")

7. `mv` (Move): Mueve un archivo o directorio a una ubicación especificada o cambia su nombre.
    - Uso: `mv origen destino`
    - Ejemplo: `mv archivo.txt NuevoNombre.txt` (cambia el nombre del archivo "archivo.txt" a "NuevoNombre.txt")
8. `echo`: Imprime un mensaje en la salida estándar.
    - Uso: `echo "mensaje"`
    - Ejemplo: `echo "Hola, mundo"` (imprime "Hola, mundo" en la salida estándar)
    - También podemos crear archivos. Por ejemplo: `echo "41,M,Yes,No,No,No,Often,Yes" >> mental_health_survey.csv`
9. `nano`: Editor de texto en la línea de comandos para crear y editar archivos.
    - Uso: `nano nombre_del_archivo`
    - Ejemplo: `nano archivo.txt` (abre el archivo "archivo.txt" en el editor `nano`)
10. `cat` (Concatenate): Muestra el contenido de un archivo en la salida estándar.
    - Uso: `cat nombre_del_archivo`
    - Ejemplo: `cat archivo.txt` (muestra el contenido del archivo "archivo.txt")

Recuerda que estos son solo algunos de los comandos más básicos de la shell, pero hay muchos más disponibles según tus necesidades específicas. Te recomiendo los cursos de DataCamp Introduction to Shell e Introduction to Bash Scripting. ¡Buena suerte en tu camino como científico de datos!

Configuración de las credenciales

`git config`

El comando `git config` se utiliza para configurar y mostrar las opciones de configuración de Git en tu sistema. A través de este comando, puedes establecer diversas configuraciones que afectan el comportamiento y la apariencia de Git, así como ver las configuraciones existentes.

Nombre de usuario y dirección de correo electrónico

Antes de poder utilizar Git, es imporante configurar tu nombre de usuario y dirección de correo electrónico. Esta información de identificación que se asociará con tus *commits* y otras operaciones en Git. Esto es importante porque permite a otros colaboradores y herramientas de Git reconocer quién realizó cada cambio en el repositorio.

del nombre de usuario y dirección de correo electrónico

Al ejecutar los comandos siguientes en Git Bash o en el terminal que estés usando:

```
git config --global user.name "Tu Nombre"
git config --global user.email "tu_email@ejemplo.com"
```

Estás estableciendo la configuración global de Git en tu sistema. La opción `--global` asegura que estos valores se apliquen de manera global para todos tus repositorios de Git en esa máquina. Si omites --global, se configurará solo para el repositorio actual.

Reemplaza "Tu Nombre" con tu nombre de usuario real que deseas utilizar en tus commits de Git. Puede ser tu nombre completo o un alias, según tus preferencias.

Reemplaza "tu_email@ejemplo.com" con la dirección de correo electrónico asociada a tu cuenta de GitHub. Es importante usar la misma dirección de correo electrónico que utilizaste para registrarte en GitHub o para identificar tu cuenta en el servicio Git remoto que estés utilizando. Esto asegurará que tus commits se asocien correctamente con tu cuenta.

Nota importante: Estos comandos de configuración se ejecutan una sola vez, generalmente al configurar Git por primera vez en tu máquina. Una vez configurados, Git utilizará estos valores de manera predeterminada para tus futuros commits en todos los repositorios de Git que utilices en tu computadora.

PAT (Personal Access Token)

La configuración y el almacenamiento del PAT de GitHub en Git son necesarios para permitir que Git se autentique automáticamente al interactuar con repositorios remotos en GitHub.

Para establecer la conexión con GitHub y configurar tu PAT:

1. Genera un PAT en GitHub.

2. Configura Git para usar el PAT. En Git Bash, ejecuta el siguiente comando:

```
git config --global credential.helper store
```

Esto configura Git para almacenar de forma segura tus credenciales para acceder a repositorios remotos. Al ejecutar este comando, Git guarda tus credenciales en un archivo encriptado y las recuerda para futuras sesiones.

3. A continuación, ejecuta el siguiente comando para agregar el PAT a la configuración de Git:

```
git config --global user.password PAT
```

Reemplaza "PAT" con el token que copiaste de GitHub.

Ejecuta el siguiente comando para verificar que la configuración se haya guardado correctamente:

```
git config user.password
```

Configuración del repositorio de Git local

`git init`

El comando `git init` se utiliza para iniciar un repositorio local de Git en un directorio específico. Al ejecutar este comando, se crea una nueva instancia de repositorio en ese directorio, que permite realizar el seguimiento de cambios, realizar *commits* y realizar otras operaciones de control de versiones utilizando Git.

Cuando se ejecuta `git init`, se crea una carpeta oculta llamada ".git" en el directorio actual. Esta carpeta contiene todos los archivos y registros necesarios para el funcionamiento interno de Git y para controlar los cambios en el repositorio.

Es importante tener en cuenta que `git init` solo se ejecuta una vez en un directorio para iniciar el repositorio. Después de la inicialización, puedes utilizar otros comandos de Git, como `git add`, `git commit`, `git branch`, etc., para administrar y trabajar con el repositorio.

Cómo configurar el repositorio

A continuación se presentan los pasos para configurar un repositorio de Git local en tu escritorio de Windows utilizando Git Bash:

1. Abre Git Bash en tu escritorio de Windows.

2. Crea la carpeta donde deseas crear tu repositorio. Puedes utilizar el comando `mkdir` seguido del nombre de la carpeta. Por ejemplo:

```
mkdir Mis Documentos/nuevo_proyecto
```

Esto creará una nueva carpeta llamada "nuevo_proyecto" en el directorio actual.

Recuerda que para revisar cuál es tu directorio actual en Git Bash, puedes utilizar el comando `pwd`. Al ejecutarlo te mostrará la ruta completa del directorio en el que te encuentras actualmente.

3. Especifica el directorio actual utilizando el comando `cd` seguido del nombre de la carpeta. Por ejemplo:

```
cd Mis Documentos/nombre_proyecto
```

4. Inicia el repositorio ejecutando el siguiente comando:

```
git init
```

Para verificar si se ha creado la carpeta .git, que está oculta en sistemas operativos como Windows, puedes utilizar el siguiente comando:

```
ls -a
```

El comando `ls` se utiliza para listar el contenido de un directorio, y el argumento `-a` muestra todos los archivos, incluidos los archivos ocultos.

Cómo guardar los cambios en un repositorio

`git add`

Los comandos`git add` y `git commit` son dos comandos fundamentales en Git que se utilizan para realizar cambios y registrarlos en el historial de *commits* del repositorio.

`git add` se utiliza para agregar cambios al área de preparación (*staging area*) en Git. Los cambios pueden ser archivos modificados, nuevos o eliminados. Al agregar cambios con `git add`, estás seleccionando qué modificaciones deseas incluir en el próximo *commit*.

- Para agregar todos los archivos en la carpeta actual:

```
git add .
```

- Para agregar un archivo específico:

```
git add nombre_del_archivo
```

Nota: Para eliminar un archivo del área de preparación sin eliminarlo del directorio de trabajo, utiliza el comando `git rm --cached nombre_del_archivo`.

`git commit`

`git commit` se utiliza para confirmar los cambios preparados en el área de preparación y guardarlos en el historial de confirmaciones del repositorio. Cada *commit* en Git tiene un mensaje descriptivo asociado que explica los cambios realizados.

Para realizar un commit, se utiliza el siguiente comando:

```
git commit -m "Descripción del cambio"
```

Reemplaza "Descripción del cambio" con un mensaje descriptivo que resuma los cambios realizados en el *commit*. Es recomendable utilizar mensajes claros y significativos para facilitar la comprensión de los cambios en el historial.

El estado actual del repositorio

`git status`

El comando `git status` muestra el estado actual del repositorio de Git. Proporciona información sobre los cambios realizados en el directorio de trabajo y el área de preparación (*staging area*).

Uso básico:

```
git status
```

Este comando muestra una descripción clara y concisa del estado actual del repositorio. Indica si hay archivos modificados, eliminados, nuevos o sin seguimiento, y también proporciona instrucciones sobre qué acciones se pueden tomar para actualizar el repositorio.

Ejemplo de salida de `git status`:

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

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)

        modified:   index.html

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        newfile.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

En este ejemplo, git status muestra que el archivo `index.html` ha sido modificado pero no se ha agregado para confirmar los cambios. Además, hay un archivo nuevo llamado `newfile.txt` que aún no se ha agregado al área de preparación.

Recuerda que tanto `git log` como `git status` son comandos muy útiles para obtener información sobre el historial de confirmaciones y el estado actual del repositorio de Git. Puedes utilizar estos comandos regularmente para mantener un seguimiento de los cambios realizados en tu proyecto y tomar decisiones adecuadas en función de esa información.

El historial de *commits*

`git log`

El comando `git log` se utiliza para ver el historial de *commits* en un repositorio de Git. Proporciona información detallada sobre los commits realizados, como el autor, la fecha, el mensaje y el identificador único de cada commit.

A continuación, se explican los elementos clave que se muestran en la salida del comando `git log`:

- Hash del commit: Es una cadena alfanumérica única que identifica de forma exclusiva cada *commit*.
- Autor: Nombre y dirección de correo electrónico del autor que realizó el commit.
- Fecha: Fecha y hora en la que se realizó el commit.
- Mensaje: Descripción breve y significativa del *commit* que proporciona información sobre los cambios realizados.

Uso básico:

```
git log
```

Este comando muestra una lista de commits en orden cronológico inverso, es decir, desde el *commit* más reciente hasta el más antiguo. La salida incluye detalles como el autor, la fecha, el mensaje y el *hash* del *commit*.

Ejemplo de salida de `git log`:

```
commit 8ef5f7a6c3d8398a31b13a2d4bc327ae6348a12d
Author: John Doe <johndoe@example.com>
Date:   Mon Jun 14 15:27:52 2023 +0000

    Added new feature XYZ

commit b6f12d55fe67b8a81bce18e3e1e1676f964e44c9
Author: Jane Smith <janesmith@example.com>
Date:   Fri Jun 10 09:42:17 2023 +0000

    Updated documentation

commit 92c392f367d1f29c7d584976a3f5b019864f2b12
Author: John Doe <johndoe@example.com>
Date:   Wed Jun 08 18:09:35 2023 +0000

    Initial commit
```

Cómo referirse al historial un *commit* específico

Para ver el historial detallado de un *commit*, ejecuta el siguiente comando reemplazando "HASH" con el hash del *commit*:

```
git log HASH
```

En Git, para referirte a un *commit* específico utilizando su hash, generalmente solo necesitas especificar los primeros caracteres del hash que sean suficientes para identificar de manera única ese *commit*. Por lo general, se recomienda utilizar al menos 7-8 caracteres iniciales para una identificación segura del *commit*.

Cómo vincular un repositorio local con tu respositorio remoto en GitHub

`git remote`

El comando `git remote` se utiliza para administrar las conexiones con repositorios remotos. Para vincular un repositorio local a un repositorio remoto utilizando Git debes seguir los siguientes pasos:

1. Crea un repositorio remoto en GitHub.

2. En Git Bash dirígete al directorio de tu repositorio local usando `cd`.

3. Ejecuta el siguiente comando para agregar el repositorio remoto:

```
git remote add origin https://github.com/tu_usuario/mi_proyecto.git
```

4. Renombra el *branch* principal (opcional pero recomendado):

Si el repositorio remoto utiliza "main" como el *branch* principal en lugar de "master" (la convención anterior), ejecuta el siguiente comando para renombrar tu *branch* principal a "main":

```
git branch -M main
```

5. Envía los cambios locales al repositorio remoto:

Ejecuta el siguiente comando para enviar tus cambios locales al repositorio remoto:

```
git push -u origin main
```

Reemplaza "main" con el nombre de tu *branch* local si utilizas un nombre diferente.

¡Listo! Has vinculado con éxito tu repositorio local a un repositorio remoto en GitHub. Ahora puedes realizar operaciones de *push* y *pull* para mantener tus repositorios sincronizados.

Cómo mantener el repositorio local sincronizado con el repositorio remoto

`git push`

Para mantener tus repositorios sincronizados, puedes utilizar las operaciones de *push* y *pull* en Git. El comando `git push` se utiliza para enviar los cambios de tu repositorio local al repositorio remoto. Esto permite que otros colaboradores del proyecto vean y accedan a tus cambios. Los *commits* y los *branches* que hayas creado en tu repositorio local se envían al repositorio remoto.

Uso básico:

1. Asegúrate de haber realizado y confirmado tus cambios locales utilizando git commit.

2. En Git Bash, dentro del directorio de tu repositorio local, ejecuta el comando `git push` seguido del nombre del repositorio remoto y el *branch* que deseas enviar. Por ejemplo:

```
git push origin main
```

Nota importante: Si es la primera vez que realizas un push en esta rama, agrega la opción `-u`:

```
git push -u origin main
```

`git pull`

El comando `pull` se utiliza para obtener y fusionar los cambios del repositorio remoto en tu repositorio local. Básicamente, realiza dos operaciones en conjunto: primero, recupera (*fetch*) los cambios del repositorio remoto y luego los fusiona (*merge*) con tu rama local. Esto asegura que tu rama local esté actualizada con los cambios más recientes del repositorio remoto.

Uso básico:

En tu terminal, dentro del directorio de tu repositorio local, ejecuta el comando `git pull` seguido del nombre del repositorio remoto y el branch que deseas actualizar. Por ejemplo:

```
git pull origin main
```

Git realizará un *pull* para traer los cambios más recientes del repositorio remoto a tu repositorio local. Si hay conflictos entre los cambios locales y los cambios remotos, Git te guiará para resolver los conflictos.

Es importante tener en cuenta que antes de realizar un *push* o *pull*, siempre es recomendable hacer un `git status` para verificar el estado actual de tu repositorio y asegurarte de que no haya conflictos pendientes o cambios sin *commit*.

Cómo trabajar con las ramas

`git branch`

El comando `git branch` se utiliza en Git para trabajar con ramas (*branches*). Los *branches* son versiones paralelas de un repositorio que permiten desarrollar características o solucionar problemas de forma aislada sin afectar el *branch* principal (generalmente llamado "master" o "main").

A continuación, se muestran algunos usos comunes del comando `git branch`:

Listar *branches:* Para ver todas las ramas disponibles en tu repositorio, ejecuta el siguiente comando:

```
git branch
```

Cambiar el nombre de un *branch:* En Git, la opción `-M` en el comando git branch `-M` se utiliza para renombrar la rama actual.

Por lo tanto, si estás en el *branch* "master" y deseas cambiar su nombre a "main", puedes utilizar el siguiente comando:

```
git branch -M main
```

Crear un nuevo *branch*: Puedes crear un nuevo *branch* utilizando el siguiente comando:

```
git branch nombre_branch
```

Reemplaza "nombre_branch" con el nombre que deseas asignar al nuevo *branch*. El nuevo *branch* se creará en el punto actual del historial de *commits*.

Eliminar un *branch*: Si ya no necesitas un *branch*, puedes eliminarlo con el siguiente comando:

```
git branch -d nombre_rama
```

Ten en cuenta que no puedes eliminar el *branch* actual en el que te encuentras. Si deseas eliminar el *branch* actual, primero debes cambiar a otro *branch*.

¡Felicidades! Ya sabes usar Git y GitHub.