

Introducción a Bash

Bash es un lenguaje de comandos para la terminal de Unix usado por defecto en la mayoría de las distribuciones de Linux y algunas de MacOS.

Saber usar bash es importante porque permite ejecutar instrucciones en un sistema sin la necesidad de una interfaz gráfica, para nuestro caso aprender a usar bash será un requerimiento fundamental de satisfacer antes de entrar de lleno en las tareas y rutinas que se llevarán a cabo en los servidores/clusters de AWS.

Movimientos básicos en la terminal

Antes de comenzar a ver las características de bash debemos saber al menos como movernos a través de la jerarquía de carpetas a la que entraremos. Imaginemos que todas las carpetas de nuestro computador están estructuradas de forma jerárquica de acuerdo a lo que contienen, incluyendo los archivos, por ejemplo:

```
Documentos/  
|--- Trabajos/  
|    |___ planilla.xls  
|--- Proyectos Personales/  
|    |--- investigacion.pdf  
|    |___ data.txt  
|--- Descargas/  
|    |___ informe.pdf
```

En esta estructura, si estamos viendo actualmente la carpeta de **Documentos**, necesitaremos ingresar primero a la carpeta de **Proyectos Personales** antes de poder abrir los archivos de **investigación.pdf** y **data.txt**. En una terminal de Linux o MacOS se puede saber la carpeta actual en a la que estamos apuntando (o "la que estamos parados actualmente") mediante el comando `pwd`, esto nos devolverá la ruta completa que referencia la carpeta en a la que apuntamos actualmente en la jerarquía global de carpetas del sistema.

Ahora supongamos que queremos editar una línea del archivo data.txt, para esto primero debemos movernos a la carpeta Proyectos personales que es donde se encuentra este archivo, para poder movernos de carpeta se utiliza el comando cd (Change Directory) seguido de la carpeta a la queremos movernos, de la siguiente forma:

```
cd Proyectos\ Personales
```

Nota: Para retroceder un nivel en la jerarquía de directorios con `cd` podemos escribir `cd ..` (con dos puntos).

Notar que hay un `\` justo después de la palabra Proyectos esto es porque si escribiéramos simplemente `cd Proyectos Personales` el proceso creería que Proyectos sería la carpeta a la que queremos movernos mientras que Personales lo asociaría a algún argumento que le estamos tratando de entregar a cd o algo similar, para indicarle a la terminal que debe leer el espacio como parte de un solo nombre se antecede el espacio con un signo `\`. Una vez ejecutada la línea anterior deberíamos estar sobre la carpeta de `Proyectos Personales`.

Nota: En realidad no es necesario ir avanzando carpeta por carpeta, el comando `cd` admite tanto rutas relativas (con respecto al directorio en el que nos encontramos) como rutas absolutas (con respecto al home o raíz del sistema) por lo que podemos entregarle una sola gran ruta absoluta y movernos directamente a cualquier carpeta dentro del sistema.

Edición de archivos

Ahora que estamos en la carpeta `Proyectos Personales` podemos listar los archivos y directorios que se encuentran en esa carpeta mediante el comando `ls` (list), al ejecutarlo debiésemos observar una lista en la terminal con los archivos y carpetas que se encuentran dentro del directorio en el que nos encontramos.

```
(base)
ignacio@x86_64-conda-linux-gnu /mnt/d/Documentos/Desafio LATAM
% ls
'Big Data'
'Competencias ML module.ipynb'
'Correccion lec3_naivebayes.rtf'
'Correcciones y notas'
'Datasets tentativos actividades finales.txt'
'Desafio LATAM.rar'
'Desafios_01'
'Detalle MachineLearning.pdf'
'Detalles Cambios'
'Drafts'
'Estado DS 2020'
'Fundamentos de Data Science'
'Images'
'Intro a Python'
'Intro to CNN.ipynb'
'IntroToDS'
'Machine Learning'
'Machine Learning G6'
'Maria'
'Proyectos'
'Prueba.ipynb'
'SQL'
'Super resolution example.ipynb'
'Syllabus.ipynb'
'__pycache__'
'cifar-10-batches-py'
'cifar-10-python.tar.gz'
'cifar-100-python'
'cifar-100-python.tar.gz'
'data'
'data-science-bootcamp'
'dslatam-pres'
'dslatam-pres-master'
'full_vin_limpio.csv'
'introcnn.py'
'introcnn_alumnos.ipynb'
'introcnn_alumnos.zip'
'machine-learning-module'
'notas evaluacion contenido.pdf'
'preguntas docentes.pdf'
!1714

(base)
ignacio@x86_64-conda-linux-gnu /mnt/d/Documentos/Desafio LATAM
% cd Machine\ Learning
!1715

(base)
ignacio@x86_64-conda-linux-gnu /mnt/d/Documentos/Desafio LATAM/Machine Learning
% ls
!1716
2009_1perc.csv
2009_1perc.csv
2010_1perc.csv
'Machine Learning Quiz.rar'
'Merged Desafios Machine Learning.pdf'
'Merged Lecturas Machine learning.pdf'
'Programa - Machine Learning.pdf'
'Prueba1'
'Prueba2'
'Revisiones'
'Semana 1 - Regularizacion y Expansiones Basales'
'Semana 2 - Algoritmos de Clasificacion'
'Semana 3 - SVM y EM'
'Semana 4 - Ensemble 1'
'Semana 5 - Ensemble 2'
'Semana 6 - NN part 1'
'Semana 7 - NN part 2'
'Semana 8 - NN part 3'
'figsdectrree.py'
'img'
'quizz_machine_learning.md'
'quizz_machine_learning.pdf'
'text_emotion.csv'
```

Imagen 1. Ejemplos uso comandos `ls` y `cd`.

Se muestra una imagen de ejemplo del uso de `ls` y `cd`, la estructura de los directorios no es la misma que el ejemplo pero los comandos son exactamente los mismos.

Nota: El comando `ls` admite el ingreso de un argumento que permite mostrar los archivos y carpetas ocultas en un directorio, para esto debemos escribir `ls -a`.

Volviendo a nuestro ejemplo, si queremos editar el archivo `data.txt` debemos abrirlo de alguna forma, para esto podemos usar alguno de los editores de texto por terminal pre-instalados en los sistemas tipo Linux, por ejemplo, Vim o Nano.

Para abrir un archivo como texto plano con Nano basta invocar al programa de la siguiente forma:

```
nano data.txt
```

Eso abrirá una ventana en la misma terminal donde se muestre el contenido en texto plano (si es que el tipo de archivo lo permite) donde lo podremos editar.

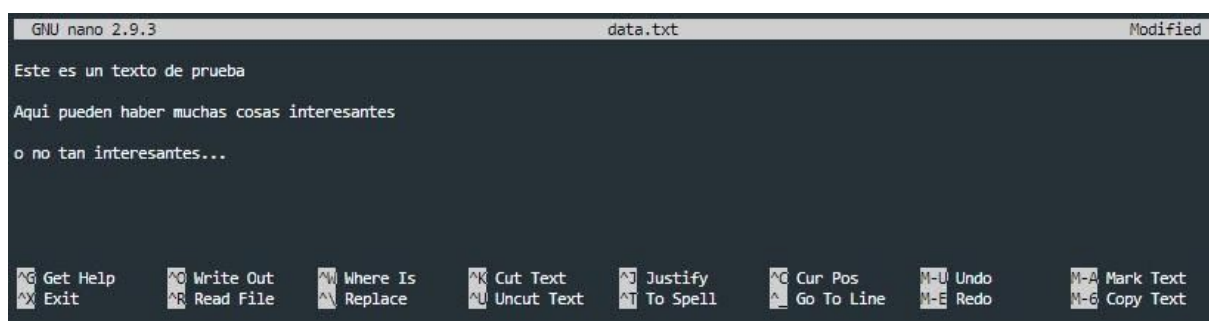


Imagen 2. Ejemplo uso comando `nano`.

Una vez dentro del editor hay que tener en cuenta algunos comandos básicos, los cuales son mostrados en la parte inferior del editor:

- **Guardar Cambios:** `ctrl + o` y luego `enter`.
- **Salir del editor:** `ctrl + x` y luego `enter`.

Con las flechas del teclado podemos mover el cursor y editar lo que necesitemos, luego, debemos guardar los cambios, se nos preguntará si queremos guardar los cambios (mediante un mensaje en la parte inferior del editor) a lo que debemos simplemente presionar `enter`.

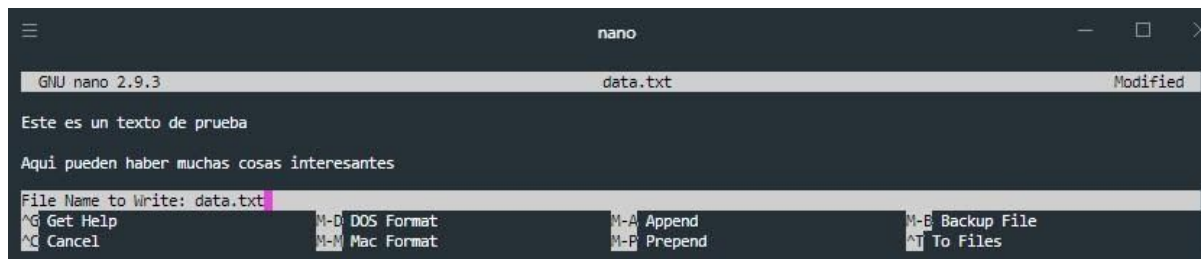


Imagen 3. Ejemplo de guardado.

Al presionar `enter` veremos un mensaje indicando que se guardaron correctamente los cambios en el archivo (De forma análoga, si no queremos guardar los cambios basta usar `ctrl + c` como se muestra abajo en el editor).

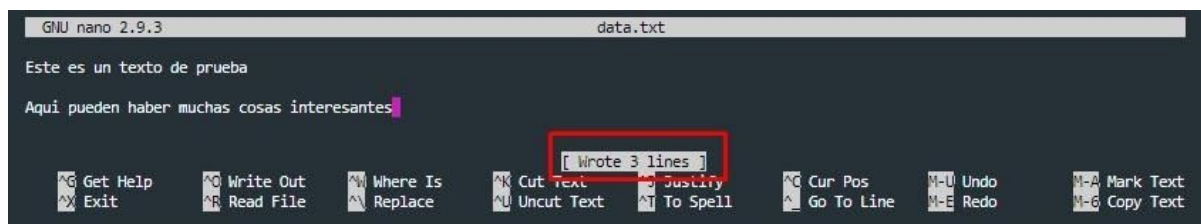


Imagen 4. Ejemplo 3 uso de nano.

Una vez que hayamos guardado nuestros cambios basta usar el comando `ctrl + x` y volveremos a la terminal.

Nota: Si queremos crear un archivo como el que acabamos de editar, con nano, basta abrirlo con la misma sintaxis que usamos antes, si este no existe nano lo creará y nos permitirá editarlo en el lugar, hay que tener en cuenta que nano por defecto creará el archivo en la misma carpeta en la que nos encontramos.

Movimiento, copia y eliminación de archivos

Mover archivos

Para mover un archivo de una carpeta a otra se utiliza el comando `mv` (move) Supongamos que queremos mover el archivo `data.txt` a la carpeta `Trabajos`, hay varias formas de hacer esto, en todas debemos entregarle al comando `mv` la ruta (absoluta o relativa) donde se encuentra el archivo que queremos mover y la ruta (absoluta o relativa) hacia donde queremos moverlo.

En nuestro ejemplo, podemos decir lo siguiente en la terminal (considerando que estamos parados en la carpeta `Proyectos Personales`):


```
mv data.txt ../Proyectos\ Personales
```

La línea anterior le dice a `mv` que debe mover el archivo `data.txt` que se encuentra en la carpeta actual hacia la carpeta `Proyectos Personales` que se encuentra en el directorio previo en la jerarquía (eso estamos referenciando con `".."`) . Adicionalmente también podríamos haber dado la ruta absoluta hacia donde queremos mover `data.txt`, por ejemplo:

```
mv data.txt Documentos/Proyectos\ Personales
```

Copiar Archivos

Copiar carpetas de un directorio a otro sigue exactamente la misma sintaxis que `mv`, con la diferencia que ahora el comando en cuestión es `cp` (copy).



```
(base)
ignacio@x86_64-conda-linux-gnu /mnt/d/Documentos/Desafio LATAM/Proyectos/Revision Contenido
% nano data.txt → Creamos el archivo !1723
(base)
ignacio@x86_64-conda-linux-gnu /mnt/d/Documentos/Desafio LATAM/Proyectos/Revision Contenido
% mv data.txt ../ → Movemos el archivo a la carpeta anterior en la jerarquía !1724
(base)
ignacio@x86_64-conda-linux-gnu /mnt/d/Documentos/Desafio LATAM/Proyectos/Revision Contenido
% ls
'09_g_Apoyo Prueba 2' '09_g_Apoyo Prueba 2.zip' 'Machine SQL'
Nos movemos a la carpeta anterior y verificamos que haya quedado el archivo en esta !1725
(base)
ignacio@x86_64-conda-linux-gnu /mnt/d/Documentos/Desafio LATAM/Proyectos/Revision Contenido
% cd ..
(base)
ignacio@x86_64-conda-linux-gnu /mnt/d/Documentos/Desafio LATAM/Proyectos
% ls
'CreacionContenido' 'G13 Proyecto final' 'Preguntas entrevistas' 'Revision Contenido' 'Revision Entrevistas' data.txt
(base)
ignacio@x86_64-conda-linux-gnu /mnt/d/Documentos/Desafio LATAM/Proyectos
% cp data.txt /Revision\ Contenido/data.txt
cp: cannot create regular file '/Revision Contenido/data.txt': No such file or directory
(base)
ignacio@x86_64-conda-linux-gnu /mnt/d/Documentos/Desafio LATAM/Proyectos
% cp data.txt ../Revision\ Contenido/data.txt → Copiamos el archivo a la carpeta donde estaba originalmente !1729
(base)
ignacio@x86_64-conda-linux-gnu /mnt/d/Documentos/Desafio LATAM/Proyectos
% cd Revision\ Contenido
(base)
ignacio@x86_64-conda-linux-gnu /mnt/d/Documentos/Desafio LATAM/Proyectos/Revision Contenido
% ls
'09_g_Apoyo Prueba 2' '09_g_Apoyo Prueba 2.zip' 'Machine SQL' data.txt
(base)
ignacio@x86_64-conda-linux-gnu /mnt/d/Documentos/Desafio LATAM/Proyectos/Revision Contenido
% !1732
```

Imagen 5. Ejemplo uso de `mv` y `cp`.

Notar que agregamos en la ruta de destino de `cp` un punto antes, el punto seguido de un slash: `./` referencia al directorio actual (como ruta relativa), por lo que el línea `../Revision\ Contenido` esta referenciando a la carpeta `Revision Contenido` que se encuentra **dentro** del directorio actual.

Eliminar archivos

Para eliminar un archivo se utiliza el comando `rm` seguido del nombre del archivo o de la ruta absoluta/relativa al mismo en caso de que no se encuentre en el mismo directorio en el que nos encontramos nosotros. Por ejemplo:

```
rm data.txt
```

o

```
rm Documentos/Proyectos\ Personales/data.txt
```


Variables de Entorno (Environment Variables)

Entenderemos por sesión en el contexto de bash a una ventana de la terminal, en realidad esto es un proceso con su correspondiente memoria asignada y por lo tanto con sus propias variables. Abrir dos sesiones es equivalente a abrir dos terminales.

Las variables de entorno tienen la misma función que las variables que conocemos y hemos estado usando hasta ahora en Python: Sin entidades que almacenan información que requerimos recuperar en un futuro, más específicamente, las variables de entorno en el contexto de bash almacenan las rutas a binarios de programas, opciones de configuración con las que se tiene que iniciar ciertos procesos, etc.

Existen dos tipos de variables de entorno dependiendo de su alcance:

1. Variables locales

Son variables definidas durante la sesión de una terminal y existe en la memoria principal del proceso mientras esta sesión esté activa, una vez que termine ese proceso desaparece del sistema, por ejemplo, podríamos crear una variable local que almacene el nombre del motor de bases de datos al que posteriormente nos conectaremos, para definir esta variable basta utilizar el comando `export` en la terminal de la siguiente forma:

```
export DBMS="POSTGRESQL"
```

La variable `DBMS` almacena un string con el nombre del motor y ahora cuando alguna rutina, ejecutada en nuestra sesión, requiera esa información podrá obtenerla a través de esa variable. Una vez que terminemos el proceso de la terminal (lo que no siempre es equivalente a simplemente cerrar la ventana gráfica de la terminal) esta variable desaparecerá y si la requerimos en algún otro momento tendremos que declararla nuevamente.

2. Variables Globales

Una variable global es definida por defecto al momento de iniciar la terminal y puede ser utilizada a través de todas las sesiones. Para poder crear una variable global esta debe ser definida explícitamente en alguno de los archivos de configuración de bash, por ejemplo, si queremos que la variable del ejemplo anterior sea global y que por lo tanto todas las sesiones puedan accederla y se inicialice siempre al abrir un proceso de la terminal, podemos ir al archivo `.profile` (usualmente en el home del usuario) y escribir, con nano, la misma línea con la que definimos la variable local anteriormente:

```
# ~/.profile  
  
export DBMS="POSTGRESQL"
```

Una vez guardadas las modificaciones al archivo `.profile` cada vez que se inicie un proceso de la terminal se creará la variable `DBMS` con el valor que fijamos.

Se puede consultar fácilmente la lista con todas las variables cargadas actualmente por el proceso escribiendo `env` en la terminal.

La variable PATH

Dentro de las variables disponibles en el sistema, la variable `PATH` juega un papel fundamental, esta variable almacena las rutas a todos los ejecutables y scripts de los programas que usamos normalmente como Python, Conda, Jupyter, etc. Esta variable es global y hace que el sistema revise todos los directorios del sistema especificados en el `PATH` en busca de los scripts que se invocarán. Se puede verificar el valor de la variable `PATH` (y el de cualquier otra variable) utilizando el comando `echo` (es un análogo al `print()` de python) seguido del nombre de la variable que se quiere revisar, para poder referenciar a una variable esto se debe hacer anteponiendo un signo de `$` antes del nombre de la variable:

```
(base)
ignacio@x86_64-conda-linux-gnu /mnt/d/Documentos/Desafio LATAM/Proyectos/Revision Contenido
% echo $PATH
!1734
/mnt/d/ProgramFiles/spark/spark-2.4.5-bin-hadoop2.7//bin:/home/linuxbrew/.linuxbrew/bin:/home/linuxbrew/.linuxbrew/sbin:/home/ignacio/a
aconda3/bin:/home/ignacio/anaconda3/condabin:/home/ignacio/bin:/usr/local/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/
in:/usr/games:/usr/local/games:/mnt/c/Program Files/NVIDIA GPU Computing Toolkit/CUDA/v10.2/bin:/mnt/c/Program Files/NVIDIA GPU Computi
g Toolkit/CUDA/v10.2/libnvvp:/mnt/d/ProgramFiles/Anaconda3:/mnt/d/ProgramFiles/Anaconda3/Library/mingw-w64/bin:/mnt/d/ProgramFiles/Anac
nda3/Library/usr/bin:/mnt/d/ProgramFiles/Anaconda3/Library/bin:/mnt/d/ProgramFiles/Anaconda3/Scripts:/mnt/c/Program Files (x86)/Common
iles/Oracle/Java/javapath:/mnt/c/Windows/system32:/mnt/c/Windows:/mnt/c/Windows/System32/Wbem:/mnt/c/Windows/System32/WindowsPowerShell
v1.0:/mnt/c/Windows/System32/OpenSSH:/mnt/c/Program Files (x86)/NVIDIA Corporation/PhysX/Common:/mnt/c/Program Files/NVIDIA Corporation
NVIDIA NVDLISR:/mnt/c/Program Files/Intel/WiFi/bin:/mnt/c/Program Files/Common Files/Intel/WirelessCommon:/mnt/c/WINDOWS/system32:/mnt/
/WINDOWS:/mnt/c/WINDOWS/System32/Wbem:/mnt/c/WINDOWS/System32/WindowsPowerShell/v1.0:/mnt/c/WINDOWS/System32/OpenSSH:/mnt/c/Program Fil
s/Java/jdk1.8.0_231/bin:/mnt/c/Program Files/NVIDIA Corporation/Nsight Compute 2019.5.0:/mnt/c/Program Files/NVIDIA Corporation/NVIDIA
GX:/mnt/c/WINDOWS/system32:/mnt/c/WINDOWS:/mnt/c/WINDOWS/System32/Wbem:/mnt/c/WINDOWS/System32/WindowsPowerShell/v1.0:/mnt/c/WINDOWS/Sy
tem32/OpenSSH:/mnt/c/Users/ignac/AppData/Local/Microsoft/WindowsApps:/mnt/c/Users/ignac/AppData/Local/hyper/app-3.0.2/resources/bin:/mn
/d/ProgramFiles/Microsoft VS Code/bin:/snap/bin
(base)
```

Imagen 6. Rutas de ejecutables y scripts.

Finalmente, podemos ver las variables que están cargadas en la sesión activa con el comando `env` en la terminal:

```

.ignac/Desktop
(base)
ignacio@x86_64-conda-linux-gnu /mnt/c/Users/ignac
% env
HOSTTYPE=x86_64
_=usr/bin/env
LANG=C.UTF-8
WSL_DISTRO_NAME=Ubuntu-18.04
USER=ignacio
PWD=/mnt/c/Users/ignac
HOME=/home/ignacio
NAME=LAPTOP-R9TDLGRD
XDG_DATA_DIRS=/usr/local/share:/usr/share:/var/lib/snapd/desktop
SHELL=/bin/bash
TERM=xterm-256color
SHLVL=4
LOGNAME=ignacio
PATH=/mnt/d/ProgramFiles/spark/spark-2.4.5-bin-hadoop2.7//bin:/home/linuxbrew/.linuxbrew/b
io/anaconda3/bin:/home/ignacio/anaconda3/condabin:/home/ignacio/bin:/usr/local/bin:/usr/lo
in:/bin:/usr/games:/usr/local/games:/mnt/c/Program Files/NVIDIA GPU Computing Toolkit/CUDA
puting Toolkit/CUDA/v10.2/libnvvp:/mnt/d/ProgramFiles/Anaconda3:/mnt/d/ProgramFiles/Anacon
Anaconda3/Library/usr/bin:/mnt/d/ProgramFiles/Anaconda3/Library/bin:/mnt/d/ProgramFiles/An
```

Imagen 7. Variables activas.

Creando scripts en bash

Bash permite crear scripts con procedimientos a ser realizados sobre archivos y carpetas o variables del entorno. Los scripts de bash no necesitan extensión, pero por convención son creados con la extensión `.sh`. Podemos ejecutar un script de bash de cualquiera de las dos formas siguientes:

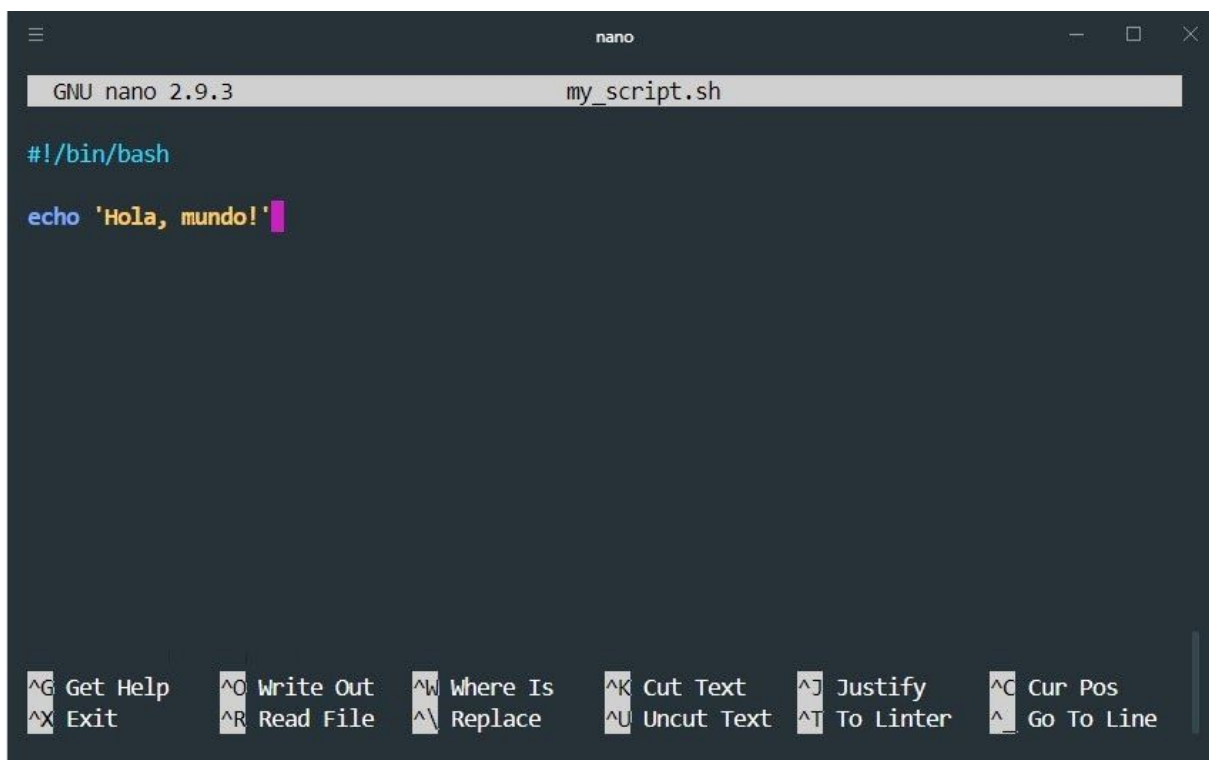
```
./my_script.sh
```

ó

```
bash my_script.sh
```

Además, al momento de hacer el script, debemos colocar en la primera línea del mismo algo llamado "*shebang*", esta línea debe contener lo siguiente: `#!/bin/bash`, esto le dice a nuestro computador donde está el intérprete con el que tiene que ejecutar el script.

Un ejemplo de script de prueba sería el siguiente:



```
GNU nano 2.9.3 my_script.sh

#!/bin/bash
echo 'Hola, mundo!'
```

Imagen 8. Ejemplo `my_script.sh` "Hola, mundo!".

Luego al ejecutarlo observamos que es exactamente equivalente a haber ejecutado esa línea de bash en la terminal:

```
(base)
ignacio@x86_64-conda-linux-gnu /mnt/c/Users/ignac/Desktop
% ./my_script.sh !1747
Hola, mundo!
(base)
ignacio@x86_64-conda-linux-gnu /mnt/c/Users/ignac/Desktop
% echo 'Hola, mundo' !1748
Hola, mundo
```

Imagen 9. Resultado de script y uso de `echo 'Hola, mundo'`.

En bash existen variables pre-definidas que pueden ser usadas libremente dentro de un script, algunas de estas están definidas como sigue:

- `$0` contiene el nombre del script.
- `$<n>` contiene el n-ésimo (con `n>0`) argumento entregado al script al momento de ejecutarlo, por ejemplo, si ejecutamos un script de la siguiente forma:

```
./my_script 'Mi nombre es Ignacio'
```

Podemos invocar a la variable `$1` dentro del script:



Imagen 10. Ejemplos utilizando `echo` dentro de script.

- `$@` contiene todos los argumentos pasados al script.
- `$?` contiene el código de salida del último proceso ejecutado.

Existen otras variables que almacenan elementos más técnicos que no utilizaremos (como el ProcessID), por lo que por ahora solo nos interesarán estas.

- `*` se usa en los scripts de para referenciar a todos los archivos y carpetas en el directorio actual.

Ciclos for

En bash podemos hacer ciclos **for** de forma muy similar a como se hace en python, por ejemplo:



```
GNU nano 2.9.3 my_script.sh

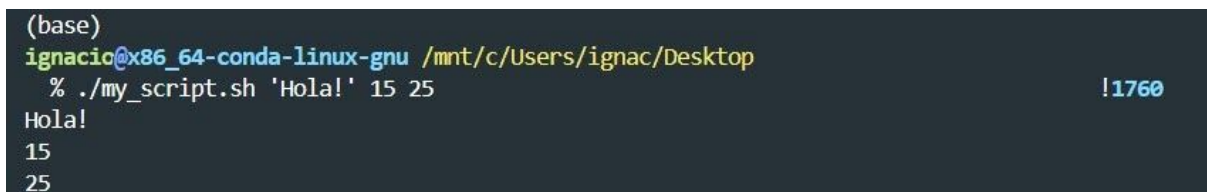
#!/bin/bash

for usr_arg in $@; do
    echo $usr_arg
done
```

[Read 5 lines]

^G Get Help	^O Write Out	^W Where Is	^K Cut Text	^J Justify	^C Cur Pos
^X Exit	^R Read File	^_ Replace	^U Uncut Text	^T To Linter	^_ Go To Line

Imagen 11. Ejemplo de uso **for**.



```
(base)
ignacio@x86_64-conda-linux-gnu /mnt/c/Users/ignac/Desktop
% ./my_script.sh 'Hola!' 15 25 !1760
Hola!
15
25
```

Imagen 12. Resultado.

Sentencias if-then

Es posible hacer condiciones `if-then`, análogas a los `if-else` de la siguiente forma:

A screenshot of a terminal window with the nano text editor open. The editor is editing a file named 'my_script.sh'. The script content is as follows:

```
#!/bin/bash
if [ $1 -eq 2 ]; then
    echo "$1 es igual a 2"
else
    echo "$1 no es igual a 2"
fi
```

The nano editor interface includes a top bar with 'GNU nano 2.9.3' and 'my_script.sh'. A bottom status bar shows '[Read 7 lines]' and various keyboard shortcuts like '^G Get Help', '^O Write Out', '^W Where Is', '^K Cut Text', '^J Justify', '^C Cur Pos', '^U Undo', '^M Mark Text', '^X Exit', '^R Read File', '^H Replace', '^U Uncut Text', '^T To Linter', '^G Go To Line', '^E Redo', and '^C Copy Text'.

Imagen 13. Ejemplo de `if-then`.

En el código mostrado, se debe notar que se termina el uso de una seguidilla de `if-then/else` con las letras `fi`. También, mencionar que `-eq` lo que hace es evaluar la pregunta lógica de si el primer argumento entregado (el de la izquierda: `$1`) y el segundo (el de la derecha: `2`) son equivalentes.

A screenshot of a terminal window showing the execution of the script. The prompt is '(base)'. The user runs 'ignacio@x86_64-conda-linux-gnu /mnt/c/Users/ignac/Desktop' followed by '% ./my_script.sh 2'. The output is '2 es igual a 2'. The prompt changes to '(base)'. The user runs '% ./my_script.sh 5'. The output is '5 no es igual a 2'. The prompt changes back to '(base)'. The terminal also shows line numbers !1772 and !1773.

Imagen 14. Resultado de script con `if-then`.