Uso de bash (parte 1)

Imprimir texto

- **echo** y **printf**

  Ambos reciben strings como argumentos y lo imprimen por la salida estándar de la terminal (**stdout**). Probar ejemplo:

  ```
  > echo “Hola mundo”
  ```

  ```
  > printf “Hola mundo”
  ```

  P: ¿Se obtiene el mismo resultado?

  P: ¿Qué información nos dice el manual sobre las *opciones* de ambos?

Listar contenido de directorio

- **ls**

  Lista en stdout todos los archivos y directorios (ficheros) que se encuentran en el directorio actual. Ejemplos:

  ```
  > ls
  ```

  ```
  > ls /
  ```

  ```
  > ls /home/
  ```

  P: ¿Puedes descifrar el siguiente llamado?

  ```
  > ls -l -t -r -h --all
  ```

  ```
  > ls -ltrha
  ```

Como **argumento**, el guión “-” indica que *todos los caracteres siguientes deben procesarse como **opciones*** del programa.
El guión **doble** “--”, es equivalente pero para procesar una **string**.

Navegación por directorios

home es el **directorio personal** de cada usuario del sistema.

- **cd**

  Permite **cambiar** al **directorio** indicado como argumento, o regresar a **home**. Ejemplos:

  `> cd Desktop` Ingresa al directorio Desktop.

  `> cd /` Ingresa al directorio **raíz del sistema.**

  `> cd` Regresa a home.

  `> cd .` Ingresa al directorio actual.

  `> cd ..` Ingresa al directorio **que contiene** al actual.

  Los argumentos que recibe cd pueden ser rutas (paths) completas de los directorios o rutas parciales (a partir del directorio actual). **Las jerarquías se indican con el slash "/".**

  P: ¿Cuál es el path completo de su actual directorio home?
  P: ¿A qué directorio se refieren: "**./..**" ; "**../..**" ; "**./../../**"?

Creación de archivos y directorios

- **touch**

  Crea un **archivo** vacío. Por ejemplo:

  ```
  > touch apuntes.txt
  ```

  P: ¿Qué tamaño tiene este archivo creado?

- **mkdir**

  Crea un **directorio** vacío. Por ejemplo:

  ```
  > mkdir datos
  ```

  P: ¿Cómo se crean múltiples directorios anidados?
  P: Revisar las propiedades (**pertenencia** y **permisos**) de lo creado.

Eliminación de archivos y directorios

- **rm**

  Elimina un(os) **archivos(s)**, siempre y cuando se tenga **permiso** para ello.

  ```
  > rm apuntes.txt
  ```

  P: Probar las opciones -i y -v de rm.

- **rmdir**

  Elimina un(os) **directorio(s)**, siempre y cuando se tenga **permiso** para ello y esté(n) **vacíos**.

  ```
  > rmdir datos
  ```

  P: ¿Qué sucede cuando se intenta borrar un directorio **no vacío**?
  P: ¿Cómo se puede borrar un directorio **incluyendo** todo su contenido?
  P: ¿Se puede hacer todo lo anterior utilizando **rm**?

Copiar y mover

- **cp**

  Copia un **archivo** o **directorio** de una localización a otra. Es necesario indicar la opción de **recursividad** para los directorios.

  ```
  > cp apuntes.txt apuntes_copia.txt
  ```

  ```
  > cp -r datos datos2
  ```

  P: ¿Qué sucede cuando no se le indica recursividad al copiar directorios?

- **mv**

  Mueve un **archivo** o **directorio** de una localización a otra. Es una instrucción sensible al **uso de “/”**. También sirve para **renombrar**.

  ```
  > mv apuntes_copia.txt datos2/
  ```

  ```
  > mv datos2 curso
  ```

Redirección a archivo

- **>** y **>>**

  Estos **metacaracteres** permiten **capturar la salida estándar (stdout)** de los comandos o programas y **redireccionarlos** a otro destino, usualmente un **archivo**. Por ejemplo:

  ```
  > echo “Hola mundo” > texto.txt
  ```

  ```
  > date >> date.txt
  ```

- Otro metacaracter útil es el “**;**”, que se utiliza para combinar varias instrucciones en una sola línea. Por ejemplo:

  ```
  > mkdir test ; echo “Texto de prueba” > test/texto.txt
  ```

Mostrar archivos en stdout

- **cat**

  Lee un **archivo** y lo imprime por **stdout**. Por ejemplo:

  ```
  > cat date.txt
  ```

  ```
  > cat texto.txt
  ```

  P: Usando cat, determine la diferencia entre los metacaracteres > y >>.

- **head** y **tail**

  Son equivalentes a cat, pero solo muestran las primeras/últimas 10 líneas de un archivo.

  P: ¿Es posible cambiar el número de líneas mostradas?

Mostrar archivos en stdout

- **more** y **less**

  Permiten visualizar archivos (o salidas estándar) extensos con una interfaz interactiva. less es la interfaz usada por man. Ejemplo:

  (descargar archivo:)

```
> wget https://www.gnu.org/licenses/gpl-3.0.txt
```

```
> more gpl-3.0.txt
```

```
> less gpl-3.0.txt
```

  En ambas se puede ingresar a página de ayuda con “h”, o salir con “q”.

Usuarios y grupos en linux

- Para garantizar la capacidad **multi-usuario** de Linux, así como para **administrar permisos y recursos** del sistema, el sistema define **usuarios** y **grupos**, cada uno identificado con una ID numérica.
- Los usuarios pueden estar asociados a: aplicaciones, servicios, o personas que acceden al sistema. Para estos últimos, se les crea un directorio **home**.
- Cada usuario tiene asociado un **grupo principal**, y grupos **secundarios**.
- El detalle de los usuarios y grupos definidos están en:
  `/etc/passwd`
  `/etc/group`

  P: Explorar estos archivos con los comandos aprendidos.
  P: ¿Qué otros archivos relevantes puedes encontrar en /etc?

/etc/
Almacena todas las configuraciones del software instalado en el sistema.

- Cada archivo y directorio del sistema tiene asignado un **usuario y grupo propietario**.
- El usuario (y grupo) administrador se llama **root**.

Actividad

- Utilizando los **comandos de navegación y visualización** anteriores (cd, ls, cat, tail, head, less, more), **explorar el sistema** desde el **directorio raíz /.**
  ¿Puedes identificar el uso de cada uno de los directorios del sistema mediante la exploración de su contenido?

- Utilizar los comandos de **creación y eliminación de archivos** (touch, rm) para verificar que (no) tienen permisos en los directorios de sistema, tratando de crear un archivo con su nombre en ellos.

Permisos

- Cada archivo o directorio en Linux define permisos para **tres categorías**: el **usuario** propietario (u), el **grupo** propietario (g), y **otros** (o). La combinación de estas tres hacen **todos** (a). Los permisos disponibles son:

  - **Lectura (r):** Permite leer el contenido del archivo.

  - **Escritura (w):** Permite modificar el contenido del archivo.

  - **Ejecución (x):** Permite ejecutar el archivo como un programa o script.

- Se utiliza chmod para **asignar**, **agregar** (+) o **quitar** (-) permisos a un archivo. Como argumento recibe un string del tipo: “categorías”+”modificador”+”permisos”. Por ejemplo:

  `> chmod a+w texto.txt` Da permisos de lectura a todos

  `> chmod o-rw date.txt` Quita permisos de lectura y modificación a otros

  También se puede usar un código numérico de tres dígitos para asignar permisos. Utilizando r=4, w=2 y x=1, la suma define los permisos deseados para cada categoría. Por ejemplo:

  `> chmod 644 date.txt` Asigna rw para usuario y r para grupo y otros.

  P: Cree un archivo de texto con contenido, modifique los permisos de usuario propietario y verifique que estos son efectivos. ¿Si le quita los permisos w, puedes volver a otorgarlos?

Variables locales

- Se pueden definir **variables locales** utilizando el signo “=”, sin espacios. Por ejemplo:

```
> contador=1
> MENSAJE=”Hola mundo”
```

  El nombre de las variables es sensible a mayúsculas/minúsculas, y no puede comenzar con un número.

- Para hacer **referencia a una variable**, se usa el signo $ o ${ }. Por ejemplo:

```
> echo $MENSAJE
> echo ${contador}
```

  P: Probar lo siguiente:

```
> uno=1; dos=2; tres=uno+dos
> echo $uno; echo $dos; echo $tres
```

  ¿Qué puede concluir sobre el uso de variables en bash?

- Las variables locales sólo pueden ser referenciadas desde el intérprete que las creó, sin transmitirse hacia **derivados**.

Variables de entorno (globales)

- Para que una variable sea compartida hacia los programas derivados del intérprete, es necesario convertirla en una **variable de entorno**. Para eso se usa **export**. Por ejemplo:

```
> SALUDO=”Que tengas un lindo dia”
> export $SALUDO
```

- Se puede acceder a la lista de variables definidas en un entorno bash mediante el comando **env.**

  P: Verificar la lista de variables de entorno con env.
  ¿Que almacenan $HOME, $PWD, $USER, $SHELL, $PS1?

- Para eliminar una variable (global o local), se puede usar **unset**.

```
> unset $SALUDO
```

Usualmente se utilizan nombres con mayúsculas para las variables de entorno.

Aritmética de variables

- La aritmética de variables tiene **restricciones** en Bash, pero se pueden hacer cálculos simples (principalmente enteros). Para ello, es necesario usar $(( )).

```
> uno=1; dos=2
> echo $(($uno + $dos))
> echo $(($uno / $dos))
```

P: Probar definir distintas variables numéricas (enteras y reales) y verificar qué operaciones aritméticas son aceptadas por Bash.
¿Qué puedes concluir respecto a los tipos de datos de las variables de bash?

Editores de texto

Aunque todos los Linux traen editores de texto en modo gráfico (GUI), es importante conocer y saber usar algunos que funcionan en la CLI. Ejemplos:

- **nano**
  Ideal para iniciar. Permite editar texto de forma directa e incluye su ayuda en pantalla.
- **emacs**
  Editor de texto avanzado del proyecto GNU. Requiere uso de combinaciones de teclas con Ctrl para realizar operaciones avanzadas.
- **vim** (sucesor de **vi**)
  Editor de texto avanzado que utiliza dos ambientes (intercambiables con las teclas Esc e i).

Bash scripting

Un script de bash es un **archivo de texto** que contiene instrucciones a ser ejecutadas por un intérprete **bash**.

- La **primera línea** del archivo debe ser:
  `#!/bin/bash`
- El archivo debe tener **permisos de ejecución.**
- Usualmente se guardan con **extensión .sh** (aunque no es necesario).
- El metacaracter **“#”** se utiliza para ingresar comentarios.

P: Escribir un script que cree directorios anidados /foo/bar/, y cree archivos de texto con descripciones en cada uno de ellos. Debe asignar permisos de solo lectura a los archivos dentro de bar, y luego listar todo lo creado. Documentar (con #) cada etapa del proceso.

Bash scripting - inputs

- Argumentos de ejecución

  En un **script** de bash, las variables **$0**, **$1**, **$2**, etc, almacenan los argumentos de entrada que se utilizaron para ejecutar el script (partiendo por el nombre del mismo).
  El string completo utilizado se almacena en **$@** y el número de entradas en **$#**.

  P: Crear un script que muestre por stdout: la línea de su ejecución, el número de argumentos, y los primeros 3 utilizados (en orden).

- stdin

  Se le puede solicitar al usuario que ingrese información por el teclado (standard input, stdin) con el comando read, el cual guarda lo ingresado en una variable.

  ```
  > read VARIABLE
  ```

  También se puede incluir un mensaje en el llamado de read con la opción -p. Por ejemplo:

  ```
  > read -p "Ingresa tu nombre: "   NOMBRE
  ```

  P: Crear un script interactivo que solicite datos personales y los guarde en un archivo de texto dentro de un directorio “personas”.

Posibles input/output (I/O) de un software.