Sistemas Operativos

Práctica 0: Preliminares

Parte 1 – Terminal de Linux | Bash Scripting

El comando `man` muestra el manual de usuario para cualquier comando que se pase como argumento. Por ejemplo, para ver el manual del comando `ls`:
`$ man ls`
Escrolear para ver el contenido que se precise, y salir con `q`.
Agregar `-k` para buscar en las páginas del manual del sistema (también conocidas como "man pages") mediante una palabra clave específica. Ejemplo:
`$ man -k network`

Ejercicio 1 (*Comandos de información*)

Utilizar `man` para responder las siguientes preguntas:

a) ¿Qué hace el comando `whoami`?
b) ¿Qué hace el comando `uname`? ¿Para qué sirve la opción `-a`?
c) ¿Qué hace el comando `id`?
d) ¿Qué hace el comando `ps`? ¿Para qué sirve la opción `-e`?
e) ¿Qué hace el comando `top`? ¿Qué sucede al ejecutar `top -n 10`?

Ejercicio 2 (*Inspeccionar archivos y directorios*)

a) Al trabajar en una terminal de Linux, siempre estás trabajando desde un directorio. Por defecto, comenzás en tu directorio home.

   Para obtener la ruta absoluta de tu directorio de trabajo actual:

   ```
   $ pwd
   ```

b) Para listar los archivos y directorios en el directorio actual:

   ```
   $ ls
   ```

   Si está vacío, no retorna nada.

   Para ver los archivos y directorios dentro de cualquier directorio, se puede pasar el path como argumento:

   ```
   $ ls [PATH TO DIRECTORY]
   ```

   i. ¿Qué se obtiene al ejecutar `ls /`?
   ii. ¿Cómo se puede observar el contenido del directorio `/bin`?

   Para imprimir una lista más larga de archivos con información adicional:

   ```
   $ ls -l
   ```

   iii. Consultá el manual para `ls` y explicá para qué sirven estas opciones:

Opción	Descripción
-a	
-d	
-h	
-l	
-S	
-t	
-r	

iv. ¿Qué resultado se obtiene de ejecutar `ls -la /etc`?

Ejercicio 3 (*Creando archivos y directorios*)

a) El comando `mkdir` se usa para crear un nuevo directorio.

i. Creá un directorio llamado `scripts` en tu directorio actual:

```
$ mkdir scripts
```

ii. Usá el comando `ls` para verificar que el directorio fue creado.

b) Para cambiar tu directorio de trabajo actual al directorio `scripts`:

```
$ cd scripts
```

Usar el comando `pwd` para verificar que estás en el directorio que querías.

Si ingresás el comando `cd` sin argumentos, volvés al directorio home.

Para navegar al directorio padre de tu actual directorio:

```
$ cd ..
```

Recordar los siguientes símbolos para navegar a paths especiales:

Símbolo	Path al que refiere
∼	Directorio Home
/	Directorio Root
.	Directorio actual
..	Directorio padre

c) Volvé a tu directorio home y creá un archivo vacío llamado `miarchivo.txt`:

```
$ touch miarchivo.txt
```

Usá el comando `ls` para verificar que se creó el archivo. En caso de que el archivo ya existiese de antes, el comando `touch` actualiza el timestamp de último acceso o la fecha de última modificación. Para ver esta información:

```
$ stat miarchivo.txt
```

Ejercicio 4 (*Manejo de archivos y directorios*)

a) El comando `find` se usa para buscar archivos en un directorio. Se puede buscar por nombre, tipo, dueño, tamaño o timestamp.

Este comando busca en el árbol de directorios a partir del directorio dado. Por ejemplo, para encontrar todos los archivos .txt en el directorio `/etc` y todos sus subdirectorios:

```
$ find /etc -name "*.txt"
```

b) El comando `rm` se usa para borrar archivos, y con la opción `-i` pide confirmación antes de borrar nada.

i. Borrá el archivo `miarchivo.txt`:

```
$ rm -i miarchivo.txt
```

ii. Usá `ls` para verificar el borrado.

c) Para borrar directorios con todo su contenido, se usa el comando `rm` con la opción `-r`.

i. Creá un directorio `carpeta1`, movete dentro, y creá varios archivos dentro:

```
$ touch a.txt b.txt c.txt d.txt
```

ii. Volvé al directorio padre de `carpeta1` y eliminala:

```
$ rm -r carpeta1
```

iii. ¿Cuál es la diferencia entre `rm -r` y `rmdir`?

d) Se puede usar el comando `mv` para mover archivos de un directorio a otro y/o renombrarlo.

i. Creá un archivo llamado `usuarios.txt`. Cuidado que si movés un archivo a un directorio donde hay otro archivo con el mismo nombre, lo va a sobreescribir.

ii. Podés cambiar el nombre con el mismo comando:

```
$ mv usuarios.txt info-user.txt
```

En este caso estás en el mismo directorio, así que sólo cambiaste el nombre del archivo. Podés verlo con `ls`.

iii. Ahora mové el archivo al directorio `/tmp`:

```
$ mv info-user.txt /tmp
```

iv. Revisá con `ls` que se hizo bien:

```
$ ls
$ ls -l /tmp
```

e) Podés usar el comando `cp` para copiar el archivo `info-user.txt`, que ahora está en el directorio `/tmp`, al directorio de trabajo actual:

```
$ cp /tmp/info-user.txt info-user.txt
```

Revisá con `ls` que se hizo bien.

Ejercicio 5 (*Ver y modificar permisos de acceso*)

Cada archivo y directorio en Linux tiene permisos para tres categorías: 'user', 'group', y 'all users' (o 'other').

Estos son los permisos que se setean para cada archivo y directorio:

Permiso	**Símbolo**
read	r
write	w
execute	x

a) Movete al directorio `/home/taller0` y mirá los permisos de `archivo_misterioso.txt`.

Un ejemplo de la salida de este comando podría ser:

```
$ -rw-r--r-- 1 ssoo ssoo 8121 Ago 15 16:45 archivo_misterioso.txt
```

El primer - indica que se trata de un archivo, si fuese un directorio habría una `d`. Las primeras tres entradas corresponden al usuario, los siguientes tres corresponden al grupo, y los últimos tres son de 'other'.

i. ¿Quiénes tienen permiso de lectura? ¿Y de escritura? ¿Y de ejecución?

b) El comando `chmod` permite cambiar los permisos para un archivo. Una combinación de estos caracteres especifica qué permisos cambiar:

Opción	**Descripción**
r,w,x	**Permisos:** read, write, execute
u,g,o	**Categoría de usuario:** user, group, other
+,-	**Operación:** dar o quitar

Por ejemplo, para quitar permisos de lectura para todos los usuarios:

```
$ chmod -r archivo_misterioso.txt
```

Verificá que cambiaron los permisos con `ls -l` .

Para volver a darle los permisos de lectura:

```
$ chmod +r archivo_misterioso.txt
```

Si quisiera quitar los permisos de lectura sólo para 'other':

```
$ chmod o-r archivo_misterioso.txt
```

Ejercicio 6 (*Archivos de texto: ver contenido*)

En el directorio `/home/taller0` vas a encontrar un archivo `esto_es_un_script.sh`. El sufijo `.sh` es una convención para identificar que el archivo es un `bash script`.

a) El comando `cat` muestra el contenido de un archivo:

```
$ cat esto_es_un_script.sh
```

Si el archivo es muy largo para verse en la terminal, sólo muestra la última parte.

b) Otra alternativa es usar el comando `more`:

```
$ more esto_es_un_script.sh
```

Esto muesta la primera parte del archivo primero. Con la barra espaciadora podés ir pasando de a *página* hasta el final del archivo. Al llegar al final del archivo se vuelve a la línea de comandos. También se puede volver con `q`.

c) Para poder avanzar y subir en el archivo se puede usar el comando `less`:

```
$ less esto_es_un_script.sh
```

Acá también se empieza a mostrar la primer página y luego permite subir o bajar con las flechitas o `AvPag` y `RePag`. En este caso, al llegar al final del archivo no se vuelve a la línea de comandos, para esto se debe usar `q`.

d) Por defecto, el comando `head` imprime las 10 primeras líneas de un archivo:

```
$ head esto_es_un_script.sh
```

También podés especificar el número de líneas a imprimir:

```
$ head -3 esto_es_un_script.sh
```

e) Por defecto, el comando `tail` imprime las 10 últimas líneas de un archivo:

```
$ tail esto_es_un_script.sh
```

También podés especificar el número de líneas a imprimir:

```
$ tail -2 esto_es_un_script.sh
```

Ejercicio 7 (*Archivos de texto: extraer líneas y campos*)

a) El comando `grep` permite especificar un patrón y buscar las líneas del archivo que coincidan con el patrón.

i. Imprimí todas las líneas de `archivo_misterioso.txt` con la palabra `escalera`:

```
$ grep escalera archivo_misterioso.txt
```

ii. Consultá el manual para `grep` y explicá para qué sirven estas opciones:

Opción	Descripción
-n	
-c	
-i	
-v	
-w	

iii. ¿Qué resultado se obtiene de ejecutar `grep -v login /etc/passwd`?

b) El comando `cut` permite ver sólo los campos especificados de cada línea de texto en un archivo. Por ejemplo, para ver sólo los dos primeros caracteres de cada línea:

```
$ cut -c -2 animales.txt
```

Y para ver el contenido de la línea a partir del segundo caracter:

```
$ cut -c 2- animales.txt
```

También se puede usar `cut` para extraer un campo de un archivo con delimitadores. En el archivo `nombres_y_numeros.csv` hay una lista de nombres de personas con sus números de teléfono. Para extraer sólo los números de cada persona:

```
$ cut -d "," -f2 nombres_y_numeros.csv
```

i. ¿Qué representa cada argumento en esta última línea?

Ejercicio 8 (*Crear y ejecutar un bash script básico*)

a) Creá un archivo llamado `saludar.sh`. Y colocá el siguiente contenido dentro del archivo:

```
# script que pide un nombre y saluda
echo -n "Escribí tu nombre"
read nombre
echo "Hola $nombre"
echo -n "Felicidades por tu primer script"
```

b) Chequeá los permisos del archivo. Si el archivo existe y tiene permisos de lectura, ejecutalo:

```
$ bash saludar.sh
```

c) Para que el script sea un archivo ejecutable y puedas usar su nombre como un comando de terminal, hay que añadir una línea que especifica el path al intérprete del script, que en este caso es *Bash shell*.

i. Primero hay que saber el path al intérprete. Para eso se puede usar el comando `which`:

```
$ which bash
```

Posiblemente devuelva el path `/bin/bash`.

ii. Abrí el archivo `saludar.sh` y colocá la siguiente línea al inicio del script:

```
#! /bin/bash
```

iii. Chequeá los permisos del archivo, y añadí permisos de ejecución si no los tuviera.

iv. Ejecutá el archivo:

```
$ ./saludar.sh
```

El . al principio refiere al directorio actual. Esta línea le está diciendo a Linux que ejecute el script `saludar.sh` y que lo puede encontrar en el directorio actual.

Ejercicio 9 (*Más sobre shell scripting*)

En Bash hay caracteres que se interpretan de manera especial:

Caracter	Significado
#	Precede a un comentario
;	Separador de comandos
*	'Comodín' para nombres de archivos
?	'Comodín' para 1 caracter en nombres de archivos

Por ejemplo:

```
# Esto es un comentario

# Acá pongo múltiples comandos en una única línea
echo "Hola "; echo "Mundo =)"

# Acá veo todos los archivos .txt en el directorio actual
ls *.txt

# Acá veo todos los archivos cuyo nombre empiece con archivo,
# siga con un caracter cualquiera, y luego termine con .txt
ls archivo?.txt
```

a) La sintaxis para sentencias condicionales es así:

```
if [ condición ] # con los corchetes y espacios
then
    sentencias cuando condición es true
else
    sentencias cuando condición es false
fi
```

i. Creá un nuevo script de bash y hacelo ejecutable.

ii. En tu script, escribí el código necesario para que se imprima una pregunta que el usuario deba responder por "si o no". Se debe almacenar la respuesta del usuario en una variable.

iii. Usá la sentencia condicional para definir un mensaje diferente para cada respuesta del usuario.

b) Hay operadores lógicos para comparar por igual (==), distinto (!=), menor o igual (<=), etc.

También hay operadores aritméticos para sumar (+), restar (-), multiplicar (*) y dividir (/) enteros. Se debe usar la notación $(()). Por ejemplo:

```
# esta es una forma de sumar
echo $((3+2))

# y esta es otra forma usando variables
a=3
b=2
c=$(($a+$b))
echo $c
```

i. Creá un script de bash que pida al usuario dos números enteros, los almacene, y luego imprima la suma y el producto de ambos números.

ii. Añadí al script la lógica para determinar si la suma es mayor o igual, menor o igual, o igual al producto. Se debe imprimir un mensaje apropiado para cada caso.

c) Un **array** en Bash es una lista de elementos que se escriben entre paréntesis y se delimitan entre sí con espacios. Los arrays se trabajan de esta forma:

```
mi_array=(1 2 "tres" "cuatro" 5) # creo array con elementos
declare -a array_vacio # creo array vacío

mi_array+=("six") # agrego elemento
mi_array+=(7) # agrego elemento

echo ${mi_array[0]} # imprimo el primer elemento
echo ${mi_array[2]} # imprimo el tercer elemento
echo ${mi_array[@]} # imprimo todos los elementos
```

Se pueden recorrer los arrays con `for` loops:

```
# una forma
for item in ${mi_array[@]}; do
echo $item
done

# otra forma
for i in ${!mi_array[@]}; do
echo ${mi_array[$i]}
done
```

Los `for` loops también se pueden usar para iteraciones típicas:

```
N=6
for (( i=0; i<=$N; i++ )) ; do
    echo $i
done
```

i. En el directorio home hay un archivo llamado `tabla.csv`. Mirá su contenido en la terminal para ver qué pinta tiene.

ii. Creá un script de bash que parsee las columnas de la tabla en 3 arrays.

iii. Creá un nuevo array con la diferencia de la segunda columna y la tercera columna, y asegurate de que tenga un nombre de columna (como las demás).

iv. Creá un nuevo archivo .csv combinando la tabla original con la nueva columna.
Hint1: Investigá qué pasa en la terminal cuando hacés esto:

```
$ echo "escribo una cosa" > ejemplo.txt
$ echo "escribo otra cosa más" > ejemplo.txt
$ echo "y otra vez" >> ejemplo.txt
```

Hint2: Buscá el manual de `paste`, fijate cómo te puede servir.

Parte 2 – Programacion en C | Makefile

Ejercicio 10 (*Programa con argumentos, manejo de strings*)

Se quiere tener un programa que reciba dos argumentos: un número entero N y una cadena de texto. Al ejecutarse el programa, si el tamaño de una palabra del texto es mayor a N, entonces esa palabra debe ser invertida y pasada a mayúsculas. Por ejemplo:

```
$ ./wordInverter 4 "Hola, me gustaria introducirme al mundo!
Hola, me AIRATSUG EMRICUDORTNI al ODNUM
```

Ejercicio 11 (*Standard input/output*)

CUIT es un número que se usa para la identificación de aquellos que están inscriptos en el sistema tributario argentino. Tiene un total de 11 dígitos, donde

- Los dos primeros números indican el tipo global.
- En el caso de personas humanas, le siguen los ocho números del Documento Nacional de Identidad (DNI).
- En empresas y negocios, se asignan los números de sociedad que el ente regulador asigna en forma aleatoria.
- La clave se completa con un dígito verificador.

Este dígito verificador sigue este algoritmo: A cada uno de los 10 dígitos que conocemos, se multiplica por: 5, 4, 3, 2, 7, 6, 5, 4, 3, 2 respectivamente. Se suman los valores obtenidos, el resultado se divide por 11, y del resultado se toma la parte entera. Se le restan 11 y el resultado es el número verificador.

Se pide un programa en C que tome por entrada estándar un número de CUIT y comprobar si es válido. En caso que sea válido, mostrar un mensaje diciendo que lo es. En caso que no lo sea, corregir el dígito verificador e imprimir por pantalla en el formato "xx-xxxxxxxx-x".

Por ejemplo:

```
$ ./verificadorCuit 33693450239
"El CUIT ingresado es valido"

$ ./verificadorCuit 30500011838
"El CUIT ingresado es válido"

$ ./verificadorCuit 30576124272
"El CUIT ingresado no es válido. Su información corregida es 30-57612427-5
```

Ejercicio 12 (*Manejo de memoria, manejo de structs*)

Se pide un programa que tome comandos por entrada estándar. Se desea programar dos comandos:

a) `agregarInformacionUsuario`: Cuando se selecciona este comando, se desea ingresar datos de una persona, como nombre, edad y CUIT. Se desea guardar la información en el sistema, solamente de este usuario. Mostrar un mensaje de éxito cuando se termine de cargar. También hay que verificar que tanto la edad como el CUIT sean válidos. La edad se debe encontrar en el rango entre 18 y 99. El CUIT se debe validar usando el ejercicio anterior. Imprimir un mensaje indicando si hubo alguna información inválida.
b) `verInformacionUsuario`: Se desea imprimir por pantalla la información del usuario ingresada. En caso que no se haya ingresado nada previamente, imprimir por pantalla que no hay información para mostrar.

Por ejemplo:

```
$ ./plataformaCargaUsuarios

> verInformacionUsuario
¡Ningún usuario fue cargado a la plataforma todavía!

> agregarInformacionUsuario
Ingrese nombre de persona:
> Rocio
Ingrese edad:
> 25
Ingrese cuit:
> 20004285876
¡Su información fue cargada con éxito!

> verInformacionUsuario
nombre: Rocio, edad: 25, cuit: 20004285876

> agregarInformacionUsuario
Ingrese nombre de persona:
> Rocio
Ingrese edad:
> 150
Ingrese cuit:
> 20004285876
La información cargada es invalida, por favor, intente de nuevo
```

Sugerencia: Usar `scanf`, `malloc` y `strcmp`

Ejercicio 13 (*Manejo de memoria*)

Al programa del ejercicio anterior, se lo desea extender para poder guardar tantos usuarios como deseen. Es decir, en vez de guardar solamente el último usuario cargado, se debe permitir guardar todos los usuarios que se ingresen. Luego, cuando se quiera usar el comando de imprimir, debe mostrar todos los usuarios guardados.

Por ejemplo:

```
$ ./plataformaCargaUsuarios
```

```
> agregarInformacionUsuario
Ingrese nombre de persona:
> Rocio
Ingrese edad:
> 25
Ingrese cuit:
> 20004285876
¡Su información fue cargada con éxito!

> agregarInformacionUsuario
Ingrese nombre de persona:
> Juan
Ingrese edad:
> 26
Ingrese cuit:
> 20985165352
¡Su información fue cargada con éxito!

> verInformacionUsuario
nombre: Rocio, edad: 25, cuit: 20004285876
nombre: Juan,  edad: 26, cuit: 20985165352
```

Sugerencia: Usar `realloc`

Ejercicio 14 (*Manejo de structs*)

Se desea agregar un nuevo comando `buscarInformacionUsuario`, que dado un CUIT, nos devuelva que si existe un usuario con este cuit, nos imprima su información. En caso contrario, mostrar que no existe usuario en la plataforma.

Ejercicio 15 (*Estructura de archivos y Makefile*)

Teniendo resueltos los ejercicios anteriores (sobre CUIT), se quiere organizar la solución en una estructura de archivos de la siguiente forma:

- Se tiene que tener un archivo `main`, que es el encargado de tomar el input de usuario para los comandos.
- Luego, cada comando tiene que estar en un archivo aparte específico. Por ejemplo: el comando `buscarInformacionUsuario` tiene su archivo `buscarInformacionUsuario.c` y debe poder ser invocado desde el archivo `main`.
- Se quiere poder compilar estos archivos usando un `makefile` y el comando `make`. También se debe poder hacer `make clean`.