Universidad de Costa Rica

Facultad de Ingeniería
Escuela de Ingeniería Eléctrica
IE-0117 – Programación Bajo Plataformas Abiertas
II ciclo 2025

Informe :

Laboratorio 1

Danny Garro Arias B93230

Profesora:
Carolina Trejos Quirós.

28 de agosto de 2025

Índice

1. Introducción

El presente informe busca mostrar los resultados obtenidos, así como el procedimiento necesario para resolver los problemas de programación presentados. El primer problema muestra la necesidad de realizar un script en Bash que imprima información relevante sobre el sistema utilizado por el usuario, mientras que el segundo problema abarca conceptos básicos de procesamiento de texto.

En esta última parte, además se incluye el comando *getopts*, que permite habilitar opciones y argumentos para un script. Esto es de gran utilidad, ya que le permiten al usuario ejecutar diversas funcionalidades para el mismo programa.

2. Scripting y el sistema

El objetivo de este ejercicio es obtener información relevante del sistema. Para ello, se debe escribir un script en Bash que imprima la siguiente información:

- Número de CPUs.
- Total de memoria RAM disponible.
- Espacio libre en disco.
- Versión de kernel de Linux.
- Tiempo de actividad del sistema (uptime).
- Información sobre el sistema operativo (distribución y versión).
- Nombre del host del sistema.

La solución comenzó con la creación de un script en bash. Para esto se abrió una terminal, se posicionó el directorio de trabajo en la carpeta deseada `/home/dgarroa/plataformas_abiertas/Laboratorios/1_laboratorio` y se creó el script con el siguiente comando:

```
$ emacs info_sistema.sh &
```

Esto abrió el editor de texto *emacs* y al mismo tiempo creó el archivo de Bash `info_sistema.sh`. Se otorgaron permisos de ejecución al archivo con el siguiente comando:

```
$ chmod +x info_sistema.sh
```

Los comandos utilizados para mostrar la información requerida fueron los siguientes, además se explica la forma en que producen la salida deseada.

El comando **nproc** devuelve la cantidad de unidades del procesador (CPUs).

```
echo "`nproc`"
```

Para mostrar información de la memoria RAM, se utilizó el comando **vmstat**, el cual brinda información de procesos, memoria, discos, cpu, entre otros. Se emplea la opción *-s* para brindar la información de forma tabular y se emplea *–unit M* para mostrar los resultados en unidades de Mb; finalmente, por medio de un pipe, se envía al comando **grep**, cuya función es mostrar las salidas que coincidan con un argumento, en este caso *"memory"*, para mostrar únicamente los resultados correspondientes a la memoria RAM.

```
echo " `vmstat -s --unit M | grep memory`"
```

Para mostrar información acerca del espacio libre en disco se empleó el comando **df**, el cual se encarga de brindar un reporte del uso del disco y sus particiones. Se empleó la opción *-h* para mostrar los resultados en unidades fáciles de leer, y esto se envió por medio de un pipe al comando **awk**. El comando **awk** es una herramienta para procesar texto, en este caso se empleó *BEGIN {OFS = " "}* para establecer el espacio como delimitador de la salida, cuyos valores son las columnas 1, 4 y 5 como se indican en *{print , $1, $4, $5 }*. Finalmente, la salida de **awk** se envía, con un pipe, al comando **column**, con la opción *-s ' '* para definir el delimitador que se desea para la separación en columnas, y finalmente la opción *-t* para brindar el formato en forma tabular, tal como se muestra en la salida. Con respecto al resultado mostrado, la primera columna corresponde a la partición, la segunda columna (Avail) indica el espacio disponible, y la tercera columna indica el porcentaje de uso de la unidad de memoria correspondiente.

```
df -h | awk 'BEGIN {OFS = " "} {print "    ", $1, $4, $5}' | column -s ' ' -t
```

Para mostrar la versión de kernel de Linux se emplea **uname**, cuya función es imprimir información del sistema. Como únicamente se requiere la información del kernel, se empleó la opción *-r*, correspondiente al *kernel release*.

```
echo "`uname -r` "
```

El siguiente comando empleado fue **uptime**, cuya función es brindar información del tiempo de actividad. Se empleó la opción *-p*, con el fin de mostrar la salida de forma más legible.

```
echo "`uptime -p`"
```

Lo siguiente fue mostrar la información de la distribución y versión del sistema operativo, para lo cual se utilizó el comando **lsb_release**, cuya opción *-i* y *-r* brindan el ID del distribuidor y el número de *release*, respectivamente.

```
echo "    - `lsb_release -i` "
echo "    - `lsb_release -r`"
```

Finalmente, para mostrar el hostname, cuyo significado es el nombre del dispositivo, se utiliza el comando **hostname**.

```
echo "`hostname`"
```

Cuya ejecución final del programa brindó el siguiente resultado:

```
$ ./info_sistema.sh

Informacion del sistema:

 Numero de CPUs: 8
```

```
Memoria RAM:
        15825 M total memory
        1946 M used memory
        3416 M active memory
        7124 M inactive memory
        4338 M free memory
          17 M buffer memory

Espacio libre en disco:
       Filesystem  Avail  Use%
       tmpfs       1,6G   1%
       /dev/sdb8   205G   21%
       tmpfs       7,8G   1%
       tmpfs       5,0M   1%
       /dev/sdb8   205G   21%
       /dev/sdb7   679M   23%
       /dev/sdb1   33M    67%
       tmpfs       1,6G   1%

Version del kernel de Linux: 6.2.0-37-generic

Tiempo de actividad del sistema: up 20 hours, 51 minutes

Informacion sobre el sistema operativo:
   - Distributor ID:        Ubuntu
   - Release:        22.04

Host: dgarroa-Nitro-AN515-52
```

3. Scripting y procesamiento de texto

3.1. Lectura de archivo

Inicialmente, se requiere crear un script que reciba la ruta de un archivo como argumento por línea de comandos, que muestre el número total de líneas e imprima las primeras 3 líneas del archivo.

Para esto, se comenzó declarando la variable *nombre_ arch*, cuyo valor es la primera entrada argumental *$1* proporcionada al ejecutar el programa.

```
nombre_arch="$1"
```

Para imprimir la cantidad de líneas, se empleó el comando **wc**, cuya función es contar lineas, palabras y bytes. Empleando la opción *-l*, es posible contar la cantidad de líneas. Para pasar la información de entrada a **wc** se utilizó un *pipe*, cuya información corresponde a la salida del comando **cat**, que imprime envía a la salida el contenido completo del archivo.

```
echo "Lineas: `cat $nombre_arch | wc -l`"
```

Luego, para mostrar las primeras tres lineas del archivo se utilizó el comando **head**, cuya función principal es precisamente la descrita, pemitiendo controlar la cantidad de líneas por medio de la opción *-n* e ingresando seguidamente la cantidad deseada.

```
echo "` head -n 3  $nombre_arch`"
```

Además de esto, se debe asegurar que lo anterior cumpla con los siguientes requerimientos:

- Si el archivo no existe o no tiene permisos de lectura, muestre un mensaje de error y finalice la ejecución del script con un código de salida diferente de 0.
- Si se utiliza la bandera -h, muestre una guía de como utilizar su script.

Para satisfacer el primer punto, se colocó el código mostrado anteriormente dentro de un *If*, que evalúa si el archivo tiene permiso de lectura y si el archivo existe. En caso de que alguna de las condiciones no se cumpla, se ejecuta el *else* y se brinda un error de *1*. Lo anterior con el fin de mostrar que se produjo un error por parte del usuario, al indicar una entrada invalida o no tener permisos.

```
if [[ -e $nombre_arch && -r $nombre_arch ]]; then
    ...
else
    echo "ERROR: Archivo no existe o no tiene permiso de lectura"
    exit 1
fi
```

Para satisfacer el segundo punto, se agregó un bloque de código que verifica si la primera entrada argumental es *-h*; que en caso de ser correcto, se imprime un mensaje de ayuda. EL código empleado para este bloque tiene la siguiente forma:

```
if [[ $1 == "-h" ]]; then
    ...
    exit 0
fi
```

El resultado de la salida, basado en los comandos mencionados, es el siguiente:

```
$ ./ejercicio2.sh access-2025-05.log

Lineas: 8000
Primeras 3 lineas:
203.0.113.174 - - [01/May/2025:00:00:57 +0530] "GET /search?q=coffee HTTP/1.1" 200 1522
127.0.0.1 - - [01/May/2025:00:01:37 -0700] "OPTIONS /home HTTP/1.1" 200 8232
172.20.252.24 - - [01/May/2025:00:04:18 +0100] "GET /search?q=guitar HTTP/1.1" 400 1119
```

Como se muestra en el resultado anterior, se imprimió satisfactoriamente el mensaje de que el archivo cuenta con 8000 líneas, y se muestra el contenido de las primeras tres de estas. AL ingresar *-h* se obtiene el siguiente resultado:

```
$ ./ejercicio2.sh -h
Uso del script: ejercicio.sh [archivo]

EL script muestra lo siguiente sobre el archivo indicado:
     - Muestra la cantidad de lineas
     - Muestra el contenido de las primeras tres lineas

Si no se indica un archivo, el archivo no existe o no tiene permisos de lectura, se produce una salida
     - ERROR: Archivo no existe o no tiene permiso de lectura

Ejemplo de uso del codigo:
  $./ejercicio2.sh access-2025-06.log
```

3.2. Filtrar e imprimir por ruta

En esta parte se requiere extender el script para que extraiga el PATH de cada línea y muestre los primeros 5 paths más repetidos.

Para esto, se implementó la siguiente serie de comandos:

```
# Extrae los paths (columna 7), ordena, cuenta valores unicos, ordena numéricamente y de forma decender
echo ""
echo "Rutas mas repetidas:"
awk '{print $7}' $nombre_arch | sort | uniq -c | sort -n -r | head -n 5
```

Para lo cual se obtuvo la siguiente salida, para el archivo *access-2025-05.log*:

```
  1619 /home
   625 /contact
   475 /about
   473 /pricing
   216 /api/logout
```

El funcionamiento de la serie de comandos es la siguiente. Se empleó una serie de *pipes*, por lo que cada salida de un comando a la izquierda de cada *pipe* corresponde a la entrada del comando a la derecha del *pipe*. Se comienza con el comando **awk** que extrae la columna 7 (correspondiente al path de cada línea) del archivo. Esta información entra en el comando **sort**, que realiza un ordenamiento inicial de la información de forma alfabética. Es importante realizar este primer ordenamiento, ya que se requiere que las líneas repetidas se encuentren adyacentes. Seguido de esto, el comando **uniq** se encarga de extraer únicamente las líneas no repetidas; a esto se agrega la opción *-c*, que realiza un conteo de la cantidad de líneas correspondientes a cada línea repetida. Para que lo anterior funcione, las líneas repetidas deben estar adyacentes. Luego, se aplica nuevamente el comando **sort**, para ordenar la información, pero esta vez de forma numérica (con la opción *-n*) por cantidad de repeticiones y en orden descendente (con la opción *-r*).

3.3. Opciones con getops

En esta última parte se requiere emplear el comando **getopts**, para agregar las siguientes opciones adicionales al script.

- -f campo: Campo a filtrar. Puede ser PATH, IP o STATUS. Si no lo especifica, utilice PATH por defecto.
- -t N: cantidad de resultados a mostrar. Si no se especifica, utilice 5 por defecto.

Para lograr esto, se empleó un código con la siguiente forma:

```
while getopts "hf:t:" opciones; do
    case ${opciones} in
        h)
            ayuda
            ;;
        f)
            variable_extraer=$OPTARG
            var_spec=1
            ;;
        t)
            cantidad_resultados=$OPTARG
            cant_spec=1
            ;;
        \?)
            echo "Error opción invalida. Utilice -h para leer la descripción del programa."
            exit 1
            ;;
        :)
            echo "Error, debe ingresar un argumento para la opción."
            exit 1
            ;;
    esac
done
```

La funcionalidad de este bloque es la siguiente. Se comienza con un ciclo **while**, que va a iterar sobre lo realizado por **getopts**. El comando **getopts** toma el string, en este caso *"hf:t:"*, lo cual indica que se esperan las opciones *-h*, *-f* y *-t*, donde para *-f* y *-t* se esperan argumentos (en este caso el campo a extraer y la cantidad de resultados, respectivamente). Seguido de esto se encuentra la variable *opciones*, que corresponde a la variable donde se guardará el argumento ingresado, para cada vez que se detecte una opción entre las declaradas.

Luego de esto se emplea un case, que irá comparando el argumento momentáneamente guardado en *opciones* con los casos especificados. El funcionamiento de cada caso es el siguiente:

- h): ejecuta la función *ayuda* (función creada por aparte).
- f): guarda el argumento pasado a *-f* en la variable *variable_ extraer*. Además se asigna un 1 a la variable *var_ spec*. Esto para registrar que se produjo el caso *-f*.
- \?): muestra un mensaje si se ingresa un argumento inválido.

- :): muestra un mensaje si no se ingresa un argumento para una opción que lo requiere.

Antes del bloque de código mostrado, se declararon ciertas variables por defecto. Lo anterior para establecer lo siguiente:

- Campo a extraer por defecto es *path* y corresponde a la columna 7.
- Cantidad de resultados a extraer por defecto son 5.
- Los valores iniciales de las variables *var_spec* *cant_spec* son cero.

Una vez que se terminan de procesar las opciones indicadas por **getopts**, se requiere que el nombre del archivo (indicado como un argumento posicional) siga siendo *$1*. Para esto se utiliza la siguiente línea de código, que restaura el valor de los argumentos posicionales del script:

```
shift $((OPTIND - 1))
```

Luego de esto, es posible ejecutar con normalidad el código que se creó en el primer inciso del presente ejercicio. Para la parte de código del segundo inciso, se agregó el siguiente bloque antes:

```
if [[ $var_spec = 1 ]]; then
   if [[ $variable_extraer = "path" ]]; then
       columna=7
   elif [[ $variable_extraer = "ip" ]]; then
       columna=1
   elif [[ $variable_extraer = "status" ]]; then
       columna=9
   else
       echo ""
       echo "Argumento para -f no especificado o incorrecto, utilizando path por defecto"
       echo "Posibles opciones: path, ip, status."
       variable_extraer="path"
   fi
fi

if [[ $cant_spec = 1 ]]; then
    cantidad=$cantidad_resultados
fi
```

El primer bloque *If* verifica si la variable *var_spec* cambió a 1, si esto es así, entonces se verifica cual es el valor de la variable *variable_extraer*, de forma que se guarde en la variable *columna* el número de la columna a extraer. Para casos en los que los valores de la variable a extraer no son los esperados, se imprime un mensaje y se mantiene el valor por defecto.

EL segundo bloque *If* verifica si la variable *cant_spec* cambió a 1, si esto es así, entonces se guarda el valor de cantidad en la variable *cantidad*.

Finalmente, se ejecuta la misma serie de comandos descritos en la solución del inciso 2 del presente ejercicio, pero incluyendo las variables *columna* y *cantidad*, con el fin de extraer el campo indicado y mostrar la cantidad deseada de líneas.

A continuación se muestran varios ejemplos del funcionamiento de código.

Caso en por defecto, en el que no se especifica ninguna opción:

```
$ ./ejercicio2.sh access-2025-07.log

Lineas: 8000
Primeras 3 lineas:
192.168.249.130 - - [01/Jul/2025:00:03:25 +0100] "GET /product/1408 HTTP/1.1" 503 452
172.31.255.237 - - [01/Jul/2025:00:06:01 -0700] "PUT /category/music HTTP/1.1" 401 1377
192.0.2.233 - - [01/Jul/2025:00:09:09 +0100] "PUT /api/users HTTP/1.1" 200 496

Verificación de los primeros 5 path mas repetidos:

   1581 /home
    608 /contact
    500 /pricing
    474 /about
    228 /api/users
```

Caso en el que se especifica un campo y cantidad de resultados deseados; en este caso, el campo es *ip* y la cantidad es de 7 resultados.

```
$ ./ejercicio2.sh -f ip -t 7 access-2025-06.log

Lineas: 8000
Primeras 3 lineas:
10.61.213.226 - - [01/Jun/2025:00:02:42 +0530] "GET /category/fashion HTTP/1.1" 200 2627
203.0.113.236 - - [01/Jun/2025:00:05:19 +0200] "GET /home HTTP/1.1" 200 1512
198.51.100.224 - - [01/Jun/2025:00:06:18 +0000] "GET /api/orders HTTP/1.1" 500 408

Verificación de los primeros 7 ip mas repetidos:

    320 127.0.0.1
     13 198.51.100.154
     12 203.0.113.169
     12 198.51.100.189
     12 198.51.100.148
     11 203.0.113.39
     11 203.0.113.185
```

Finalmente, se muestra el resultado de descripción de programa al solicitar la opción de ayuda *-h*:

```
$ ./ejercicio2.sh -h

Uso del script: ejercicio.sh [opciones] [archivo]

EL script muestra lo siguiente sobre el archivo indicado:
```

```
    - Muestra la cantidad de lineas
    - Muestra el contenido de las primeras tres lineas.
    - Muestra los primeros n path, ip o status mas repetidos

Si no se indica un archivo, el archivo no existe o no tiene permisos de lectura, se produce una salida
    - ERROR: Archivo no existe o no tiene permiso de lectura

Opciones disponibles:

    -f,        Permite escoger entre path, ip o status
    -t,        Permite escoger la cantidad de resultados a mostrar

Ejemplo de uso del codigo:
  $./ejercicio2.sh access-2025-06.log
  $./ejercicio2.sh -f status access-2025-06.log
  $./ejercicio2.sh -t 12 access-2025-06.log
  $./ejercicio2.sh -f ip -t 7 access-2025-06.log
```