**1. ¿Qué es la shell? ¿Para qué sirve?**

El término shell se emplea para referirse a aquellos programas que proveen una interfaz de usuario para acceder a los servicios del sistema operativo. Estos pueden ser gráficos o de texto simple, dependiendo del tipo de interfaz que empleen. Los shells están diseñados para facilitar la forma en que se invocan o ejecutan los distintos programas disponibles en la computadora (es un intérprete de comandos).

Entonces, la Shell es en pocas palabras el entorno de escritorio (DE) o Manejador de Ventanas (WM) que utilizamos para trabajar en nuestra PC, sin importar la distribución que utilicemos ya sea por medio de GUI’s (entornos gráficos) o por la terminal respecto a la interacción que requerimos para poder utilizar los servcios y aplicaciones que ofrecen los sistemas operativos.

Entonces, KDE es un Shell, XFCE es un Shell, LXDE es un Shell, iOS es un Shell, Android es un Shell, Windows Phone es un Shell, la terminal es un shell (via bash).

Una shell o intérprete de órdenes o intérprete de comandos es el programa informático que provee una interfaz de usuario para acceder a los servicios del sistema operativo.

Dependiendo del tipo de interfaz que empleen, los shells pueden ser:

* De líneas texto (CLI, Command-Line Interface, interfaz de línea de comandos),
* Gráficos (GUI, Graphical User Interface, interfaz gráfica de usuario),
* De lenguaje natural (NUI, Natural User Interface, interfaz natural de usuario).

Los shell son necesarios para invocar o ejecutar los distintos programas disponibles en la computadora.

### 2. ¿En qué espacio (de usuario o de kernel) se ejecuta?

En el espacio de usuario.

**3. Si pensamos en el funcionamiento de una shell básica podríamos detallarlo secuencialmente de la siguiente manera:**

* Esperar a que el usuario ingrese un comando
* Procesar la entrada del usuario y obtener el comando con sus parámetros
* Crear un nuevo proceso para ejecutar el comando, iniciarlo y esperar que retorne
* Presentar la salida (de haberla) al usuario
* Volver a empezar.

**Este tipo de comportamiento, típico de las shell interactivas, se conoce como REPL (Read- Eval-Print Loop, Ciclo de leer, interpretar e imprimir).**

**Analice cómo implementaría este ciclo básico de interpretación de scripts.**

Este tipo de comportamiento, típico de las shell interactivas, se conoce como REPL (ReadEval-Print Loop, Ciclo de leer, interpretar e imprimir). Analice cómo implementaría este ciclo básico de interpretación de scripts.

Para implementar un REPL en Lisp, es necesario únicamente implementar estas tres funciones y una función de bucle infinito (obviamente, la implementación de eval será complicada, dado que se debe también implementar todas las funciones primitivas como car o + y operadores especiales como if.). Hecho esto, un REPL básico es una única línea de código: (loop (print (eval (read)))).

Una posible implementación de eval es un intérprete recursivo que actúa sobre el árbol de sintaxis abstracta creado por read. Otra posibilidad es compilar el árbol de sintaxis en código máquina y ejecutarlo. Las implementaciones reales de REPLs en Lisp son en ocasiones mucho más complicadas.

### 4. Investigue la system call fork:

#### a. ¿Qué es lo que realiza?

Crea un nuevo proceso duplicando el proceso existente desde donde se llamo la funcion. El proceso existente de donde la función fue llamada se convierte en el proceso padre, y el proceso creado se convierte en el proceso hijo. Los procesos resultantes de este proceso son idénticos a su padre, salvo que tienen distinto número de proceso (PID). La operación fork crea un espacio de direcciones separado para el nuevo proceso hijo. Este tiene una copia exacta de todos los segmentos de memoria del proceso padre.

#### b. ¿Qué retorna?

La llamada a fork retorna al proceso padre el identificador del proceso hijo y retorna un cero al proceso hijo.

#### c. ¿Para qué podrian servir los valores que retorna?

A partir de los resultados se puede identificar cuál de los procesos es el padre y cuál es el hijo. También puede usarse para hacer frente a un error.

#### d. ¿Por qué invocaria a la misma al implementar una shell?

Cuando se ejecuta un proceso desde una shell, la shell hace un fork() antes de ejecutar el proceso. Esto se produce así porque cuando alguien llama a una instrucción de la familia de exec esto no crea un nuevo proceso, sino que reemplaza la memoria e instrucciones del proceso actual con las del proceso que se quiere ejecutar. Así que cuando bash quiere ejecutar algo primero tiene que hacer un fork() y luego ejecutar. Si no lo hiciera así, se ejecutaría el proceso pero no podriamos acceder mas a la terminal bash.

<https://oskarth.com/unix01/>

### 5. Investigue la system call exec:

#### (a) ¿Para qué sirve?

Es una funcionalidad de los sistemas operativos que permiten correr un archivo ejecutable en el contexto de un proceso ya existente, reemplazándolo.

#### (b) ¿Comó se comporta?

Esto es llamado "overlay". El PID original no cambia, pero el código de máquina, datos, heap y stack del proceso son reemplazados por los del nuevo programa.

#### (c) ¿Cuáles son sus diferentes declaraciones POSIX?

POSIX es el acrónimo de Portable Operating System Interface, y X viene de UNIX como seña de identidad de la API.

int execl(char const \*path, char const \*arg0, ...); int execle(char const \*path, char const \*arg0, ..., char const \*envp[]); int execlp(char const \*file, char const \*arg0, ...); int execv(char const \*path, char const \*argv[]); int execve(char const \*path, char const \*argv[], char const \*envp[]); int execvp(char const \*file, char const \*argv[]);

La base de cada uno es "exec" (execute), seguido de uno o más letras:

* e - Se pasa explícitamente una matriz de punteros a variables de entorno a la nueva imagen de proceso.
* l - Los argumentos de la línea de comando se pasan individualmente (una lista) a la función.
* p - Utiliza la variable de entorno PATH para encontrar el archivo nombrado en el argumento del archivo para ser ejecutado.
* v - Los argumentos de la línea de comando se pasan a la función como una matriz (vector) de punteros.

path: El argumento especifica el nombre de ruta del archivo para ejecutar como la nueva imagen de proceso. Los argumentos que comienzan en arg0 son punteros a los argumentos que se pasarán a la nueva imagen de proceso. El valor argv es una matriz de punteros a argumentos.

arg0: El primer argumento arg0 debe ser el nombre del archivo ejecutable. Usualmente es el mismo valor que el pathargument. Algunos programas pueden confiar incorrectamente en este argumento proporcionando la ubicación del ejecutable, pero no hay garantía de esto ni está estandarizado en todas las plataformas.

envp: Argument envp es una matriz de indicadores para la configuración del entorno. Las llamadas ejecutadas por el administrador que terminan con un e modifican el entorno para la nueva imagen de proceso al pasar una lista de configuraciones de entorno a través del argumento envp. Este argumento es una matriz de punteros de caracteres; cada elemento (excepto el elemento final) apunta a una cadena terminada en nulo que define una variable de entorno.

### 6. Investigue la system call wait:

#### a. ¿Para qué sirve?

A call to wait() blocks the calling process until one of its child processes exits or a signal is received. After child process terminates, parent continues its execution after wait system call instruction.

Child process may terminate due to any of these:

* It calls exit();
* It returns (an int) from main
* It receives a signal (from the OS or another process) whose default action is to terminate.

#### b. Sin ella, ¿qué sucedería, pensando en la implementación de la shell?

Sin el wait no se podría esperar a que una línea de comando termine de ejecutarse para poder recibir la siguiente.

Every shell is structured as the following loop:

\* print out a prompt

\* read a line of input from the user

\* parse the line into the program name, and an array of parameters

\* use the fork() system call to spawn a new child process

\* the child process then uses the exec() system call to launch the specified program

\* the parent process (the shell) uses the wait() system call to wait for the child to terminate

\* when the child (i.e. the launched program) finishes, the shell repeats the loop by jumping to 1.

Although most of the commands people type on the prompt are the name of other UNIX programs (such as ls or more), shells recognize some special commands (called internal commands) which are not program names. For example, the exit command terminates the shell, and the cd command changes the current working directory. Shells directly make system calls to execute these commands, instead of forking a child process to handle them.

7. En una shell, ¿en qué se diferencian los comandos internos (o built-in) de los externos?

Comandos internos: Son los nativos de la terminal Bash. Ejecutando el comando “help” en la terminal se obtiene un listado de los comandos internos disponibles. Podemos hacer uso de type y where para saber el tipo de comando utilizado.

Comandos externos: Son los comandos no nativos de la terminal, deben de ser llamados por el intérprete de comandos para ser ejecutados, son todos los que se encuentran en la variable $PATH.

La diferencia fundamental es que los internos están incorporados a la consola y se pueden ejecutar directamente, mientras que para los  externos hay que indicar la ruta hasta la ubicación del comando.

Para los comandos externos puede ser que no tengamos que indicar la ruta hasta la ubicación del mismo de forma explícita, si esta ruta está incluida en la variable de entorno PATH.

También debemos tener precaución en el caso de que el comando exista tanto de forma interna y externa, ya que las dos versiones del comando pueden dar resultados distintos, por lo que si queremos estar seguros de que estamos ejecutando la versión externa debemos indicar la ruta del comando. Por ejemplo: pwd ó /bin/pwd.

¿Qué función cumple la variable de entorno PATH? Hint: <https://www.geeksforgeeks.org/internaland-external-commands-in-linux/>

PATH es una variable de entorno. Las variables de entorno contienen información a la que se accede a través del nombre de la variable.

Esta variable de entorno informa al shell (en la mayoría de los casos BASH) dónde se encuentran los programas binarios que puedo ejecutar en el sistema, sin tener que llamarlos por su ruta absoluta).

PATH contiene una cadena de rutas de directorios separadas por dos puntos.

Podemos ejecutar alguno de los siguientes comando para observar qué es lo que tiene:

echo $PATH o echo $PATH | tr : \\n

/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin

Por lo tanto PATH contiene una lista de directorios separados por dos puntos. Estos son los directorios en los que el shell busca el comando que uno escribe desde el teclado.

Desde el momento en que el shell encuentra el comando detiene la búsqueda y ejecuta el comando encontrado.

Algunas variables importantes:

SHELL=/bin/bash  (el tipo de shell en uso)

TERM=xterm  (el programa de terminal por defecto)

USER=pepito  (el nombre de usuario)

PWD=/home/pepito  (la ruta por defecto del usuario)

LANG=es\_ES.utf8  (el juego de caracteres de idioma)

DESKTOP\_SESSION=xfce  (el entorno de escritorio)

PATH=/usr/local/bin:/usr/bin:/bin:/usr/local/sbin:/usr/sbin:/sbin

<https://es.ccm.net/faq/315-bash-la-variable-de-entorno-path>

**Scripts**

1. Realice un script que guarde en el archivo /tmp/usuarios los nombres de los usuarios del sistema cuyo UID sea mayour a 1000.

while IFS=: read f1 f2 f3 f4 f5 f6 f7

do

if [$f3 -gt 1000]

then

$f1 >> /tmp/usuarios

fi

done < /etc/passwd

IFS -> indica un delimitador, y guarda los resultados en las variables indicadas despues (f1,f2...etc)

1. Implemente un script que reciba como parametro el nombre de un proceso e informe cada 15 segundos cuantas instancias de ese proceso estan en ejecucion.

while :

pgrep $1 | wc -l

sleep 900

done

Asumo que el parametro que recibe mi script queda cargado en $1 (porque sino no se como se hace)

pgrep -> lista el pid de los procesos que cumplan con la condicion.

wc -> cuenta palabras. (lineas,palabras,caracteres)

1. Desarrolle un script que guarde en un arreglo todos los arcchivos del directorio actual (incluyendo sus subdirectorios) para los cuales el usuario que ejecuta el script tiene los permisos de **ejecucion**. Luego, implemente las siguientes funciones: a. cantidad: Imprime la cantidad de archivos que se encontraron b. archivos: Imprime los nombres de los archivos encontrados en orden alfabetico.

array=(`find . -type f -perm -111`)

a. cantidad

echo ${#array[@]}

b. archivos

IFS=$'\n' sorted=($(sort <<<"${array[\*]}"))

echo ${sorted[@]}

1. Se le ha encomendado organizar las fotos (en formato jpg) de todos los eventos de los que su empresa ha participado en el ultimo anio, los cuales se encuentran organizados en directorios con el nombre del evento. Para facilitar su busqueda posterior, los archivos deben tener nombres que sigan el siguiente patron: **EVENTO-N.jpg**, donde:
   * **EVENTO** es el nombre del evento (el del directorio que se esta procesando)
   * **N** es un indice de foto, comenzando en 1

Realice un script que renombre los archivos de cada subdirectorio del directorio actual siguiendo lo especificado en el parrafo anterior. **Ejemplo:** dada la siguiente estructura de archivos y directorios:

bashconf15/

DSC1050.jpg

DSC1051.jpg

DSC1052.jpg

DSC1053.jpg

jsconf-14/

DSC01230.jpg

DSC01231.jpg

DSC01232.jpg

DSC01235.jpg

Se desea terminar con la siguiente estructura luego de ejecutar su script:

bashconf15/

bashconf15-1.jpg

bashconf15-2.jpg

bashconf15-3.jpg

bashconf15-4.jpg

jsconf-14/

jsconf-14-1.jpg

jsconf-14-2.jpg

jsconf-14-3.jpg

jsconf-14-4.jpg

|  |
| --- |
|  |
| #!/bin/bash |
|  |  |
|  | for element in \*; do |
|  | indice=1 |
|  | if [ -d "$element" ]; then #Si es un directorio, accedo |
|  | cd "$element" |
|  | for file in \*; do |
|  | if [ -f "$file" ] && [ ${file##\*.} == "jpg" ]; then |
|  | extension=${file##\*.} |
|  | actual\_file\_name=${file%.\*} |
|  | directory\_name=${element%.\*} |
|  | mv "$actual\_file\_name.$extension" "$directory\_name-$indice.$extension" |
|  | ((indice++)) |
|  | fi; |
|  | done; |
|  | cd .. |
|  | fi; |
|  | done; |
|  |  |

1. Escriba un script que liste en orden alfabetico inverso el contenido del directorio actual. Es decir, si el contenido son los archivos:

archivo\_1.txt articulo.doc directorio directorio\_2 script.sh

Se espera que el script los liste de la siguiente manera:

script.sh directorio\_2 directorio articulo.doc archivo.txt

|  |
| --- |
| #!/usr/bin/env bash |
|  | ls | sort -r |

1. Realice un script que copie todos los archivos del directorio home del usuario que lo ejecuta, a un subdirectorio del mismo llamado backup cambiandoles el nombre para que este en **MAYUSCULAS**. **No se deben procesar los subdirectorios del home del usuario**, unicamente los archivos ubicados directamente en este. Si el directorio backup existe al iniciar el script, el contenido del mismo debe borrarse antes de copiar los archivos. **Ejemplo:** si el home del usuario contiene:

home/

mi\_usuario/

so/

practica1.pdf

ejercicios/

ejercicio-1.sh

ejercicio-2.sh

archivo1.txt

mi-script.sh

otro\_archivo.txt

se espera tener lo siguiente luego de la ejecucion del script:

/

home/

mi\_usuario/

backup/

ARCHIVO1.TXT

MI-SCRIPT.SH

OTRO\_ARCHIVO.TXT

so/

practica1.pdf

ejercicios/

ejercicio-1.sh

ejercicio-2.sh

archivo1.txt

mi-script.sh

otro\_archivo.txt

|  |
| --- |
| #!/usr/bin/env bash |
|  | fileArray=$( find ~/ -maxdepth 1 -type f | cut -d/ -f4 ) |
|  | echo ${fileArray[@]} |
|  | rm -r ~/backup |
|  | mkdir ~/backup |
|  | for file in ${fileArray[@]};do |
|  | cp ~/$file ~/backup/$(echo $file | tr 'abcdefghijklmnopqrstuvwxyz' 'ABCDEFGHIJKLMNOPQRSTUVWXYZ') |
|  | done |

1. Un escritor tiene organizados los capítulos de su próximo libro en distintos archivos de texto plano en un mismo directorio, y le ha solicitado ayuda para concatenar el contenido de cada uno de ellos en un único archivo final llamado libro.txt, de modo tal que éste último contenga el texto de todos los otros archivos, uno luego del otro. Puede asumir que los archivos de los capítulos tienen nombres alfabéticamente ordenados:

capitulo-01.txt,capitulo-02.txt, ..., capitulo-48.txt, por ejemplo.

Tip: man cat.

|  |
| --- |
| #!/usr/bin/env bash |
|  | arrayFile=$(find -maxdepth 1 -type f ! -path . | sort ) |
|  | for file in ${arrayFile[@]}; do |
|  | cat $file >> libro.txt |
|  | done |

## Redes y Sistemas Operativos

### 1. La herramienta netcat provee una forma sencilla de establecer una conexión TCP/IP. En una terminal levante una sesión de netcat en modo servidor, que escuche en la IP 127.0.0.1 (localhost) en un puerto a elección. En otra terminal conéctese, también vía netcat, al servidor recién levantado. Interactúe y experimente con ambas terminales.

Para levantar una sesión netcat en modo servidor se debe escribir el comando nc -l [ port ]. Este parámetro (l = listen) se utiliza para indicar a nc que se tiene que quedar esperando recibir pedidos de conexión en lugar de inicializar una conexión con un host remoto. El número de puerto tiene que ser mayor a 1024 (uno no privilegiado).

Para abrir una conexión ahora es necesario que en otra terminal ejecutamos el comando nc [ ip ] [ port ]

Ejemplo: nc -l 1234 → Servidor nc 127.0.0.1 1234 → Cliente

### 2. netcat también es bueno al momento de transmitir archivos sobre una red TCP/IP. Utilizando dos terminales como se hizo en el ejercicio anterior, transmita el archivo /etc/passwd desde una sesión de netcat hacia la otra. Tip: recordar pipes y redirecciones.

nc -l 1234 → Servidor cat /etc/passwd | nc 127.0.0.1 1234 → Cliente

1. Desarrolle un script que reciba en su entrada estándar una lista de hosts e imprima en su salida estándar únicamente aquellos que tienen el puerto 80 abierto. Cuando un host no tiene el puerto 80 abierto, netcat tardará varios segundos en determinar que la conexión no se puede establecer.

Tip: utilizar la opción -w de netcat para disminuir el tiempo de timeout.

|  |
| --- |
| #!/bin/bash |
|  |  |
|  | # $\* devuelve todos los parametros dentro de un arreglo |
|  | # $? devuelve el estado de salida del último comando ejecutado |
|  |  |
|  | # nc -z Specifies that nc should just scan for listening daemons, without |
|  | # sending any data to them. It is an error to use this option in |
|  | # conjunction with the -l option. |
|  |  |
|  | # nc -w Connections which cannot be established or are idle timeout after |
|  | # timeout seconds. The -w flag has no effect on the -l option, |
|  | # i.e. nc will listen forever for a connection, with or without the |
|  | # -w flag. The default is no timeout. |
|  |  |
|  |  |
|  | if [ $# -eq 0 ] |
|  | then |
|  | echo "No ha ingresado ningun host." |
|  | exit 1; |
|  | fi |
|  |  |
|  | array=() |
|  |  |
|  | for host in $\* |
|  | do |
|  | nc -z -w 5 $host 80 |
|  |  |
|  | if [ $? = 0 ] |
|  | then |
|  | array[${#array[\*]}]=$host |
|  | fi |
|  | done |
|  |  |
|  | echo "${array[\*]}" |

4. Desarrolle un script que reciba en su entrada estándar una lista de hosts con el puerto 80 abierto y, para cada uno, realice un requerimiento HTTP GET a la URI raíz y devuelva el valor del campo Content-Length de la respuesta. Deberá ser posible utilizar como entrada estándar la salida estándar del script anterior.

1. echo www.google.com www.debian.org www.linti.unlp.edu.ar | ./cl.sh
2. #=> www.google.com: 262
3. #=> www.debian.org: 470

#=> www.linti.unlp.edu.ar: 223

Tip: printf "GET / HTTP/1.0\r\n\r\n"| ...

|  |
| --- |
| #!/bin/bash |
|  |  |
|  | if [ $# -eq 0 ] |
|  | then |
|  | echo "Pelotudo no dijiste ningun host con el puerto 80 abierto." |
|  | exit 1; |
|  | fi |
|  |  |
|  | array=() |
|  |  |
|  | for host in $\* |
|  | do |
|  | result=`printf "HEAD / HTTP/1.0\r\n\r\n"| nc $host 80 | grep "Content-Length" | grep -o '[0-9]\*'`; |
|  | echo "$host: $result" |
|  | done |

### 5. Intérprete y describa qué es lo que hace el siguiente fragmento de código extraído de la man page de netcat.

#### 1 → $ rm −f /tmp/f; mkfifo /tmp/f

#### 2 → $ cat /tmp/f | /bin/sh −i 2>&1 | nc −l 127.0.0.1 1234 > /tmp/f

### Tip: man mkfifo

mkfifo es un comando que se utiliza para crear “named pipes”. En la línea 1, se está definiendo el named pipe bajo el nombre “/tmp/f”. La instrucción anterior se realiza para que, en el caso de que dicho pipe ya exista, al intentar crearlo con mkfifo no haya error alguno.

/bin/sh o dash permite generar una shell. Con el parámetro -i se fuerza a que la misma se comporte de forma interactiva. Seguido a esto hay una forma de redirección. [n1]<&n2 Entrada estándar duplicada (o n1) del descriptor de archivo n2.

En el tercer paso del pipe se establece un servidor escuchando en el local host en el puerto 80 usando netcat.

Al hacer esto, crea un fifo en /tmp/f y hace que nc escuche en el puerto 1234 de la dirección 127.0.0.1 en el lado del 'servidor', cuando un 'cliente' establece un conexión exitosa a ese puerto, / bin / sh se ejecuta en 'servidor' lado y el indicador de shell se da al lado 'cliente'.

6. Desarrolle un script que permita al usuario chatear con otra instancia del mismo script.  
Para ello, el script deberá recibir como parámetro si va a funcionar como (c)liente o como (s)ervidor. También deberá recibir como parámetro un nickname para el usuario. Por  
ejemplo, para invocar el script en modo servidor con el nick jvg, debería ejecutar:  
./nc-chat.sh s jvg  
Además, los mensajes transmitidos deben comenzar con la fecha, hora y nick del emisor.  
Por ejemplo:  
Thu Mar 12 13:03:14 ART 2015, jvg says:  
Knock knock Neo.

|  |
| --- |
| #!/bin/bash |
|  |  |
|  | # Agradecimientos a Franco Borrelli |
|  |  |
|  | if [ "$#" != 2 ] |
|  | then |
|  | echo "Pasame los parametros bien imbecil" |
|  | exit 1; |
|  | fi |
|  |  |
|  | if [ "$1" != 's' ] && [ "$1" != 'c' ] |
|  | then |
|  | echo "El primer parametro tiene que ser una c o una s, no es muy dificil"; |
|  | exit 2; |
|  | fi |
|  |  |
|  | if [ "$1" == 's' ] |
|  | then |
|  | echo "Started listening on port 1234." |
|  | echo "Stream (Press ctrl + c to end session) >" |
|  | awk -v date="$(date)" -v user="$2" '{ print date,",",user,"says:\n",$0; fflush()}' | nc -l 12345 |
|  | else |
|  | awk -v date="$(date)" -v user="$2" '{ print date,",",user,"says:\n",$0; fflush()}' | nc localhost 12345 |
|  | fi |