Pontificia Universidad JAVERIANA Bogotá

Sistemas Operativos

Taller T11 – unix/cocalc/compilación /argumentos

Objetivo: comprender e iniciar el uso más básico de comandos, archivos y procesos en un sistema operativo.

Instrucciones:

1. **Grupos**: el taller se realiza en grupos definidos en clase o en la plataforma virtual del curso.
2. **Entrega**: la entrega consiste en un informe en PDF que se sube al buzón asignado en la plataforma virtual del curso. Este informe debe contener todas las evidencias: respuestas a preguntas, resultados de los experimentos o ejecuciones solicitadas en el orden solicitado, resultados de pruebas y análisis solicitados, incluidas capturas de pantalla asociadas. Si el taller requiere el desarrollo de código, el informe y todos los archivos de código, texto o de resultado de ejecución deben incluirse en un archivo comprimido (zip ó 7z).
3. **Plagio**: Cualquier evidencia de plagio o copia tiene como consecuencia calificación de cero puntos en la asignación. Toda entrega será analizada con herramientas anti-plagio.
4. **Plataforma**: todas las operaciones de este taller se realizarán bajo plataforma Linux, en cualquier forma (Linux instalado en computador con doble boot ó Linux instalado en máquina virtual). Si Ud. no dispone de alguna de las opciones, puede usar como último recurso la herramienta https://cocalc.com/ y crear una cuenta, crear un proyecto, cargar archivos, y usar un terminal de Linux.

---

1. Creación de una Jerarquía de Directorios.

a) En su directorio HOME cree dos directorios: **Talleres** y **Teoría** usando el comando **`mkdir`.**

b) Baje de la asignación del aula virtual los archivos taller.txt, *dormilon.c* y *ListaArreglos.tgz* (Carpeta: Laboratorios/Taller1) y colóquelos en el directorio **Talleres desde el directorio HOME.** Muévase al directorio **Talleres** usando el comando:

```
$ cd Talleres
```

c) Ejecute ***pwd*** en el mencionado directorio y luego haga ***cd***, ¿qué pasa? ¿Qué hacen estos comandos? ¿En qué directorio quedan después de hacer cd?

d) Descomprima desde la línea de comandos el archivo *ListaArreglos.tgz y colóquelo dentro del directorio* ***Talleres.*** Para descomprimir el archivo debe ejecutar los siguientes comandos*:*

```
$ gunzip ListaArreglos.tgz $ tar –xvf ListaArreglos.tar
```

Esto debe crear dentro de Talleres, el directorio **Lista Arreglos**

2. Lectura de Archivos

a) Lea el contenido del archivo **`/etc/passwd`** usando los comandos **`head, tail, more`** y **`cat`**. ¿Cuál es la diferencia entre estos cuatro comandos? Realice las mismas operaciones sobre el archivo taller.txt. ¿Cómo se leen las primeras 4 líneas de un archivo?

b) Utilizando el manual del sistema (comando **`man)`** investigue qué hacen los comandos **`uname, grep, cp`** ***y*** **`chmod`**. Utilizando el comando **`uname`** y los flags correspondientes indique los detalles del procesador y del sistema operativo sobre los que está trabajando.

Detalles del procesador y SOP_____

Comando y flags usados: _____

En el directorio donde colocó el archivo *dormilon.c* ejecute e indique qué hace el siguiente comando:

```
$ cat dormilon.c | grep include | wc –l
```

¿El número que se obtiene finalmente, qué está contando?

_____

_____

c) Ubíquese dentro del directorio *ListaArreglo*s. Escriba, utilizando pipes, una orden al shell que usando el comando **`ls`** (y otros comandos) cuente el número de archivos con extensión **.in** en el directorio. La salida de ese conjunto de comandos debe escribirse al archivo **`tmp.txt`.**

Escriba los comandos que usa para satisfacer este requerimiento:

_____

_____

3. Compilación de programas y procesos

a) Ubíquese en el directorio donde se encuentra el archivo *dormilon.c*. Realice su compilación de la siguiente forma:

```
$ gcc –o dormilon dormilon.c
```

Ejecútelo en segundo plano (usando &).

```
$./dormilon &
```

b) invoque los comandos **ps** y **top** para observar los procesos que hay en el sistema. Use **kill** con el flag -9 para matar uno de los procesos (trate de hacerlo primero con el que dice <defunc> . Invocar nuevamente **ps**. Finalmente, use kill -9 con el otro proceso zombie y verifique que efectivamente todos los procesos terminan.

```
$ kill -9 piddelproceso
```

4. Permisos y atributos de archivos

a) Elimine los permisos de lectura al archivo **dormilon.c,** luego agregue de nuevo el permiso de lectura, pero sólo al propietario del archivo. Recuerde que usuario (u), grupo (g), los que no son del grupo (o) y todos (a), tienen permisos de lectura (r), escritura (w) y ejecución (x).

Ejemplos: Si se quiere eliminar los permisos de lectura al grupo para el archivo *pepe.tex*, se haría:

```
$ chmod g -r pepe.tex
```

si se quiere hacer un script ejecutable (**tuscript**), escribe

```
$ chmod +x tuscript
```

Escriba los comandos utilizados para eliminar permisos de lectura:

_____

Escriba los comandos para agregar permiso de lectura al propietario:

_____

---

5. Compilación y Argumentos

1. Realice un programa en lenguaje C, llamado **menor,** que recibe en su invocación 2 enteros y como salida el programa informa si el primer número es mayor, menor o igual que el segundo. A continuación, se presentan 3 ejemplos de invocaciones y posibles salidas:

```
$./menor  3 4
3 es menor que 4

$./menor  34 34
34 es igual a 34

$./menor  60 1
   60  s mayor que 1
```

2. Abajo se muestra un programa sencillo en Lenguaje C que lee un archivo de texto usando las llamadas OPEN y READ. En muchos casos un archivo no se puede abrir,

de modo que se requiere programar con validación. Para estudiar estos casos, haga las siguientes pruebas.

a. Modifíquelo para que reciba como argumento el nombre del archivo que leerá.
b. Pruebe su programa con archivos que existen (que Ud haya creado previamente), y con nombres de archivos que no existen. ¿Qué sucede en cada caso?
c. Modifique el programa para evitar errores en la ejecución o ejecuciones no deseadas, usando la llamada **perror()**, de modo que cuando el archivo no exista el programa termine sin realizar la lectura.

```
// Programa de ejemplo
// Llamadas a sistema OPEN y READ.

#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include<unistd.h>

int main()
{
  int fd, sz;

  char *c = (char *) calloc(100, sizeof(char));

  fd = open("test.txt", O_RDONLY);

  sz = read(fd, c, 10);

  printf("se llamo a read(%d, c, 10). Devolvio que "
                " %d bytes fueron leidos.\n", fd, sz);

  c[sz] = '\0';

  printf("Esos bytes soin los siguientes: %s\n", c);
}
```