Introducción a Unix: Unidad 2

Curso basado en uno propuesto por William Knottenbelt, UK, 2001

Daniel Millán, Nora Moyano & Evelin Giaroli

Facultad de Ciencias Aplicadas a la Industria, UNCuyo.

Junio de 2017

Introducción

Esta unidad se divide en dos partes:

A – Manejo de archivos

Los temas que se cubrirán en la **Primer Parte** son:

1. Permisos de archivos y directorios en más detalle y cómo estos se pueden cambiar.
2. Maneras de examinar el contenido de los archivos.
3. ¿Cómo encontramos archivos cuando no sabemos su ubicación exacta?
4. ¿Cómo buscamos una cadena de caracteres en uno o varios archivos?
5. Formas de ordenar archivos.
6. Herramientas para la compresión de archivos y copias de seguridad.
7. Manipulación de medios extraíbles.

B – Manejo de procesos

Los temas que se cubrirán en la **Segunda Parte** son:

8. El concepto de un proceso.
9. Pasar la salida de un proceso como entrada a otro utilizando tuberías.
10. Redirección de la entrada y la salida del proceso.
11. Procesos relacionados con *shell*.
12. Control de otros procesos.

    Herramientas útiles:

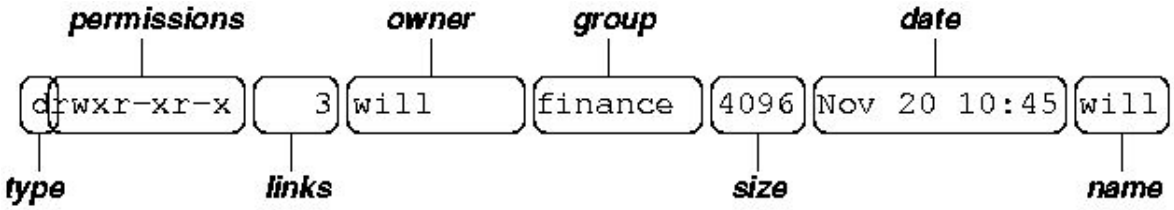
13. Procesamiento de archivos de texto con **sed** y **awk**.
14. Páginas del manual

1. Permisos de archivos/directorios

Como hemos visto en la Unidad 1, cada archivo o directorio en un sistema Unix tiene tres tipos de permisos.

Los permisos son:

lectura (**r**), escritura (**w**), ejecución (**x**)



Cada tipo de permiso describe qué tipo de operaciones se pueden realizar por diferentes categorías de usuarios.

Las tres categorías de usuarios son:

propietario (**u**), grupo (**g**), otros (**o**)

- Dado que los archivos y directorios son entidades diferentes, la interpretación de los permisos asignados a ellos también los es:

Permiso	Archivo	Directorio
Lectura *r*	El usuario puede **ver el contenido** del archivo.	El usuario **puede listar los archivos** en el directorio.
Escritura *w*	El usuario puede **modificar los contenidos** del archivo.	El usuario puede **crear** nuevos archivos **y eliminar archivos** existentes en el directorio.
Ejecución *x*	El usuario puede utilizar el nombre de **archivo como un comando** UNIX.	El usuario puede **acceder al directorio**, pero no puede listar los archivos a menos que tenga permiso de lectura. El usuario puede leer los archivos si tiene permiso de lectura en ellos.

Interpretación de los permisos de archivos y directorios

- Los permisos de archivos y directorios sólo pueden ser modificados por sus propietarios, o por el superusuario (root), mediante el uso de la utilidad del sistema **chmod.**

  $chmod *opciones archivos*

- **chmod** acepta opciones en dos formas.
  - como una secuencia de 3 dígitos octales (0–7). Cada dígito octal representa los

permisos de acceso para el usuario, grupo y otros.

r*2^2 + w*2^1 + x*2^0 ➔permiso (donde r,w,x es 0=No o 1=Si)

- simbólicamente, utilizando los símbolos:
  - **u** (usuario), **g** (grupo), **O** (otros), **a** (todo = **ugo**)
  - **R** (lectura), **w** (escritura), **x** (ejecutar),
  - **+** (suma permiso), **-** (quita permiso), **=** (asigna permisos)

- **chown/chgrp** cambian propietario/grupo de un archivo o directorio

Ejemplo 1:

```
$chmod 600 private.txt
```

Define los permisos de private.txt a rw-------. Es decir que. solo el propietario puede leer y escribir el archivo.

Example 2:

```
$chmod ug=rw,o-rw,a-x *.txt
```

Define los permisos de todos los archivos que terminan con extensión *.txt a rw-rw----. Por lo tanto el propietario y los usuarios del grupo pueden leer y escribir los archivos, mientras que otros usuarios no tienen ningún tipo de permiso.

Example 3:

chmod además posee la opción -R la cual permite modificar de forma recursiva los permisos, e.g.

```
$chmod -R go+r play
```

posibilita a los usuarios del grupo y a otros usuarios leer el directorio play así como todos sus directorios.

2. Inspección de archivos

- Además de **cat**, **less** y **more** existen otras órdenes muy útiles que pueden ser empleadas: **file**, **head**, **tail**, **od**, ...
  - **file** analiza el contenido de un archivo, brinda una descripción de alto nivel sobre qué tipo de archivo parece ser.

```
$file myprog.c webpage.html
 myprog.c:      C program text
 webpage.html:  HTML document text
```

- **head** y **tail** muestran las primeras y últimas líneas de un archivo. Además permiten especificar el número de líneas, y en el caso de tail mediante la opción –f es posible *monitorear* un archivo que cambia en el tiempo.
- **od** muestra el contenido de un archivo, de texto o binario, en una variedad de formatos (“c” ASCII, “o” octal, “x” hexadecimal...)

```
$cat hola.txt
$od -c hola.txt
```

3. Búsqueda de archivos

- Al menos tres formas de encontrar archivos cuando no se conoce su ubicación exacta: **find**, **which**, **locate**
- **find** se utiliza cuando se tiene una idea aproximada del árbol de directorios donde del archivo podría estar (o si se está dispuesto a esperar un tiempo), se puede utilizar

  ```
  $find directorio -name archivo
  ```

- **find** puede buscar archivos por:
  - tipo: "-type f" archivos, "-type d" directorios
  - permisos: "-perm o=r" archivos y directorios que son leíbles por otros
  - tamaño: "-size" etc.
  - Además es posible ejecutar comandos sobre los archivos

  ```
  $find . -name "*.txt" -exec wc '{}' ';'
  ```

  Cuenta el número de líneas en cada archivo de texto en y por debajo del directorio actual "." '{}' se sustituye por el nombre de cada archivo encontrado y el ';' termina la cláusula -exec. Pruebe: wc -w, wc -c, wc -wc.

- **which** se utiliza para averiguar donde se almacena un programa de aplicación o utilidad del sistema escribiendo

  ```
  $which ls
  /bin/ls
  ```

- **locate** busca archivos de forma mucho más rápida que **find**

  ```
  $locate *txt
  ```

  - **locate** almacena todos los nombres de ficheros en el sistema en un índice que por lo general se actualiza sólo una vez al día.
  - **locate** no encontrará los archivos que se han creado muy recientemente.
  - **locate** puede informar un archivo como si estuviera presente, aunque el archivo haya sido recientemente eliminado.
  - **locate** no permite buscar archivos por tamaño, permisos, etc.

Nota: **find** es una herramienta muy útil y potente, dedique un cierto tiempo a inspeccionar sus opciones (**man find**, y google).

4. Búsqueda de texto en archivos

- **grep** (General Regular Expression Print) es una gran herramienta que permite buscar

dentro de archivos.

```
$grep opciones patrón archivos
```

- **grep** busca un dado "patrón" en todos los "archivos" (o en la entrada estándar -ver procesos- si no se nombran archivos), imprime las líneas donde encuentra alguna coincidencia.
- Algunas opciones útiles que grep ofrece son:
  "-c" imprime el número de líneas que coinciden, "-i" ignora mayúsculas, "-v" imprime las líneas que no coinciden y "-n" imprime el número y el contenido de la línea coincidente.
  Ejemplo:
  ```
  $grep -vi hola *.txt
  ```
  Busca en todos los archivos de texto en el directorio actual aquellas líneas que no contienen ninguna forma de la palabra hola (por ejemplo, Hola, HOLA, hOLA,...).
- ¿Cómo buscar todos los archivos recursivamente que contienen un cierto patrón? ➔ pasar la salida de **find** en **grep**

  ```
  $grep patrón `find . -name "*.txt"`
  ```
- Para más opciones ver **man grep** y google.

5. Ordenar archivos

- Hay dos herramientas útiles para ordenar archivos en UNIX:

  ```
  $sort archivos
  ```

  ```
  $uniq archivo
  ```
- **sort** ordena las líneas contenidas en un grupo de archivos alfabéticamente o numéricamente (-n)
- **uniq** elimina líneas adyacentes duplicadas de un archivo
- Cómo es común en UNIX estos comando suelen combinarse para dar una herramienta más potente (vea tuberías en la parte B de esta unidad)

  ```
  $sort –n *.txt | uniq
  ```

6. Compresión y copias de seguridad

- Existe una gran variedad de herramientas para realizar copias de seguridad y comprimir archivos. Los más útiles son:
- **tar, cpio, compress, gzip**
- **tar** (*tape archive*) crea copias de seguridad de directorios completos (comúnmente) en un único *registro*. El nuevo registro contiene archivos e información acerca de ellos, como ser sus nombres, propietario, marcas de tiempo y permisos de acceso.
- **tar** NO realiza ningún tipo de compresión por defecto.

  ```
  $tar -cvf archivo.tar inputs
  ```
  crea "c" un *archivo.tar*

`$tar –tvf archivo.tar` lista "t" el contenido de *archivo.tar*

`$tar –xvf archivo.tar` restaura "x" el contenido de *archivo.tar*

- Para comprimir cada elemento del registro con **gzip** utilizar

  `$tar -cvfz archivo.tgz inputs`

- **compress y gzip** son utilidades para comprimir y descomprimir archivos individuales (que pueden o no ser ficheros de archivo). Para comprimir archivos, utilice:

  `$compress archivo`

  `$gzip archivo`

- En cada caso *archivo* será reemplazado por *archivo.Z* o *archivo.gz*, es decir se realiza *in–place* (el archivo original es reemplazado por uno que está comprimido).
- Para descomprimir

  `$compress –d archivo.Z`

  `$gzip –d archivo.gz`

- **compress** (LZW) es tecnología de los 80s, **gzip** 90s, **bzip2** 00s, y **xz** 10s.

Comentarios:

- *Compress* es significativamente más viejo (1983) y está basado en el algoritmo de compresión *LZW*.
- *Gzip* fue escrito a principios de los 90 y está basado en el algoritmo *DEFLATE*.
- En general *compress* se ejecuta en menor tiempo y utiliza menos memoria RAM, no obstante *gzip* generalmente obtiene mejores niveles de compresión.
- A principios de los 90 hubo problemas de distribución y uso de *compress* relacionado con temas de patentes de LZW motivados por reclamos de su dueño Unisys (principalmente relacionado con el formato GIF el cual utiliza LZW). Ese fue el detonante y el motivo por el cual los desarrolladores de *gzip* se abocaron a generar un software de compresión de propósito general que no estuviera sujeto a patentes.

7. Medios extraíbles

- UNIX es compatible con herramientas de acceso a medios extraíbles como CD-ROMs, pen drives y HDD externos (USB).

  mount, umount

- **mount** sirve para unir el sistema de archivos que se encuentra en algún dispositivo al árbol de ficheros.
- **umount** permite desmontar dicho dispositivo (recordar hacer esto antes de retirar el dispositivo).
- El archivo /etc/fstab contiene una lista de los dispositivos y los puntos en los que se montan al sistema de archivos principal, para ver que tiene en si PC puede ejecutar en una

shell

```
$cat /etc/fstab
```

8. Procesos

- Un proceso es un programa en ejecución.
- Cada vez que se invoca una utilidad de sistema o un programa de aplicación desde una *shell*, uno o más procesos "child" son creados por la shell en respuesta a la orden.
- Todos los procesos de UNIX se identifican mediante un identificador de proceso único o PID (por sus siglas en inglés).
- Un proceso importante que siempre está presente es el proceso *init*. Este es el primer proceso que se crea cuando se pone en marcha un sistema UNIX y por lo general tiene un PID de 1.
- Todos los demás procesos se dice que son "descendientes" de init.

9. Tuberías

- El operador tubería "**|**" se utiliza para concatenar herramientas del sistema y pasar datos entre sí. Permite crear de forma simple herramientas más complejas.
- Ejemplos:

```
$cat hola.txt | sort | uniq
```

  - Crea tres procesos (*cat*, *sort* y *uniq*) que se ejecutan simultáneamente.
  - A medida que se ejecutan (de IZQUIERDA a DERECHA), la salida del proceso **cat** se pasa al proceso **sort**, que a su vez se pasa al proceso **uniq**. **uniq** muestra su salida en pantalla (una lista ordenada sin líneas duplicadas).

```
$cat hola.txt | grep "perro" | grep -v "gato"
```

  - Encuentra todas las líneas en hola.txt que contienen la cadena "perro" pero no contienen la cadena "gato".

10. Redirección de entrada y salida

- La salida de los programas se suele escribir en la pantalla, mientras que su entrada por lo general viene desde el teclado (si no se dan argumentos de archivos).
- En términos técnicos, se dice que los procesos suelen escribir en la **salida estándar** (la pantalla) y toman su entrada desde la **entrada estándar** (teclado).
- De hecho, hay otro canal de salida, se llama **error estándar,** donde los procesos escriben sus mensajes de error; de forma predeterminada los mensajes de error se envían a la pantalla.
- Para redirigir la salida estándar a un archivo "nuevo" en lugar de hacia la pantalla, se utiliza el operador **>** (o **>>** si se desea agregar la salida a un archivo existente).
- En UNIX los números 0, 1 y 2 se asignan a la entrada estándar, salida estándar y el error estándar, respectivamente.

```
$cat noexiste 2> errors
$cat errors
cat: noexiste: No such file or directory
```

- Se puede redirigir el error estándar y la salida estándar a dos archivos diferentes o al mismo archivo:

```
$find . 1> files  2> errors
$find . 1> output 2>> output
$find . >& output
```

- Se puede combinar la redirección de la entrada con la redirección de la salida, pero cuidado no se debe utilizar el mismo nombre de archivo en ambos lugares:

```
$cat < output > output
```

ya que esto destruye el contenido de la salida del archivo. Esto se debe a que lo primero que hace la shell cuando ve al operador es crear un archivo vacío el cual recibe la salida.

what's the difference between `cat file` and `cat < file`

When `cat` is run with argument files, it will open and read from each of these files to standard output. So with `cat file1.txt`, `cat` opens `file1.txt` and print its contents to standard output:

```
$ cat file1.txt file2.txt
```

When `cat` is run with no argument files, it will read from standard input (e.g., as if you typed directly to cat) and write to standard output. But `< file1.txt` tells the *shell* (e.g., bash) to open the file `file1.txt`, and redirect it to `cat` through standard input:

```
$ cat < file1.txt file2.txt
```

As you can see, the effect is the same, that cat will read the file given; the difference is who opens the file. In the first case, `cat` gets told to open the file itself, while in the second case the shell opens it, and `cat` doesn't know anything except that there's a stream of data coming in through its standard input, that it is supposed to print.

Finally, if you want to use the contents of `file1.txt` as an *argument* to `cat`, you need to use the syntax

```
$ cat $(< file1.txt)
# same as: cat file2.txt [... contents of file2.txt]
```

11. Control de procesos de la Shell

- Las *shells* proporcionan sofisticadas herramientas para controlar los trabajos en ejecución (procesos):

Ctrl-Z, Ctrl-C, fg, bg, &, jobs, ps, kill.

- Por ejemplo, se está editando un archivo de texto y se desea interrumpir su edición. Con el control de tareas, puede suspender el editor, volver a la línea de comandos, y empezar a trabajar en otra cosa. Una vez terminado, puede cambiar de nuevo al editor y continuar como si nunca lo hubiera dejado.

- Los trabajos pueden correr en el **primer plano (foreground)** o en el **fondo (background).**

Sólo puede haber una tarea en el primer plano, el cual tiene el control de la *shell* - que recibe la entrada del teclado y envía la salida a la pantalla.

- Las tareas que corren en el *fondo* no reciben entradas de la terminal, en general se ejecutan en "silencio".
- Los trabajos en el primer plano pueden ser *suspendidos* (**Ctrl-Z**), y o bien reanudados (**fg**) o enviados al segundo plano (**bg**).

- Tenga en cuenta que la *interrupción* de un trabajo (**Ctrl-C**) implica que dicho trabajo es eliminado de forma permanente y no se puede reanudar.
- Desde la línea de comandos se pueden enviar tareas directamente al segundo plano, añadiendo un carácter '**&**' a la línea de comandos.

  `$find / 1>output 2>errors &`

  [1] 27501 ([i] número del proceso enviado al fondo, PID del proceso)
- **jobs** lista todas los tareas asociadas con la *shell* actual
- **ps** muestra los PIDs de los procesos asociados con la shell y las tareas que ejecutan (programas: find, grep,...)
- **kill** envía una señal pidiendo finalizar un proceso (PID), o tarea (%i).

  `$kill -9 PID` ➔ fuerza la interrupción del proceso bruscamente

12. Control de otros procesos

- Se puede usar **ps** para mostrar todos los procesos que se ejecutan en la PC (no sólo los procesos en su *shell* actual)

  `$ps –fae` (ver *man ps*, o *info ps*)

  `$ps –aeH` (jerarquía de procesos completa, incluye el proceso *init*)
- Muchas versiones de UNIX tienen una utilidad de sistema llamada **top** que proporciona una manera interactiva para supervisar la actividad del sistema.
- Teclas *clave* de **top** son:

  s - establecer la frecuencia de actualización k - matar proceso (por PID)
  u - visualización de los procesos de un usuario q - salir
- **pkill** permite matar procesos mediante su nombre en lugar del PID.
- Por razones obvias de seguridad, sólo puede matar a los procesos que pertenecen a usted (a menos que usted es el *root*).

13. Procesamiento avanzado de archivos de texto

- **sed** (stream editor)
- **sed** le permite realizar transformaciones básicas de texto en un flujo de entrada (i.e. un

archivo o entrada de una tubería).

- Por ejemplo, puede eliminar las líneas que contienen un dado texto, o puede sustituir un patrón de texto por otro en un archivo.
- **sed** es un lenguaje de mini-programación y puede ejecutar secuencias enteras de comandos, no obstante su lenguaje es oscuro y probablemente el más olvidado (se basa en un antiguo y *esotérico* editor de línea de UNIX llamado "**ed**").

- **sed** es útil cuando se usa directamente desde la línea de comandos con parámetros simples:

```
$sed "s/pattern1/pattern2/" inputfile > outputfile
```
substitutes pattern2 for pattern1 once per line

```
$sed "s/pattern1/pattern2/g" inputfile > outputfile
```
substitutes pattern2 for pattern1 for every pattern1 per line

```
$sed "/pattern1/d" inputfile > outputfile
```
deletes all lines containing pattern1

14. Páginas de manual

- Para la mayoría de los comandos UNIX existe disponible una mayor información (que "*--help*"), a través de las páginas de manual en línea, con acceso a través del comando **man**.
- La documentación en línea es de hecho dividida en secciones:
  - 1 Instrucciones de nivel de usuario
  - 2 Las llamadas al sistema
  - 3 Funciones de biblioteca
  - 4 Dispositivos y controladores (*drivers*) de dispositivos
  - 5 Formatos de archivo
  - 6 Juegos
  - 7 Material diverso - paquetes de macros, etc.
  - 8 Mantenimiento del sistema y comandos de operación
- **info** es una alternativa interactiva, un poco más amable y práctica. Lamentablemente no está disponible en todos los sistemas.