Laboratorio Sesión 03: Gestión de procesos y scripts

En esta práctica veremos como gestionar los procesos del sistema y como obtener información y personalizar el mismo. Además ampliaremos un poco más la programación de scripts en `bash`.

3.1. Gestión de procesos

En este apartado vamos a ver como gestionar procesos en Linux. El principio es fácil: cuando queremos lanzar una tarea, basta con teclear el nombre, como hemos hecho hasta ahora. Por ejemplo: `yes`. Tecleadlo.

Ahora bien, en algunos casos nos interesa hacer cosas un poquito más complicadas. Por ejemplo podemos querer parar el proceso anterior para poder volver a usar la consola. Lo más fácil es probar con `ctrl + c`. Bien, ya hemos parado el proceso. Sin embargo podemos hacer más cosas con él. Podemos, por ejemplo, pararlo y reanudarlo: teclead `yes` y a continuación teclead `ctrl + z`. Esto suspende el proceso, pero no lo interrumpe. Ahora podéis reanudar el proceso en primer plano (como antes de pararlo) con el comando: `fg` (foreground). También podéis detenerlo (`ctrl + z`) y continuarlo en background: `bg` (background). Hacedlo. Si tenéis varios procesos parados y queréis continuar solo uno de ellos, usad el comando `jobs` para ver que procesos hay pendientes. Podéis poner el número de proceso que os interese después de `fg` o `bg`.

Bien, ahora, si no habéis hecho nada "extra", tenéis un problema, tenéis un proceso en background que no sois capaces de parar y que os está bloqueando la consola. ¿Cómo podéis pararlo? Abrid otra consola. A continuación ejecutad `ps aux`. Os aparecerá una lista con todos los procesos que se están ejecutando junto con su "pid" (IDentificador de Proceso). Si ahora queréis parar el proceso hay que enviarle una señal a este proceso mediante el comando `kill`:

```
kill -9 <pid>
```

Fijaros que en realidad `kill` no mata los procesos, solo les envía una señal. Si esa señal es la 9, es la señal de matar, si es la 15 es la señal de terminar, etc. Otra forma de ver los procesos que estáis ejecutando es mediante el comando `top` (del que se sale con la `q`).

Pregunta 3.1: *¿Qué significan las opciones* `aux` *del comando* `ps`*?*

Pregunta 3.2: *¿Qué otra información muestra* `top` *sobre el estado actual de la máquina además de los procesos que más recursos consumen?*

Cuando enviáis un proceso a ejecución, además, podéis enviarlo directamente en background, añadiendo un `&` al final de la línea de ejecución del proceso. Por ejemplo:

```
yes &
```

... que os habrá vuelto a bloquear la consola. Si queréis evitar el bloqueo y no os interesan los mensajes de salida que os dé el comando podéis escribir la combinación:

```
yes &> /dev/null &
```

Pregunta 3.3: *¿Qué hace exactamente la línea anterior?*

Eso si, ahora tendréis un proceso inútil ocupando recursos... habría que matarlo. Aunque se puede hacer a mano, vamos a ver alguna forma un poco más elaborada (y en este caso claramente basada en la premisa de matar moscas a cañonazos). Para ello podéis probar una línea como la siguiente (no la tecleéis todavía):

```
kill -9 `ps aux | grep "[0-9] yes" | cut -d " " -f 4`
```

Antes, para ver que hace exactamente la línea anterior es una buena idea que probéis a ejecutar simplemente lo que hay dentro de las comillas simples:

```
ps aux | grep "[0-9] yes" | cut -d " " -f 4
```

Si tenéis suerte la salida será un número (concretamente el "PID" del comando `yes`). Si no es así (lo más probable) probad con otro número detrás de la `f` (el resultado estará entre 3 y 15 más o menos, depende de los espacios). Vamos ahora a analizar lo que hace la línea paso a paso:

Pregunta 3.4: *¿Que líneas imprime* `ps aux | grep yes` *sin más parámetros? (probadlo)*

Pregunta 3.5: *¿Para que sirve el* `"[0-9] yes"` *detrás de* `grep`*?*

Pregunta 3.6: *¿Qué hace el comando* `cut`*? ¿Qué número debéis poner detrás de la* `-f` *para que funcione? ¿Por qué puede variar?*

Bueno, es una forma de hacerlo... A continuación, si ya habéis matado el comando `yes` volved a crearlo, vamos a ver otra forma de pegarle otro cañonazo. Probad con la combinación siguiente:

```
kill -9 `ps aux | grep "[0-9] yes" | awk '{print $2}'`
```

Y ojo con los líos de las comillas. Fijaros que esta vez hemos usado el comando `awk` en vez de `cut`. Este comando es mucho más potente que `cut` (y también bastante más complicado).

Pregunta 3.7: *¿Qué hace básicamente el* `awk '{print $2}'` *? Para averiguarlo fácilmente os aconsejo probar la línea:* `ps aux | grep "[0-9] yes"| awk '{print $2}'` *y cambiar los valores que van detrás del $ del 0 al 4 por ejemplo. Eso sí, volviendo a crear un proceso inutil* `yes` *si lo habéis vuelto a matar.*

Si queréis saber muuuucho más sobre `awk` miraros la dirección `http://www.linuxfocus.org/Castellano/September1999/article103.html` y perded un ratito.

3.2. Información sobre el sistema

Vamos a ver ahora una lista de comandos útiles para obtener información sobre el sistema en el que estamos trabajando:

- `who`
- `whoami`
- `uname -a`
- `lshw`
- `dmesg`
- `cat /proc/cmdline`
- `cat /proc/cpuinfo`
- `uptime`
- `last`
- `free`
- `vmstat`
- `df`

- `du`

Me temo que la mejor manera de ver que hacen es ejecutándolos. En caso de duda tenéis `man`, `info` y la web (bueno, y podéis preguntar, claro).

Pregunta 3.8: *¿Qué información muestra cada una de las órdenes anteriores?*

3.3. Algunas otras herramientas útiles

En un entorno gráfico Linux, además de todo lo visto, suele haber otras herramientas interesantes para trabajar con el sistema. Vamos a ver por encima algunas de las más usuales y útiles:

- Herramienta de conexión a internet: `pppconfig` (texto) para módems y `pppoeconf` para configurar redes adsl. También tenéis una GUI que podéis lanzar desde menú (y que suele ser suficiente para casi todo).
- Herramienta para configurar impresoras: en un sistema moderno seguramente tendréis `cups` (es un servidor, la forma más práctica de configurarlo es desde el menú) para configurar las impresoras... es casi inmediato dado que usa un menú muy potente. En un sistema antiguo (cada vez es menos probable que se os de el caso) tendréis que imprimir mediante los comandos `lp` o `lpd` (solo uno de los dos estará instalado en cada sistema). Actualmente estos comandos siguen en el sistema por compatibilidad, pero son redirecciones hacia `cups`. En los ordenadores del laboratorio no hay impresoras pero podéis ver el configurador mediante la opción del menú.

  Pregunta 3.9: *¿Cuántos tipos de impresoras se pueden configurar? ¿Cuáles? Fijaros en que es muy fácil configurar impresoras compartidas.*

- Herramienta para tareas automáticas: `cron` o `anacron` permiten planificar tareas repetitivas cada cierto tiempo. Probad a pedir la ayuda (`man`) de `cron`, `crontab` y también la página 5 de `crontab` y probad a programar una taréa con `crontab -e`. Tened cuidado de que no use una terminal (por ejemplo podéis probar a crear un fichero con `touch`).

  Pregunta 3.10: *¿Qué significan cada uno de los campos del fichero para programar tareas? ¿Qué línea habéis insertado con* `crontab` *para programar una tarea?*

  Pregunta 3.11: *¿Cómo programaríais una copia periódica de datos?*

- Configuración del escritorio: todos los escritorios tienen herramientas que permiten configurarlos en mucho detalle (de hecho hasta cambiarles totalmente la apariencia). Además, los escritorios de Linux suelen ser los mejores (en general, incluyendo los de software privativo) en temas de accesibilidad, aunque a veces es necesario instalarla estas características por separado. En cualquier caso, si alguna vez trabajáis con alguna persona con algún tipo de discapacidad, se suele aconsejar decantarse por escritorios de software libre.

  Pregunta 3.12: *¿Qué herramientas gráficas de configuración de escritorio tenéis en el entorno?*

  Pregunta 3.13: *¿Qué opciones de accesibilidad para discapacitados tenéis por defecto en este escritorio?*

- Además, podréis encontrar todas las herramientas típicas: calculadora, capturadores de pantalla, visores de imágenes, editores de imágenes (`gimp`), de texto (`libreoffice`, `kwrite`), de notas, de contactos, simuladores de ms-dos (`dosbox`) y windows (`wine`), psicoanalistas (podéis buscar en `emacs` si tenéis tiempo), etc. Y si no podéis instalarlos con los gestores de paquetes (que veremos la semana que viene :-).

3.4. Algo más sobre scripts

Finalmente para acabar esta práctica vamos a ver algunos ejemplos más de programación de scripts que mezclan todo lo visto hasta ahora para que os hagáis una idea de su potencialidad. Un posible ejemplo de script es el siguiente:

```
#!/bin/bash
# renombra ficheros cuyo nombre contenga cierta cadena

criterio=$1
sustituto=$2

for i in $( ls *$criterio* );
do
   orig=$i
   dest=$(echo $i | sed -e "s/$criterio/$sustituto/")
   mv $orig $dest
done
```

Como podéis ver este script renombra ficheros que contengan una cierta cadena (criterio, su primer parámetro) sustituyéndola por otra (sustituto, su segundo parámetro). Su llamada será algo así como:

```
./miscript.sh cadenaorigen cadenadestino
```

Y como resultado cambiará el nombre del fichero `ficherocadenaorigen` por `ficherocadenadestino`.

Podéis observar varias cosas en este script. Quzás las más llamativas son el `$(expr)` que es equivalente a las comillas invertidas `` `expr` `` y el comando `sed` que es un editor de textos mediante comandos (podéis mirar su ayuda, pero la "s" significa buscar y reemplazar).

Pregunta 3.14: *Probad el script anterior. ¿Como lo modificaríais para que buscara ficheros con un cierto nombre (parámetro1) y reemplazará una palabra (parámetro2) de su CONTENIDO por otra (parámetro3)? Si os miráis con cariño el comando* `sed` *veréis que se puede hacer con una sola línea ;-)*

Vamos a complicar un poco la cosa. Fijaros ahora en el siguiente script (podéis llamarlo "renombra.sh"):

```
#!/bin/sh

if [ $1 = r ]; then
  shift

# una medida de seguridad
  if [ $# -lt 3 ] ; then
    echo "uso: renom r [expresión] [sustituto] ficheros"
    exit 0
  fi

# elimina el resto de información
  VIEJO=$1 ; NUEVO=$2 ; shift ; shift

# itera todos los ficheros que le hemos especificado
  for fichero in $*
  do
    nuevo=`echo ${fichero} | sed s/${VIEJO}/${NUEVO}/g`
    mv ${fichero} $nuevo
  done
```

```
  exit 0
fi

# si se llega a esta parte es que el uso fue incorrecto
echo "uso:"
echo " renombra r [expresión] [sustituto] ficheros.."
exit 0
```

Este script es más complicado de usar. Para empezar fijaros que tiene un mínimo de 4 parámetros. El primero es siempre una letra `r` (no sirve para nada en especial, sólo para que tengáis un ejemplo de como procesar parámetros). El segundo y tercero són los mismos que los dos primeros en el caso anterior (la cadena a buscar y la cadena a substituir en su lugar en el NOMBRE de los ficheros). Finalmente el cuarto parámetro (y siguientes) es el nombre de los ficheros en los que va a realizar la substitución (es decir, no busca en todos los ficheros del directorio como antes sino que solo reemplaza en la lista que le proporcionamos).

A nivel de programación, la novedad de este último script es que introduce algo de comprobación de errores mediante `if` y `test`. Este último comando es tan habitual que muchas veces dentro de un `if` se substituye `test parámetros` por corchetes `[ parámetros ]` como es el caso en el ejemplo. Es decir, siempre que queráis saber que hacen los corchetes en un script deberéis buscar la ayuda del comando `test`.

Pregunta 3.15: *¿Qué significa entonces* `[ $# -lt 3 ]`*? Si dentro del* `if`*, en lugar de una condición con* `test`*, ponéis el nombre de un programa (o de otro script), ¿de qué depende que se cumpla la condición o no?*

Pregunta 3.16: *¿Cómo escribiríais mediante* `if` *y* `test` *el equivalente a la expresión* `cdparanoia -sB "1-"&& poweroff` *vista en la práctica 2?*

Pregunta 3.17: *¿Cómo modificaríais el script para que solo renombrara ficheros (no directorios)?*

Aunque la siguiente pregunta la podéis contestar usando `bc`, os aconsejo que antes de hacerlo miréis la ayuda del comando `let`.

Pregunta 3.18: *Modificad ahora el script para que al final presente un resumen con la cantidad de ficheros consultados y cuantos de ellos ha modificado. ¿Qué habéis modificado?*

Otra opción útil en los scripts (aunque se usa menos de lo que probablemente se debería es la de crear funciones. La verdad es que las funciones, para variar, son prácticamente intuitivas. Se declaran de la siguiente forma:

```
nombre_de_funcion(){
  comandos_del_shell
}
```

Y se invocan desde otra parte del script simplemente poniendo su nombre. Eso sí, con el pequeño detalle que no se especifican parámetros sino que directamente cuando se invocan los parámetros se listan al lado del nombre de la función y dentro de esta se acceden con los consabidos `$1`, etc.

Pregunta 3.19: *Modificad el primer script de este apartado para dentro del bucle llame a una función (*`cambianombre`*) que sea la que realice el cambio de nombre. Escribid el código resultante.*