Universidad de Murcia

Facultad de Informática

Universidad Politécnica de Cartagena

Escuela Técnica Superior de Ingeniería de Telecomunicación

TÍTULO DE GRADO EN
CIENCIA E INGENIERÍA DE DATOS

Fundamentos de Computadores

Práctica 5: Procesos en el shell de Linux. Tuberías y filtros.

Boletines de prácticas

CURSO 2022 / 23

Índice general

Boletines de prácticas

B5.1. Boletín 1: Procesos en el *shell* de Linux. Tuberías y filtros

B5.1.1. Objetivos

En este boletín continuaremos profundizando en el uso de la línea de comandos de Linux con unos cuantos comandos y funcionalidades adicionales de utilidad. En particular, nos centraremos en los comandos para el lanzamiento y monitorización de procesos, su control mediante envío de señales, el redireccionamiento de sus entradas y salidas y la comunicación entre procesos mediante el uso de tuberías.

B5.1.2. Plan de trabajo

El plan de trabajo de esta sesión será el siguiente:

1. Lectura del boletín por parte del alumno.
2. Seguimiento de ejemplos presentados por el profesor.
3. Realización de los ejercicios propuestos en el boletín, con supervisión del profesor, y completándolos en su caso como trabajo autónomo.

B5.1.3. Ejecución de comandos en primer y segundo plano

Cualquier programa que podamos lanzar desde el menú del entorno gráfico se puede lanzar también desde el terminal. Hasta ahora habíamos estado lanzando, sobre todo, comandos que tienen una respuesta textual inmediata por el mismo terminal. Es decir, el programa especificado en el comando se cargaba en memoria, se ejecutaba, y el proceso correspondiente acababa enseguida, liberando de esa forma el terminal y dejándolo listo para ejecutar más comandos. Sin embargo, también pueden ejecutarse programas desde el terminal cuyos procesos resultantes no tienen por qué terminar inmediatamente. Por ejemplo, el editor de textos `gedit`, el navegador `firefox`, etc. En realidad, el shell permite lanzar varios procesos usando una sola línea de comandos, y ello se puede hacer básicamente de dos maneras:

Lanzamiento en primer plano

`comando1 ; comando2 ; comando3 ...` : Lanza procesos en secuencia, uno después de otro, cada uno de ellos siempre en primer plano. Lanzar un proceso en primer plano significa que, hasta que no acaba, no se lanza el siguiente; y hasta que no acaba el último, no vuelve el *prompt* del `shell`. De esta forma, además, se tiene un cierto control del proceso lanzado desde la propia línea de comandos, pulsando combinaciones de teclas. Así, pulsando `Ctrl-C` en el terminal se puede matar (forzar su finalización) el proceso que en ese momento se encuentra en primer plano, para pasar a ejecutar el siguiente comando, o liberando el terminal si el comando que estaba en ejecución era el último (o el único) de la secuencia. De forma similar, pulsando `Ctrl-Z` se duerme al proceso en primer plano (es decir, se le envía la señal `SIGSTOP` para detener "en seco" su ejecución, pero sin matarlo, de forma que posteriormente podemos continuar su ejecución por donde lo habíamos dejado. Dicho proceso dormido se puede despertar luego con cualquiera de los comandos `fg` (que lo despertaría devolviéndolo de nuevo a primer plano) o `bg` (que lo despertaría, pero pasándolo a segundo plano en este caso; describiremos el lanzamiento en segundo plano justo a continuación).

Lanzamiento en segundo plano

`comando1 & comando2 & comando3 & ...` : Lanza procesos en segundo plano. Lanzar un proceso en segundo plano mediante el separador de órdenes `&` implica que el *shell* no espera a que el proceso lanzado por la orden termine su ejecución para pasar a procesar el siguiente comando (o bien volver a mostrar el *prompt* inmediatamente para aceptar nuevos comandos). Por ejemplo, el comando `firefox &` lanza una ventana con el navegador, y sin necesidad de que éste termine volvemos al *prompt* de forma que pueden seguir tecleándose comandos en el terminal. El `shell` siempre imprime el PID (identificador de proceso, un entero único para cada proceso del sistema) de cada proceso lanzado en segundo plano. El lanzamiento de una orden en segundo plano depende del carácter separador que se utilice para indicar el final de dicha orden (en caso de no especificarlo, implícitamente se asume `;`). Así, por ejemplo, la orden `firefox & gedit` lanzaría a ejecución el popular navegador web en segundo plano e inmediatamente después el editor de texto, este último en primer plano.

B5.1.4. Monitorización y control de procesos

Los comandos más importantes para la monitorización y control de procesos desde la línea de comandos son los siguientes:

- `ps [-Af]` : El comando `ps` muestra un listado de los procesos activos. Usando la primera de las opciones indicadas (`-A`), muestra todos los procesos del sistema (no sólo los lanzados desde ese terminal, que sería lo que ocurriría por defecto, en caso de lanzar el comando `ps` sin opciones). Si se usa la opción `-f`, entonces se presenta un listado de procesos más largo, que muestra distintas informaciones de interés, tales como el PID de cada proceso, su consumo de CPU, el PID de su proceso padre (PPID), la hora de lanzamiento, etc. Es muy habitual usar ambas opciones a la vez, en la forma `ps -Af`. La figura I.1 muestra un ejemplo de la información mostrada por dicho comando.

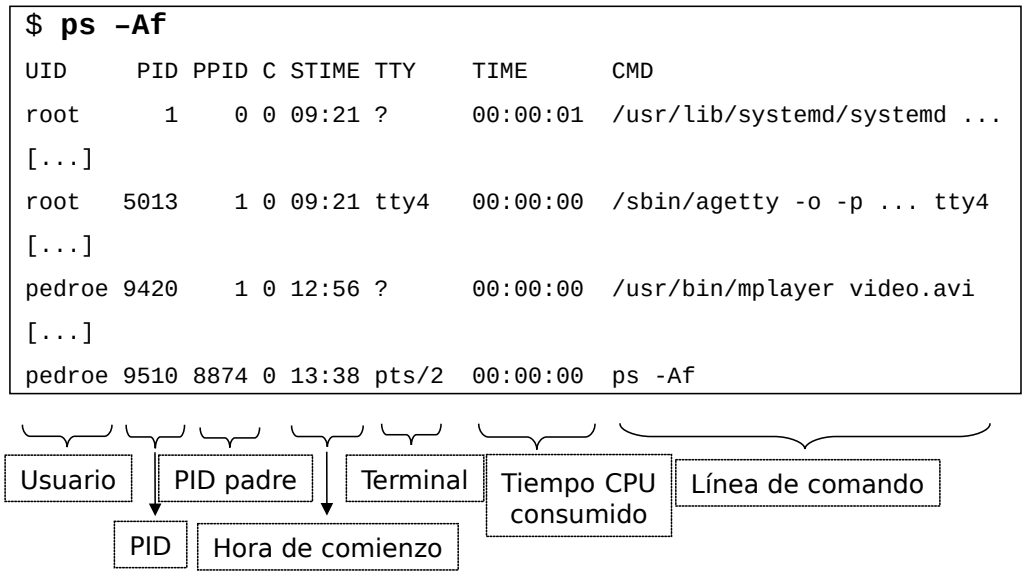

Figura I.1: Ilustración del comando `ps` (ver texto).

- `kill [-9] PID` : Lanza una señal al proceso identificado por su número de PID. Sin parámetro indicando el número de señal, se envía la señal `-15` que termina (mata) al proceso. Esto funciona en general, pero existe un mecanismo por el cual ciertos procesos pueden capturar dicha señal, evitando así su finalización. Si, aún así, se desea matar a uno de dichos procesos, la opción `-9` lo termina forzosamente (se dice que la señal no es "capturable"). Para saber el PID de un proceso determinado puede utilizarse el comando `top`.

- `pgrep PATRÓN`, `pkill PATRÓN` : `pgrep` busca entre los procesos en ejecución aquellos cuyo nombre (dado por el nombre del fichero ejecutable) coincide con el patrón proporcionado (p.ej. una cadena), y devuelve su PID. Esto nos permite localizar de manera más directa el PID del proceso al que queremos enviar una determinada señal. Se puede usar la opción `-u username` para buscar únicamente entre los procesos de un determinado usuario, y la opción `-a` para mostrar además la orden ejecutada por cada proceso coincidente con el patrón. `pkill` busca de igual forma que `pgrep`, pero además lanza una señal a los procesos coincidentes, por defecto la señal `SIGTERM` (15, terminación), o bien la señal indicada mediante la opción `--signal numseñal`.

- `top` : Proporciona una vista dinámica en tiempo real de un sistema en ejecución, mostrando información de resumen del sistema, así como una lista de procesos o hilos que están siendo gestionados por el kernel de Linux, que actualiza cada 3 segundos. Entre otra información, se muestra el tiempo de actividad, tiempo que ha estado el sistema encendido, número de usuarios, el total de tareas y procesos, los cuales pueden estar en diferentes estados, la utilización de la memoria física, etc. top muestra una línea por cada proceso en ejecución, con información agrupada en diferentes columnas, entre las que cabe destacar: el identificador de proceso (PID), el propietario, prioridad (PR), la cantidad de memoria virtual (VIRT) y RAM física (RES) utilizada, su estado (S), porcentaje de CPU utilizado desde la última actualización (CPU), tiempo total de CPU usado desde su inicio (HORA+) y el comando utilizado para iniciarlo. `htop` es una alternativa a `top` con una interfaz textual más amigable, que permite usar el ratón para seleccionar los elementos de la lista de procesos, desplazarse verticalmente para ver la lista completa de procesos y horizontalmente para ver las líneas de comando completas, así como matar más de un proceso a la vez sin insertar sus PIDs, entre otras características. A continuación, se muestra un ejemplo de la salida de `top`:

```
top - 16:40:42 up 2 days, 22:03,  1 user,  load average: 0,18, 0,21, 0,20
Tareas: 257 total,   1 ejecutar,  256 hibernar,    0 detener,    0 zombie
%Cpu(s):  0,4 usuario,  0,4 sist,  0,0 adecuado, 99,2 inact,  0,0 en espera,  0,0 hardw int [...]
MiB Mem :   3788,0 total,    347,1 libre,   1752,7 usado,   1688,2 búfer/caché
MiB Intercambio:   5701,0 total,   4261,4 libre,   1439,6 usado.   1439,5 dispon Mem

    PID USUARIO   PR  NI    VIRT    RES    SHR S  %CPU  %MEM     HORA+ ORDEN
   1653 rtitos    20   0  662652  46764  20240 S   0,3   1,2  23:46.21 Xorg
   1797 rtitos    20   0 4548568 192048  37812 S   0,3   5,0  31:18.77 gnome-shell
  10821 rtitos    20   0 3148760 210820  30984 S   0,3   5,4   6:18.63 dropbox
  23837 rtitos    20   0  843012  36196  16040 S   0,3   0,9   1:39.48 gnome-terminal-
  34914 rtitos    20   0   32,3g  64632  37192 S   0,3   1,7   6:25.16 chrome
  77097 rtitos    20   0   14764   3860   3160 R   0,3   0,1   0:00.08 top
      1 root      20   0  169888   7744   4568 S   0,0   0,2   0:06.37 systemd
```

- `htop`, `glances` : `htop` es una alternativa a `top` con una interfaz textual más amigable, que permite usar el ratón para seleccionar los elementos de la lista de procesos, desplazarse verticalmente para ver la lista completa de procesos y horizontalmente para ver las líneas de comando completas, así como matar más de un proceso a la vez sin insertar sus PIDs, entre otras características. Otra herramienta de monitorización del sistema bastante popular es `glances`, que muestra de un vistazo una gran cantidad de información, no solo referente a los procesos en marcha sino también sobre las interfaces de red, el nivel de ocupación de los sistemas de ficheros, etc., tal y como puede verse a continuación:

```
rtitos-UX310UA - IP 192.168.100.9/24 Pub 103.227.196.194                    Uptime: 2 days, 22:11:59

CPU  [  4.2%]   CPU       4.2%          MEM     63.9%          SWAP     25.1%          LOAD    4-core
MEM  [ 63.9%]   user:     2.6%          total:  3.70G          total:   5.57G          1 min:    0.51
SWAP [ 25.1%]   system:   1.6%          used:   2.36G          used:    1.39G          5 min:    0.38
                idle:    95.7%          free:   1.34G          free:    4.17G          15 min:   0.28

NETWORK       Rx/s   Tx/s   TASKS 260 (825 thr), 1 run, 198 slp, 61 oth sorted automatically
lo             1Kb    1Kb
wlp2s0        30Kb   32Kb   CPU%   MEM%      PID USER       THR  NI S Command
                             4.0    1.2     78335 rtitos      1     0 R /usr/bin/python3 /usr/bin/glanc
```

```
DefaultGateway        23ms   3.7    1.4      1653 rtitos     5     0 S /usr/lib/xorg/Xorg vt2 -display
                             2.3    5.0      1797 rtitos     12    0 S /usr/bin/gnome-shell
FILE SYS       Used  Total   2.0    1.1     23837 rtitos     5     0 S /usr/libexec/gnome-terminal-ser
/ (sda5)     20.0G  79.9G   1.3    5.4     10821 rtitos     93    0 S /home/rtitos/.dropbox-dist/drop
/home          204G   265G   1.3    1.7     34914 rtitos     8     0 S /opt/google/chrome/chrome --typ
                             1.0    9.1     34870 rtitos     24    0 S
                             1.0    0.0       197 root       1     0 S [irq/109-ELAN120]
                             0.7    0.1       793 root       3     0 S /usr/sbin/iio-sensor-proxy
                             0.3    0.3       787 root       3     0 S /usr/sbin/NetworkManager --no-d
                             0.0    6.5      2070 rtitos     5     0 S /snap/snap-store/558/usr/bin/sn
                             0.0    3.5     69102 rtitos     7     0 S evince practica5.pdf
```

B5.1.5. Entrada y salidas de un proceso. Redireccionamiento.

Dado que todos los dispositivos de entrada/salida son tratados como si fueran ficheros, a un proceso no le importa si está leyendo de un teclado o de un fichero ni tampoco si está escribiendo en la pantalla o, de nuevo, en un fichero. De esta forma, cada nuevo proceso creado por el *shell* siempre dispone de tres ficheros abiertos a través de los cuales interactuar con el terminal de texto:

- Una **entrada estándar**, denominada *stdin*, que por defecto es el teclado, y a la que se le asigna el descriptor de fichero número 0.
- Una **salida estándar**, denominada *stdout*, que por defecto es la propia ventana de texto, y a la que se le asigna el descriptor de fichero número 1.
- Una **salida de error**, denominada *stderr*, que también va por defecto a la ventana de texto, y a la que se le asigna el descriptor de fichero número 2.

Un usuario siempre puede redireccionar cualquiera de dichas entradas/salidas desde/a un fichero en el momento de lanzar un comando, mediante los caracteres < (*stdin*), > (*stdout*) y `2>` (*stderr*). Veámoslo con ejemplos:

- `ls -l > fich.txt` : En lugar de escribir la salida *sdout* en pantalla, se escribe en un fichero llamado `fich.txt` (creándolo si no existía, o sobreescribiéndolo si ya existía).
- `ls -l >> fich.txt` : Ídem, pero añadiendo la salida del comando al final del fichero, en lugar de "machacarlo" (en caso de que `fich.txt` ya exista).
- `ls -l noexiste.c *.h 2> err.txt` : La salida de error del comando (*stderr*) se redirecciona al fichero `err.txt`. También se puede usar `2>>`, para añadir al fichero en lugar de "machacar" su contenido.
- `sort < fichero` : El comando `sort` ordena líneas por orden alfabético (o numérico, con `-n`). Si no le proporcionamos ningún fichero como parámetro, este comando, al igual que otros muchos comandos, lee de la entrada estándar (teclado). Sin embargo, con < podemos redirigir *stdin* para que lea de un fichero en vez de lo que se introduzca por teclado. Nótese que en este caso, `fichero` no es un parámetro pasado al comando sino el origen de la redirección.

Como puede intuirse, el número 2 en la forma de redirigir *stderr* (`2>`) proviene del descriptor de fichero correspondiente a la salida de error. De igual forma, es correcto escribir `1>` para redirigir la salida estándar, aunque el número 1 se omite por brevedad. Así pues, podemos redirigir la salida estándar a la de error y viceversa mediante las combinaciones de caracteres `2>&1` (*stderr* a *stdout*) y `1>&2` (*stdout* a *stderr*).

La redirección de la salida de error a un fichero resulta útil en multitud de ocasiones, por ejemplo, cuando queremos capturar todos los mensajes de error y advertencia durante el proceso de compilación de un programa, para después ir resolviéndolos uno a uno. En otras ocasiones, como por ejemplo cuando realizamos búsquedas con `find` en directorios sobre los que no tenemos permisos de acceso, podemos descartar ese (largo) listado de errores haciendo un redireccionamiento de la salida de error al dispositivo nulo `/dev/null`, que no es sino un fichero especial que descarta todo lo que se escribe en él. Un ejemplo de este redireccionamiento podría ser el siguiente:

```
find / -name fichero_que_busco 2> /dev/null
```

B5.1.6. Tuberías y filtros

El enorme potencial que ofrece la línea de órdenes de Linux viene dado por su capacidad de combinar herramientas, conectando la salida que genera un comando con la entrada de otro. El redireccionamiento de la entrada/salida de un comando desde o hacia un fichero es únicamente una pequeña muestra de este potencial; en realidad, una de las claves de que la línea de comandos sea una herramienta tan poderosa es que permite *conectar* una secuencia de comandos entre sí, haciendo que la salida de un comando se convierta directamente en la entrada del siguiente comando, en una sucesión ilimitada de transformaciones, sin necesidad de tener que guardar cada resultado intermedio en un fichero.

Por ejemplo, imagina que queremos visualizar página a página (con `less`) y en orden alfabético (con `sort`) las líneas de un cierto fichero. Con lo que hemos visto hasta ahora, serían necesarios dos comandos: `sort datos > datos_ordenados; less datos_ordenados`. Para evitar tener que *volcar* a un fichero un resultado intermedio tras una determinada transformación sobre los datos de entrada, el shell ofrece una forma directa de redirigir la salida estándar de un comando a la entrada estándar de otro, denominada *tubería* (en inglés, *pipe*). Así, la barra vertical (`|`) tecleada entre dos comandos le indica al shell que queremos utilizar la salida del comando de la izquierda como entrada del comando de la derecha. Siguiendo con el ejemplo anterior, podríamos habernos evitado crear un fichero con los datos ordenados, y en su lugar redirigir la salida de `sort` a la entrada de `less` mediante una tubería: `sort datos | less`.

Filtros

Existe un grupo de comandos cuyo propósito es realizar un procesamiento o transformación muy específico sobre su entrada de manera muy eficiente (p.ej. contar, dividir, sustituir, etc.). A estos comandos se les suele denominar *filtros*, y tienen como principal característica que pueden combinarse mediante tuberías para llevar a cabo un determinada tarea de forma rápida mediante una secuencia de varios comandos entubados. Casi todas las herramientas estándar de Linux pueden trabajar de esta manera: a menos que se les indique lo contrario, leen de la entrada estándar, hacen algo con lo que han leído y escriben en la salida estándar. Esta sencilla idea de usar programas sencillos como *bloques de construcción* es una de las razones por la que Linux ha tenido tanto éxito: En lugar de crear programas enormes que intentan hacer muchas cosas diferentes, se en Linux tenemos un montón de herramientas sencillas, de forma que cada cual hace bien un trabajo muy concreto, y que funcionan bien entre sí. Este modelo de programación se llama *tuberías y filtros*. Entre los filtros más utilizados, además de `grep` (cuyo uso hemos visto en prácticas anteriores), tenemos:

- `wc` : Cuenta el número de líneas (`-l`), palabras, bytes, etc.
- `nl` : Añade el número de línea al inicio.
- `head` y `tail` : Extrae las N primeras/últimas líneas (con la opción `--lines=N`).
- `tr` : Sustituye, agrupa o elimina caracteres.
- `cut` : Muestra partes seleccionadas de cada línea.
- `rev` : Invierte el orden de los caracteres en cada línea.

Veamos el uso de filtros y tuberías con un ejemplo sencillo: Podemos usar `wc -l *.py` para mostrar el número de líneas de cada fichero del directorio actual, cuyo nombre acaba en `.py` (ficheros de código Python). Si quiséramos ordenar el resultado de la cuenta de líneas, podríamos hacerlo con `wc -l *.py | sort -n`, de forma que el fichero con menos líneas aparecería primero. Por último, podríamos estar interesados únicamente en el fichero con menor número de líneas, para lo cual utilizaríamos un filtro adicional: `wc -l *.py | sort -n | head -n 1`, donde el último filtro se queda únicamente con la primera línea producida como salida de la ordenación.

Hay que tener presente que cuando ejecutamos `wc -l *.pdb | sort -n`, el shell crea dos procesos (uno por cada comando en cada extremo de la tubería) para que `wc` y `sort` se ejecuten simultáneamente. De esta forma, la salida estándar de `wc` pasa directamente a la entrada estándar de `sort`, mientras que la salida de `sort` va a la pantalla. Si ejecutamos `wc -l *.pdb | sort -n | head -n 1`, obtenemos tres procesos concurrentes con datos que fluyen desde los archivos, a través de `wc` a `sort`, y desde `sort` a través de `head` a la pantalla. Este detalle es importante ya que el procesamiento llevado a cabo por cada filtro no tiene por qué esperar a que el comando anterior haya terminado de leer y procesar toda su entrada, sino que en función del tipo de filtrado es posible que todos los comandos de la secuencia entubada puedan operar en paralelo sobre el flujo de datos, lo cual hace mucho más eficiente su procesamiento. Nótese que no es este el caso en el ejemplo mostrado, ya que en el caso de una ordenación, no es posible empezar a producir la salida hasta que se han leído todos los datos de entrada.

Subshells

Veamos otra técnica para combinar diferentes comandos en una misma orden. En el ejemplo anterior, estábamos mostrando el número de líneas de los ficheros del directorio de trabajo actual acabados `.py`. ¿Cómo podríamos hacer esta misma operación, para todos los ficheros con ese nombre, pero situados en cualquier subdirectorio que cuelgue del actual? La respuesta es utilizar `find` de forma combinada con `wc`, de forma que los ficheros encontrados en la búsqueda actúen como parámetros del comando de conteo. Una forma de conseguir esto es utilizar un *subshell*: cuando escribimos un comando entre paréntesis `( comando )`, le indicamos al shell que lo ejecute como un proceso hijo con anterioridad al comando "padre" del que forma parte, de forma que el resultado devuelto por el subshell (a través de la salida estándar) pase a ser parte del comando "padre". Veámoslo con un ejemplo: `wc -l $(find . -type f -name "*.py")`. En este caso, el shell ejecutará en primer lugar el subshell que contiene `find`, cuyo resultado es la lista de todos ficheros regulares con el nombre indicado que cuelgan a partir del directorio actual. Dicha lista de ficheros encontrados pasa entonces a convertirse en la lista de parámetros (rutas a ficheros) que se le pasa a `wc`. Otro ejemplo muy habitual del uso de subshells es con `find` y `grep`: el primero encuentra ficheros en función de sus metadatos (nombre, tipo, tamaño, propietario, etc.), mientras que el segundo busca líneas dentro de esos ficheros que coincidan con otro patrón. Por ejemplo, el comando `grep -w "main" $(find .. -name "*.c")` busca la palabra "main" en todos los ficheros cuyo nombre acaba en `.c` situados dentro directorio padre al actual, a cualquier profundidad.

B5.1.7. Variables de entorno. PATH

Cuando interactúas a través una sesión del terminal, hay mucha información que el shell recopila previamente para determinar su comportamiento y acceso a los recursos. La forma en que el shell realiza el seguimiento de todas estas configuraciones es en un área que se denomina *entorno*, que contiene variables que definen las propiedades del sistema. Cada vez que se inicia una sesión de shell, se recopila información de configuración a partir de diversos archivos en el sistema, como por ejemplo, el fichero `~/.bashrc`.

El entorno se implementa como cadenas que representan pares clave-valor. Para mantener una lista con múltiples valores, estos suelen separarse mediante símbolos de dos puntos (`:`). Normalmente, cada par tendrá un aspecto similar a este: `KEY=value1:value2:...`. Las claves son los nombres de las *variables*. Las variables de entorno están definidas para el shell actual y son heredadas por cualquier shell o proceso secundario, y se utilizan para transmitir información a procesos que se producen desde el shell. Por costumbre, estos tipos de variables suelen definirse utilizando letras en mayúsculas. Podemos ver una lista de todas nuestras variables de entorno usando el comando `env`. La lista de variables de entorno es relativamente larga, pero entre ellas cabe destacar algunas como `PATH`, `HOME`, `LANG` o `USERNAME`. Por ejemplo, podemos referirnos genéricamente al directorio de inicio de usuario mediante la variable `$HOME`, independientemente del nombre de inicio de sesión del usuario. Puedes mostrar el valor de cualquier variable con `echo`, anteponiendo el carácter `$` al nombre de la variable (p.ej., `echo $HOME`), mientras que para cambiar su valor (de forma no permanente, sólo para la sesión de shell actual) basta con realizar una asignación mediante el operador =, al estilo de cualquier lenguaje de programación (p.ej. `LANG=en_us.utf-8`). Modificar las variables de entorno puede ser útil cuando quieres anular la configuración por defecto, o cuando necesitas gestionar nuevas configuraciones que tu sistema no tiene por qué crear por sí mismo.

Ejecución de comandos en el *shell*. Concepto de PATH.

Cuando el usuario ejecuta un comando utilizando la ruta completa (ya sea absoluta o relativa) al fichero ejecutable del programa que se quiere ejecutar (p.ej. `/usr/bin/firefox`), el shell simplemente utiliza esa ruta para encontrar el comando. Sin embargo, cuando los usuarios especifican sólo un nombre de comando y no la ruta en que se encuentra (p.ej., `ls`), el shell busca el comando en los directorios en el orden especificado por la variable de entorno `PATH`. Así pues, la variable de entorno `PATH` le dice al shell los directorios en los que debe buscar programas instalados en el sistema, ya sea `ls`, o una aplicación gráfica como Firefox. Por tanto, el correcto establecimiento de la variable `PATH` es vital para el funcionamiento del propio shell.

En pocas palabras, el funcionamiento del shell se basa en un bucle que espera a que se introduzca una orden por teclado para después buscar el comando a ejecutar en el `PATH`. Si se encuentra en uno de los directorios un fichero ejecutable cuyo nombre coincide con el comando tecleado, el shell creará un nuevo proceso mediante la llamada al sistema *fork*, y procederá a ejecutar dicho programa en el proceso hijo.

La llamada al sistema *fork* crea una copia idéntica del proceso que la invoca (el shell), salvo porque *fork* retorna valores distintos en el proceso padre y en el hijo. Si el comando se ha lanzado en primer plano, el proceso padre (el shell) utilizará una llamada al sistema denominada *wait* para esperar hasta que el proceso hijo se haya ejecutado y terminado completamente. Por su parte, el proceso hijo se encargará de ejecutar el comando correspondiente, para lo cual utilizará una llamada al sistema *exec*, a la que le pasará como parámetro la ruta del fichero con el programa ejecutable y sus opciones y argumentos. Al llamar a *exec*, el proceso hijo reemplaza su imagen (que contiene hasta ese momento el código del programa shell, obtenido de `/bin/bash`) por una nueva imagen que contiene el nuevo código del programa a ejecutar (p.ej. en `/usr/bin/ls`).

B5.1.8. Ejercicios a realizar durante la sesión

NOTA IMPORTANTE:
Antes de realizar los ejercicios, asegúrate de grabar la sesión con `script -a typescript_prac5_bol1`. Debes empezar cada ejercicio escribiendo el comentario de documentación `### EJERCICIO N ###` . Finalmente, no olvides añadir dicho fichero a tu repositorio-bitácora, respetando la organización de la misma en directorios y subdirectorios.

1. **Control de procesos en primer y segundo plano.**

   a) Lanza el comando `sleep 10`, y antes de que transcurran 10 segundos, pulsa Ctrl-C para terminar su ejecución. Vuelve a lanzar el mismo comando, esta vez en segundo plano, y tras hacerlo, pulsa de nuevo Ctrl-C. ¿Puedes terminar su ejecución?

   b) Sin teclear ningún comando, presiona repetidamente Ctrl-C. ¿Qué ocurre cada vez que lo haces? ¿Es posible terminar la ejecución del shell de esa forma?

   c) Lanza los programas `gnome-calculator` (calculadora) y `gedit` (editor) desde el terminal, con la única línea de comandos `gnome-calculator ; gedit ;`. Observa cómo, al lanzarse en primer plano, hasta que no termine el primer proceso (cuando cerramos la calculadora) no se lanza el siguiente.

   d) Vuelve a lanzarlos, pero esta vez en segundo plano, tecleando `gnome-calculator & gedit &`. Comprueba que ambos se lanzan y funcionan simultáneamente, y podrían terminarse ahora en cualquier orden.

   e) Con el comando `fg`, pasa a primer plano el último proceso que se lanzó (el editor `gedit`), y, desde el terminal, mátalo pulsando Ctrl-C.

   f) Repite el comando `fg` para hacer lo propio con el comando anterior (`gnome-calculator`), pero esta vez, en lugar de matarlo, páralo pulsando Ctrl-Z desde el terminal. Observa que el proceso `gnome-calculator` temporalmente no responde al ratón ni al teclado (puesto que está durmiendo temporalmente, lo acabamos de parar).

g) Vuelve entonces a despertarlo, pasándolo a segundo plano, con `bg`, y comprueba que su ventana ya responde perfectamente. Finalmente, mata al proceso desde el terminal esta vez usando el comando `kill [PID]`, usando el PID (identificador del proceso) que nos indicó el terminal al lanzar el proceso inicialmente. Puedes usar la orden `ps -Af` para obtener el PID del proceso.

h) Lanza de nuevo los programas `gnome-calculator` y `gedit`, pero esta vez únicamente la calculadora en segundo plano. ¿Qué ocurre si ahora pulsas Ctrl-C en el terminal?

i) Lanza el programa `firefox`. A continuación, vuelve a la terminal donde lo lanzaste y detenlo con Ctrl-Z. Tras comprobar que su ventana del navegador no responde, averigua su PID a partir de su nombre con `pgrep` y finalmente prueba a terminarlo con `pkill`. ¿Puedes finalizar su ejecución enviando la señal de terminación `SIGTERM` (15) (por defecto)? En caso contrario, prueba a enviarle la señal `SIGKILL` (9).

2. **Listado y monitorización de procesos.**

a) Usa el comando `ps -Af` para mostrar todos los procesos que se están ejecutando actualmente en el sistema, y después localiza tres procesos cuyo propietario sea nuestro usuario, y otros tres cuyo propietario sea el superusuario (`root`).

b) Localiza el proceso que ha consumido más tiempo de CPU.

c) Localiza también el último que se lanzó, mirando las horas de inicio (lógicamente, será normalmente nuestro propio `ps`).

d) **ATENCIÓN:** Finaliza la grabación de tu sesión de `script` antes de hacer este apartado (mediante el comando `exit`), ya que la ejecución del comando `top` borra el terminal y con ello la actividad que llevas registrada hasta ahora. Puedes continuar registrando tu actividad tras este apartado ejecutando de nuevo `script -a typescript_prac5_bol1`

Ejecuta `top`, y observa en el mismo la evolución del consumo de CPU de los distintos procesos presentes en el sistema. Después, lanza desde la interfaz de usuario el navegador `firefox`, y trabaja navega por distintas páginas, para observar la evolución dinámica del consumo de memoria y CPU de los procesos involucrados. Puedes dividir la pantalla entre ambas ventanas, terminal y navegador, para observar la salida de `top` mientras navegas.

3. **Redireccionamiento de la E/S**:

a) Lista todos los ficheros contenidos en `/usr/bin/`, con el comando `ls /usr/bin`.

b) Repite el comando redireccionando la salida a un fichero llamado `f1.txt`, usando el símbolo $>$.

c) Comprueba el contenido de dicho fichero con `less f1.txt`.

d) Añade ahora a dicho fichero la fecha actual y los ficheros del directorio `/etc`, con los comandos `date` y `ls`, usando esta vez el símbolo `>>`.

e) Comprueba de nuevo el contenido del fichero, esta vez con un editor de textos (p.e. el `gedit`).

f) Utilizando el comando `ls` con la opción adecuada y la redirección de la salida de error, averigua sobre qué subdirectorios de `/etc` no tienes permiso para acceder.

g) Prueba a pasar como parámetros a `ls` dos rutas, una al directorio `/etc/init.d`, y otra a alguno de los subdirectorios de `/etc` sobre los que no tienes permiso de acceso. Después, repite el comando añadiendo el redireccionamiento necesario para que el posible mensaje de error vaya a parar a un fichero llamado `errores`.

h) Ahora repite el comando cambiando en lo necesario el redireccionamiento para que toda la salida producida vaya a parar al mismo fichero.

i) Usando el comando `find` y el redireccionamiento, haz que se muestre por pantalla únicamente el nombre de los subdirectorios de `/etc` sobre los que no tienes permiso de acceso, pero sin que vayan acompañados de ningún mensaje de error. Ayuda: `find` tiene la opción `-readable`, y recuerda que puedes usar los operadores lógicos para buscar.

j) Genera otro fichero `f2.txt` a partir del anterior `f1.txt`, con las líneas ordenadas alfabéticamente, redireccionando tanto la entrada como la salida de `sort`.

4. **Tuberías y subshells.**

a) Combinando los comandos `cat` y `wc -l` mediante una tubería `|`, contar el número de líneas del fichero `f2.txt` generado anteriormente.

b) Añade un comando `grep` a la tubería anterior para que sólo se cuenten líneas que contengan la letra "z".

c) Combinar mediante tuberías los comandos `cat` y `cut -c` para mostrar por pantalla los contenidos del fichero `f2.txt` donde se ha eliminado el primer carácter de cada línea.

d) Añadir a la tubería así generada el comando `sort -r` para ordenar el resultado en orden alfabético inverso.

e) Añadiendo ahora a la tubería los comandos `head -[n]` y `tail -[n]`, conseguir que se muestren en pantalla sólo las líneas que van de las filas 5 a la 15.

f) Finalmente, y añadiendo el comando `tr` adecuado (ver manual), pasar toda la salida de minúsculas a mayúsculas.

g) Con la opción `-printf "%s;%p;%u;%M\n"`, utiliza `find` para generar un fichero `progs.csv` que contenga el nombre, tamaño, usuario y permisos de todos los ficheros del directorio `/usr/bin`, separados por punto y coma. Utiliza `less` para comprobar que el fichero tiene el contenido deseado.

h) Utilizando los datos del fichero `progs.csv` como entrada, utiliza `cut` para seleccionar únicamente el campo correspondiente al tamaño. La opción `-d` de `cut` te permite especificar el delimitador a usar, mientras que con `-f` puedes indicar los campos a seleccionar (haz `man cut` para más detalles). Recuerda que el punto y coma tiene un significado especial para el shell y por tanto debes entrecomillarlo.

i) Modificando el comando anterior, selecciona los campos tamaño y nombre del fichero `progs.csv`, y con una tubería adicional muestra el resultado ordenado por tamaño. Añade una nueva tubería con `tail` para mostrar únicamente el nombre y tamaño del fichero de mayor tamaño.

j) Utiliza el resultado devuelto por `find` para, usando un subshell de forma adecuada, mostrar con `grep` y `sort` una lista ordenada de todos aquellos ficheros regulares en `/usr/bin` que contienen la cadena `#!/bin/sh`. Recuerda entrecomillar con comillas simples (`'`) el patrón a buscar, ya que contiene caracteres especiales para el shell (p.ej. `#`).

k) A partir del comando anterior, utiliza un nuevo subshell para mostrar un listado largo de dichos ficheros (los que están en `/usr/bin` y contienen la citada cadena `#!/bin/sh`) mostrando su tamaño de forma más legible *para humanos* (busca la opción con `man ls`).

l) Añade al comando anterior una tubería para quedarte únicamente con las líneas que contengan dos dígitos seguidos de la letra "K" (tamaño superior a 10K).

m) Añade al comando anterior una tubería con `tr` para sustituir todos los caracteres espacio que aparezcan repetidos por uno sólo (busca la opción con `man tr`).

n) Añade al comando anterior las tuberías necesarias para seleccionar con `cut` el nombre de fichero, utilizando `rev` para evitar tener que contar manualmente la posición de dicho campo.

5. **Variables de entorno. Concepto de PATH.**

a) La variable de entorno `LANG` define, junto con otras, la configuración regional y el idioma del shell. Empieza mostrando su valor actual con `echo`, y luego prueba a ejecutar el comando `ls` pasándole una ruta inexistente para generar un mensaje de error. z

b) Repite el comando anterior anteponiendo `LANG=valor` para establecer otro valor distinto (no importa cuál): `LANG=fr ls noexiste` ¿Cuál es el resulado? Por último, ejecuta `locale -a` para ver las configuraciones regionales y de idioma soportadas en el sistema, y vuelve a establecer `LANG=es_ES.UTF-8`.

c) Muestra el valor de `PWD` (variable de entorno que guardar el directorio de trabajo actual, *present working directory*) y después establece dicha variable con otra ruta. ¿Ves algún cambio en el prompt? Ahora prueba a ejecutar `ls`. ¿Realmente has conseguido cambiar de directorio con la asignación?

d) La variable de entorno `OLDPWD` mantiene el directorio de trabajo anterior al actual, es decir, aquél en el que estábamos antes del último `cd`. Sitúate primero en tu directorio de inicio (`cd`) y después cambia a `/etc`. Muestra el valor de `OLDPWD` y luego prueba a asignarle otra ruta válida (p.ej., el directorio raíz). Finalmente, ejecuta el comando `cd -` y comprueba a qué directorio te has movido.

e) Crea un subdirectorio llamado `bin` en tu directorio de inicio de usuario, y copia el fichero `/usr/bin/ls` a dicho subdirectorio, con el nombre `myls`. Nótese que este fichero contiene el programa del archiconocido comando usado para listar ficheros y directorios.

f) Cámbiate al directorio `bin` recién creado, y comprueba que el comando sigue funcionando tecleando `./myls -l`, por ejemplo. Observa la necesidad de proporcionar en el comando la ruta hasta el fichero a ejecutar, por no estar nuestro directorio de trabajo actual en el `PATH`.

g) A continuación, usa el comando `chmod` sobre el fichero copiado para eliminar su permiso de ejecución, y vuelve a intentar la ejecución del comando.

h) Vuelve a añadir el permiso de ejecución y comprueba que ahora sí puedes ejecutarlo.

i) Modifica el valor de `PATH` para añadir el subdirectorio creado anteriormente: `PATH=$PATH:$HOME/bin`. Nótese como es posible utilizar el valor de cualquier variable de entorno al asignar un valor a una variable. Ahora, trata de ejecutar `myls` desde el directorio actual y repite dicho comando tras cambiarte a cualquier otro directorio.