

INFORMÁTICA

Anexo a la Práctica 1. Procesos.

AMPLIACIÓN DE LINUX.

Este anexo es una extensión de la práctica 1, y desarrolla un poco más algunos conceptos básicos de Linux. No pretende ser una guía exhaustiva, sino simplemente exponer algunas cuestiones adicionales que, no siendo necesarias para la práctica 1, pueden resultar de interés para el alumno. Está pensado como continuación de dicha práctica, por lo que se supone que se conoce todo lo expuesto allí.

VARIABLES DE ENTORNO

Con el objeto (entre otras cosas) de simplificar el uso del interfaz de comandos existen las llamadas variables de entorno. Estas variables contienen información acerca de modos de funcionamiento por omisión o acerca de lugares en el sistema de archivos donde encontrar cosas. Una de las más importantes es la llamada PATH. El PATH (se podría traducir como *ruta*) no es más que una lista de directorios del sistema operativo donde el intérprete de comandos busca las aplicaciones que se le pide ejecutar. De esta manera, si el path tiene los directorios `/bin` y `/home/larq/ejecutables` (es decir, `PATH=/bin:/home/larq/ejecutables`), entonces cuando en un terminal se teclee:

```
bash-3.2$copiar archivo1 archivo2
```

donde `copiar` es una aplicación que hemos creado nosotros, el intérprete de comandos la buscará en todos los directorios que figuran la lista que forma la variable PATH. En primer lugar buscará el primer directorio incluido en el PATH, es decir `/bin`. Y si no está ahí, entonces la buscará en el directorio `/home/larq/ejecutables`. En caso de que no lo encuentre, avisará de que no lo ha encontrado y se pondrá a esperar un nuevo comando. La búsqueda se hace siguiendo el orden de aparición en la cadena, por lo que si dos programas se llaman igual (y por lo tanto residen en directorios diferentes), se ejecutará únicamente aquel que se halle en el directorio que se encuentre antes en la variable PATH.

La variable de estado PATH es muy útil cuando se quieren disponer las aplicaciones del sistema de forma ordenada (es decir, que estén en lugares específicos para ello) en varios lugares distintos y no se quiere estar cambiando continuamente de directorio actual.

En general se pueden definir variables de estado para lo que sea necesario. Existen algunas que crea el propio sistema, como la variable PATH vista ya. El valor de estas (y de cualesquiera otras) variables de entorno se puede ver con el comando `echo`:

```
bash-3.2$echo $PATH
```

Si la variable tiene el valor visto anteriormente, la salida de este comando será:

```
/bin:/home/larq/ejecutables
```

Además de visualizarlas, es posible modificarlas. Por ejemplo, cuando uno está desarrollando código (el caso de este laboratorio), puede ser conveniente que al invocar un comando se busque inicialmente en el directorio actual. Para ello se puede usar el siguiente comando:

```
bash-3.2$export PATH=.:$PATH
```

Este comando lo que haría sería establecer (`export`) una variable de entorno llamada PATH, con el valor resultante de concatenar `.` y el valor anterior de la variable PATH (`$PATH`). El punto representa el directorio actual, sea el que sea. El comando `export` se puede utilizar para establecer variables de cualquier tipo. Si la cadena que va a representar su valor contiene espacios en blanco, entonces es preciso incluirla entre comillas:

```
bash-3.2$export VARIABLE="Una cadena de caracteres"
```

PROCESOS

Un sistema operativo multitarea como Linux es capaz de mantener varios procesos en ejecución de forma simultánea. Para comprender cómo funciona a este respecto conviene comprender qué es un proceso. Un proceso es una instancia de un programa en ejecución. Es preciso distinguir un proceso de un programa porque en Linux es posible ejecutar varias instancias de un mismo programa de forma simultánea. Un programa es un paquete de software que reside en el disco duro, por ejemplo, y que al ser invocado es cargado en la memoria del sistema y puesto en ejecución. Ese código en ejecución, con sus datos y variables de entorno, es un proceso. Al arrancar un terminal, por ejemplo, el código que lo forma se trae de disco duro a la memoria del sistema y es puesto en ejecución. Se crea un proceso “terminal”. A continuación, se pueden ejecutar algunos comandos en él. El proceso está en ejecución. En ese momento, es posible arrancar un nuevo terminal diferente al anterior en el que ejecutar otros comandos. Para ello, el sistema operativo cargará de nuevo el código del programa terminal y lo pondrá en ejecución, de forma independiente a lo hecho en la anterior invocación. Eso será otro proceso, diferente del anterior, (aunque el código que se ejecuta sea el mismo). En esta situación, con ambos terminales corriendo, cada “ventana” de terminal es una instancia diferente del mismo programa, el programa “terminal”. Cada “ventana” es un proceso.

La distinción entre programa y proceso es muy interesante, ya que desliga completamente unas ejecuciones de un programa de otras, e incluso, como se ha visto, permite realizarlas simultáneamente sin que haya confusión entre ellas. Un proceso tiene, entre otras cosas, su propio estado y datos, con lo que, si se ejecutan varias instancias del mismo programa, por ejemplo, los terminales del ejemplo anterior, cada uno mantiene de forma individual características como la historia de comandos, las variables de entorno, etc. Es posible incluso que uno de los terminales se quede bloqueado (“colgado”) y eso no impedirá al otro seguir funcionando.

Existe un tipo especial de proceso denominado ‘*job*’. Un ‘*job*’ es un proceso que se ha lanzado desde un intérprete de comandos. Estos procesos (los ‘*jobs*’) son un poco más fáciles de controlar que los demás, pero mantienen en común con ellos todas las características importantes.

A lo largo de su ejecución, un proceso se mueve entre varios estados. El más fácil de comprender es el estado “en ejecución”. En este estado, el proceso está haciendo su trabajo de forma activa, es decir, está “corriendo”. Si en algún momento el proceso debe esperar por alguna razón (por ejemplo, porque está esperando una respuesta), el proceso puede pasar a estar “suspendido”. Cuando se cumple la condición que estaba esperando (por ejemplo, cuando recibe esa respuesta), está de nuevo listo para ejecutarse, por lo que pasa al estado de “listo”. Un proceso “listo” no tiene por qué estar “en ejecución”, ya que el procesador sólo puede ejecutar un proceso en cada momento, y puede haber (de hecho, es lo más frecuente) más de un proceso “listo” para ejecutarse. La parte del sistema operativo que decide qué proceso “listo” se debe ejecutar en cada momento es el planificador (‘*scheduler*’). De hecho, el planificador también decide cuándo un proceso “en ejecución” debe dejar de ejecutarse para permitir que otro del grupo de “listos” se ejecute un rato. Existe un estado parecido al “suspendido” que es el “parado”. Los procesos “parados” están pendientes de que se les permita continuar su ejecución por medio de una “señal”. Las señales son métodos de comunicación entre procesos y se explican más adelante, pero como avance, y a modo de ejemplo, por medio de señales, un usuario puede terminar la ejecución de un proceso que está corriendo o, como se acaba de indicar, detener su ejecución.

El último estado que vamos a considerar se denomina “zombie” y es algo más complejo. Tiene que ver con el hecho de que todos los procesos deben ser “lanzados” por otro proceso. En la primera parte de esta práctica se explicó cómo el interfaz de comandos, al recibir del usuario un comando por medio del terminal (por ejemplo, el comando `ls`), lanzaba ese comando a ejecución. Lo que hacía era crear un nuevo proceso distinto de sí mismo. El nuevo proceso consistía en el código que constituye el comando `ls`. En esta situación, se dice que el interfaz de comandos es el proceso padre del comando `ls`, y éste, a su vez, es un proceso hijo del proceso interfaz de comandos. Al constituirse en proceso hijo, el comando `ls` pasa a tener entidad de proceso por sí mismo, con todas las características que esto conlleva (memoria, datos, variables de entorno, etc). Se convierte en un proceso con todas las consecuencias. Todos los procesos del sistema forman, de esta manera, una jerarquía, una estructura de tipo árbol, en la que de cada proceso padre “cuelgan” sus procesos hijos. Todos los procesos del sistema están conectados de esta manera.

Lo habitual es que cuando un proceso termina, lo notifique a su proceso padre para que el sistema operativo libere los recursos que tenía asignados a ese proceso hijo. Si es el padre el que termina antes, el proceso padre, antes de terminar su ejecución (cosa que puede pasar mucho tiempo después de haber lanzado al proceso hijo; de hecho puede haber lanzado muchos hijos más) espere a que termine el proceso hijo (todos ellos, si tiene varios). Es decir, realmente no termina hasta que han terminado todos sus hijos. Sin embargo, puede ocurrir que, por alguna razón, el proceso padre (el interfaz de comandos) no reciba la notificación de uno o varios de sus procesos hijos (por un mal funcionamiento del sistema). En este momento, los hijos dejan de tener conexión con la jerarquía de procesos del sistema. El sistema operativo no puede eliminarlos de la lista de procesos ni liberar sus recursos. Los procesos que se encuentran en esa situación se denominan procesos zombies, y, aunque no están “en ejecución” porque ya han terminado, ni están “suspendidos” ni “parados”, siguen existiendo.

Entrada y salida.

Se llama entrada estándar al lugar de donde un proceso toma los datos sobre los que trabaja si no se le indica otra cosa. Análogamente, se denomina salida estándar al lugar al que vuelca sus resultados si no se le indica ninguno específicamente. La salida y la entrada estándar suelen ser el terminal (la pantalla si es salida y el teclado si es entrada). Por ejemplo, el comando `ls` vuelca el listado en la salida estándar, es decir, la pantalla. El comando `cat` sirve para concatenar archivos, y, si no se le especifica qué archivos concatenar, toma lo que se introduzca por teclado.

Todos los procesos, incluidos los correspondientes a los que creemos más adelante en la asignatura, tienen definidas su entrada y su salida estándar. Éstas, como se ha dicho más arriba, por omisión son el teclado y la pantalla. Sin embargo, es posible especificar (redirigir) fuentes o destinos alternativos. Por ejemplo, el comando `ls` vuelca el contenido de un directorio a la pantalla. Si se quiere guardar ese listado en otro archivo, es posible redirigir la salida del comando a ese archivo especificándolo al invocar el comando:

```
bash-ln.05$ ls > listado.
```

En este caso, “listado” es el archivo en el que se va a guardar el listado, y el símbolo “>” especifica la redirección de la salida. En el caso de la entrada, el símbolo para la redirección es “<”. Generalmente los archivos a los que se redirige la salida se prefieren crear vacíos, por lo que el procedimiento es borrarlo si existe. Sin embargo, puede ser conveniente no crearlo nuevo, sino añadir la salida al contenido previo del archivo. En este caso se sustituye el símbolo “>” por “>>”.

Existe un segundo destino de salida para los procesos denominado “error estándar”. Es donde los procesos vuelcan los mensajes de error, y es distinto de la salida estándar ya que puede ser apropiado no mezclar el resultado de la ejecución del proceso con posibles avisos que deba dar. El error estándar también es el terminal, pero puede redirigirse de manera independiente a lo que se haga con la salida estándar. Para no confundir ambas salidas, el símbolo para redirigir el error estándar es “2>”. Así,

```
bash-ln.05$ ls 2> error
```

pondría en el archivo “error” los posibles mensajes de error que pueda generar, pero no así el resultado de la ejecución del comando, es decir, el listado del directorio actual. De hecho, 2 es el “descriptor” del error estándar, y por ello se puede redirigir así. La entrada y la salida estándar también tienen su descriptor, en concreto 0 y 1 respectivamente. De forma que cuando se redirige la salida estándar como en el comando:

```
bash-ln.05$ ls > listado.
```

En realidad, lo que se está haciendo es:

```
bash-ln.05$ ls 1> listado.
```

Lo que ocurre es que, como esto se hace con mucha más frecuencia que redirigir el error estándar, se sobreentiende que, si no se dice nada, se redirige la salida estándar.

Concatenación de procesos (“pipes”).

En algunas ocasiones se quiere ejecutar una sucesión de comandos en el interfaz de comandos de forma que la salida de uno sea tomada como entrada por el siguiente. En estos casos es posible encauzar los procesos para que se ejecuten uno a continuación de otro y se proceda con las entradas y salidas como se ha explicado. Para ello se crea una estructura denominada “pipe” (cauce):

```
bash-ln.05$ ls -la | grep "nada"
```

La conexión entre el comando `ls` y el comando `grep` está marcada por el cauce (“|”). El listado que genera el comando `ls`, si éste figurara de manera independiente, se mostraría en la salida estándar (la pantalla). Por su parte, el comando `grep` (que sirve para localizar patrones; se explica más adelante, pero conviene consultar en este punto la página del manual correspondiente) trabajaría sobre la entrada estándar (el teclado). Al conectarlos con un cauce, la salida estándar de `ls` se convierte en la entrada estándar de `grep`, por lo que `grep` busca la palabra “nada” en el listado generado por `ls`. Es posible concatenar tantos comandos como sea necesario de esta manera.

Ejecución en segundo plano.

En un interfaz de comandos, lo habitual es que el usuario teclee el comando que quiere ejecutar y este se ejecute a continuación, interactuando con el usuario de la manera que tenga programada. Esta es la ejecución en “primer plano”. Sin embargo, para poder ejecutar más de un comando de forma simultánea (no en secuencia, como con un cauce, sino a la vez), es preciso que todos menos uno puedan ejecutarse en lo que se llama “segundo plano”, es decir, sin interactuar con el usuario (al menos a través del terminal). Por ejemplo, si uno de los comandos que se quiere ejecutar es un listado de todos los archivos y directorios del sistema (`ls -Ra`), cosa que seguramente tarde bastante tiempo, mientras se genera el listado se quiere averiguar qué otros usuarios están conectados al sistema (con el comando `who`; mirar la página del manual correspondiente), lo que se puede hacer es generar el listado en segundo plano, de forma que, mientras se está generando, el interfaz de comandos queda libre para seguir atendiendo al usuario. Para hacer esto:

```
bash-ln.05$ ls / -Ra >ListadoCompleto&
```

El símbolo “&” al final del comando le indica al interfaz de usuario que ejecute el comando en segundo plano, con lo que el interfaz de comandos lo lanza y vuelve a atender al usuario. Cuando un proceso se ejecuta en segundo plano, no debería acceder ni al teclado ni a la pantalla, es decir, no interactuar con el usuario, ya que, en ese caso, por ejemplo, se mezclarían los mensajes de ese proceso con los del que quede en primer plano. Sólo deberían ser candidatos a procesos ejecutados en segundo plano aquellos que no necesiten interactuar con el usuario para su ejecución. Si un proceso se va a ejecutar en segundo plano, se debe redirigir su salida antes de lanzarlo, por ejemplo así:

```
bash-ln.05$ ls > /dev/null&
```

El comando anterior genera, en segundo plano (&) un listado del directorio activo (`ls`), pero lo vuelca en el archivo `/dev/null`, que es un archivo especial que no retiene información, pero que en una escritura da como resultado una operación completada con éxito. De esta manera no se muestra el listado en pantalla, con lo cual el proceso no interfiere en la salida con lo que envíe el proceso que se ejecute en primer plano.

Teniendo en cuenta la posibilidad de ejecutar comandos en segundo plano, un usuario puede tener una serie de procesos lanzados y ejecutándose en paralelo. Estos procesos serían los ‘jobs’ del terminal desde donde se lanzan. Es posible ver cuáles son con el comando `jobs` (consultar su página de manual para más información de opciones y formas de funcionar). Cada ‘job’ tiene un identificador (diferente del identificador de proceso, PID, que se verá más adelante), y se puede usar para, entre otras cosas, traerlo a primer plano, o llevarlo a segundo plano. Para hacer esto existen los comandos `bg` y `fg` (background y foreground). El comando `bg` lleva a segundo plano el job cuyo identificador se especifica. El comando `fg` hace justo lo contrario, traer a primer plano un ‘job’ que se había enviado a segundo plano.

Comando de búsqueda (grep).

Cuando se trabaja con grandes cantidades de información (como listados, por ejemplo), a veces es conveniente filtrarla para quedarse sólo con aquello que es interesante para la tarea que se está llevando a cabo. Por ejemplo, el archivo `/etc/passwd` contiene un listado de usuarios del sistema y, para cada uno, muestra algunas características. Una de ellas es el intérprete de comandos que usa por omisión, ya que en Linux existen varios que se pueden usar. El comando siguiente:

```
bash-ln.05$grep bash /etc/passwd
```

buscará entre las líneas de `/etc/passwd` cuáles contienen la cadena “`bash`”, que es uno de los intérpretes de comandos que se pueden usar, y las mostrará (la línea entera). De esta manera es posible saber todos los usuarios que utilizan `bash` por omisión. `Grep` es un comando muy versátil que permite realizar búsquedas más complejas que un simple patrón. Para este tipo de búsquedas complejas se utilizan las llamadas “expresiones regulares”, que son métodos formales para especificar lo que se tiene que buscar. Por ejemplo, se pueden buscar líneas que no contengan un determinado patrón, líneas en las que el patrón se encuentre al principio de la línea, al final, etc. Es conveniente dedicar un tiempo a leer la página del manual relativa a `grep` para conocer algunas de sus opciones de trabajo y cómo se crean las expresiones regulares, ya que es una herramienta muy útil.

Gestión de procesos.

En Linux el trabajo que realiza un usuario se organiza en procesos. Cada comando (prácticamente cada acción) que lanza un usuario es un proceso, por lo que es importante conocer las herramientas de las que dispone Linux para gestionar el funcionamiento de los procesos.

La primera de estas herramientas es el comando ‘`ps`’. ‘`Ps`’, por omisión, proporciona información de los procesos de terminal que tiene en marcha un usuario. Un proceso se dice “de terminal” cuando ha sido lanzado desde un terminal. Sin embargo, por medio de opciones o ‘flags’, se le puede indicar que muestre otro tipo de procesos, o incluso, procesos de otros usuarios. La información que puede mostrar el comando `ps` es mucha, y la más importante es: el ID de proceso (el PID), el nombre del mismo, el ID del proceso padre (el PPID) y el ID del usuario que lo ha lanzado. Por ejemplo, el comando:

```
bash-ln.05$ps -la
```

generará un listado completo (‘`l`’) de procesos de terminal de cualquier usuario (opción ‘`a`’) con las siguientes columnas:

```
UID    PID    PPID   F   CPU PRI  NI  SZ  RSS  WCHAN  S  ADDR TTY  TIME CMD
```

El listado incluirá a continuación una línea para cada proceso encontrado. De todas las columnas, las más interesantes son: UID (ID del usuario que ejecuta el comando), PID (ID del proceso), PPID (ID del proceso padre correspondiente), y CMD (el comando que ha generado el proceso).

Un listado es completo cuando tiene toda la información pertinente, es decir, todas las columnas. Un listado normal tiene menos columnas. Comparar el resultado del comando anterior con el de este otro:

```
bash-ln.05$ps -a
```

Otro comando relacionado con la obtención de información de procesos es ‘`top`’. Este comando proporciona diversos datos sobre los procesos que están corriendo en un sistema, mostrando una línea por cada proceso, y ordena las líneas según se le pida. Por ejemplo, se puede ordenar por PID, por porcentaje de uso de la CPU, por estado del proceso, etc. Una característica interesante de este comando es que actualiza la información de forma dinámica mientras no se termine, por lo que la información presentada es siempre la más reciente.

También relacionado con la información de los procesos del sistema es el comando ‘`pstree`’, que muestra el árbol jerárquico de los procesos del sistema, es decir, indicando qué procesos hijo tiene cada padre.

SEÑALES

El comando `ps` o el comando `top` son muy útiles, entre otras cosas, para obtener el PID de un proceso, ya que los procesos se gestionan a partir de este identificador. Por ejemplo, si un proceso se “cuelga”, es decir, deja de responder, se puede terminar enviándole una señal de terminación. Las señales son un mecanismo de comunicación entre procesos que se puede utilizar en un caso así. Los procesos están preparados para recibir y procesar señales que les sean enviadas (esta funcionalidad hay que incluirla en los programas a la hora de escribirlos). Cada señal está representada por un entero. Las más usadas son `SIGINT` (número 2), que es el equivalente a hacer un `Ctrl-C`, es decir, acabar el programa, `SIGTERM` (número 15), que indica al programa que termine, pero le da la oportunidad de hacerlo de manera ordenada, o `KILL` (número 9), que lo aborta sin informarle (útil cuando el proceso no responde). El comando para enviar señales a un proceso es `kill`:

```
bash-ln.05$ kill -15 21345
```

El comando anterior avisará al proceso con PID 21345 de que debe terminar. Si el proceso está correctamente programado guardará sus datos y terminará. En caso de que esté colgado, puede ser preciso ejecutar:

```
bash-ln.05$ kill -9 21345
```

que lo terminará sin darle opción a hacerlo de forma ordenada. En los comandos anteriores, en lugar del número de la señal, es posible indicar su nombre. Por ejemplo, la primera de las invocaciones a `kill` mostradas (la que usa la señal `-15`) se puede sustituir por:

```
bash-ln.05$ kill -TERM 21345
```

ya que 15 es el número que corresponde a `SIGTERM`.

Como se ha indicado antes, las señales son un mecanismo de comunicación entre procesos. Para usarlo, los programas deben crearse con la funcionalidad necesaria, es decir, deben contener las rutinas que tratan las señales. En caso contrario, no se darán cuenta de que se les envían. Sólo una de las señales, la señal `SIGKILL` (9) no puede ser tratada por el programa, ya que esta señal termina el proceso de forma fulminante.

EJERCICIOS

Para cada ejercicio de los que figuran a continuación (y para todos los de la asignatura, en general), es conveniente consultar la página de manual correspondiente (por medio del comando `man`).

1. Usar el comando `cat` para volcar información obtenida del teclado en un archivo. Para ello será necesario redirigir la salida al archivo. Para terminar la entrada de teclado, teclear `Ctrl-D`.
2. Repetir el proceso anterior con el mismo archivo, de forma que lo que se introduzca no borre lo que ya existía (lo que quedó en el archivo en el ejercicio 1).
3. Repetir el ejercicio 1 con un archivo diferente.
4. Volcar los dos archivos generados a la pantalla (con un solo comando).
5. Generar un listado de un directorio de forma que solamente se muestren los directorios (será necesario unir con un cauce los comandos `ls` (para generar el listado) y `grep` (para procesarlo). Consultar las páginas del manual de ambos comandos.
6. Repetir el ejercicio anterior, pero mostrar únicamente los archivos propiedad del usuario estándar de los equipos del laboratorio (`larq`).
7. Repetir el ejercicio anterior, pero listar únicamente aquellos archivos que se han creado en el mes de enero.
8. Realizar las siguientes tareas:
 - a. Crear un proceso que vuelque datos de teclado al archivo `/dev/null` con el comando `cat`;
 - b. Obtener su PID (puede ser necesario abrir un segundo terminal).
 - c. Pararlo con la señal adecuada (`SIGSTOP`).
 - d. Comprobar el estado del proceso.
 - e. Obtener el árbol de procesos del sistema. Localizar en el árbol el proceso.
 - f. Hacer que acabe ordenadamente.
9. Obtener un listado de todos los procesos del sistema que han sido iniciados por el supervisor del sistema (`root`).