Práctica 4 - Programación de comandos

1. Introducción

En prácticas anteriores, entre otras cosas, se utilizaron varios comandos básicos de UNIX como `ls`, `cd`, `grep`, y `chmod`. A lo largo de esta práctica se explicará qué son realmente los comandos y se pedirá implementar en Java unas versiones básicas de algunos de ellos. El objetivo es que se comprenda cómo los sistemas UNIX permiten el uso de los distintos comandos habituales, que se vea que es relativamente sencillo crear comandos propios con el funcionamiento que se requiera y practicar el manejo del compilador de Java y la programación con este lenguaje.

Esta práctica puede hacerse fuera del laboratorio en cualquier sistema operativo UNIX donde instalemos Java, en los escritorios virtualizados de la universidad[1], en los PC SC de los armarios del laboratorio o en los PCs del otro lado del aula.

[1] https://vdibroker.unavarra.es/

2. Los comandos en UNIX

La mayoría de los comandos que se han utilizado en la práctica anterior, y casi todos los que hay, no son otra cosa que ficheros ejecutables (programas), aunque hay algunas excepciones como por ejemplo `cd`, `type` y `source`, que son funcionalidades ofrecidas por la *shell* (pero por ahora podemos obviar esta diferencia).

Cuando escribimos un comando y pulsamos ENTER, la *shell* busca que exista un fichero con este nombre en algunas de las carpetas que conoce, y si lo encuentra lo ejecuta (lo cual implica crear un nuevo proceso donde se ejecuta ese programa mientras que el proceso de la shell se queda esperando a que termine). Si no encuentra ningún fichero con el nombre indicado nos devolverá un error indicando que no encuentra el programa correspondiente. Por ejemplo:

```
$ pwd
/home/alumno
$ c
c: command not found
```

Con el comando `which` se puede consultar la localización del fichero correspondiente a un comando. Por ejemplo:

```
$ which grep
/bin/grep
$ ls -l /bin/grep
-rwxr-xr-x 1 root root 211224 abr 29  2016 /bin/grep
```

Con esto se puede comprobar que el comando `grep` es un fichero que se encuentra en la carpeta `/bin` y tiene activado el permiso de ejecución para todos los usuarios. Busque en qué carpetas se encuentran los comandos `ls`, `grep`, `which` y `java`. ¿Se encuentran todos en la misma carpeta? ¿Ve alguna diferencia notable entre los ficheros?

Adicionalmente, en vez de delegar a la *shell* la tarea de encontrar el programa que debe ejecutar, se puede indicar de forma explícita indicando la ruta absoluta o relativa hasta el fichero correspondiente. Ejecute los comandos y compruebe si detecta alguna diferencia:

```
$ cd /bin
$ echo "esto es una prueba" | /bin/grep "es"
$ echo "esto es una prueba" | ./grep "es"
```

Como se ha comentado antes, la shell busca estos comandos en una lista de carpetas conocidas. Algunas carpetas habituales son:

```
/usr/bin
/bin
/usr/local/bin
/home/<usuario>/.local/bin
```

No obstante, la lista exacta y el orden de prioridad de estas carpetas depende de la configuración de la *shell*. La lista que su *shell* utiliza para buscar los programas se almacena en la variable de entorno[2] `PATH`. Pruebe a imprimirla para ver su contenido:

```
$ echo $PATH
```

De esta forma, la *shell* permite ejecutar como comandos aquellos ficheros que tengan habilitado el permiso de ejecución. No obstante, para qué el fichero pueda ejecutarse correctamente debe ser un binario en código máquina o un programa que el sistema operativo pueda interpretar[3].

La gran mayoría de los comandos que hemos utilizado hasta el momento son binarios compilados, habitualmente de programas escritos en `C` o `C++`. Por ejemplo, mire el contenido del comando `grep`:

```
$ head /bin/grep
```

Debería ver una serie de carácteres no imprimibles (habitualmente algo similar a ****) junto con alguna que otra cadena de texto legible.

Por el contrario, también existen comandos cuyo código fuente se interpreta en cada ejecución. Es posible que se encuentre con programas escritos en `Python`, pero lo más habitual es que sean programas directamente interpretados por una *shell*. Mire por ejemplo el contenido del comando `which`.

```
$ head /bin/which
```

Aquí verá texto completamente legible. Además, si se fija en la primera línea del programa verá una secuencia de texto habitual llamada shebang[4]. Esta línea inicia con los carácteres `#!` y continúa con la ruta absoluta al programa que debe interpretar las líneas posteriores. En este caso, `/bin/sh`, nos indica que el programa se interpretará por la *shell* `sh`.

Dependiendo del lenguaje en el que se haya escrito el programa, encontrará distintos contenidos en esta línea inicial. ¿Qué esperaría encontrar en ficheros que empiecen con las siguientes líneas?

```
#!/bin/bash
#!/usr/bin/python3
#!/bin/false
```

[2] https://www.hostinger.com/es/tutoriales/variables-en-bash
[3] https://www.freecodecamp.org/espanol/news/lenguajes-compilados-vs-interpretados/
[4] https://es.wikipedia.org/wiki/Shebang

3. Compilando y ejecutando un programa

En lo restante de esta práctica nos vamos a centrar en la programación de comandos en Java. Por desgracia, la versión de Java que hay instalada en los ordenadores del laboratorio no permite la ejecución de ficheros utilizando *shebang* (sí le funcionará en los ordenadores virtuales de la universidad). A su vez, Java no es un lenguaje que se compile a código máquina, por lo que su compilación tampoco genera binarios directamente ejecutables por el sistema operativo. En su lugar, el compilador de Java genera un fichero intermedio conocido como *Bytecode,* que posteriormente es interpretado y ejecutado por la máquina virtual de java (JVM)[5]. Esto al final quiere decir que lo que tendremos que lanzar para ejecutar nuestros programas Java es el propio programa `java` que es el que contiene la JVM, dándole el Bytecode de nuestro programa.

Utilizaremos como primer ejemplo el siguiente programa. Abra un editor de texto de su preferencia, copie el código que encontrará a continuación y guarde el fichero con el nombre `Echo.java`. Dedique un momento a leer y entender el código del programa.

```
class Echo {
  public static void main(String[] args) {
    for (int i = 0; i < args.length; i++) {
      System.out.print(args[i]);
      if (i < args.length - 1) System.out.print(" ");
    }
    System.out.println();
  }
}
```

Podemos compilar el programa utilizando el comando `javac`.

```
$ javac Echo.java
```

Si la compilación ha sido satisfactoria, deberíamos ver que se ha generado un fichero `Echo.class`. En este fichero se habrá almacenado el *Bytecode* generado por el compilador. Al ejecutar el comando `java`, éste leerá el fichero `Echo.class` e interpretará su contenido en la JVM.

Pruebe los siguientes comandos:

```
$ java Echo
$ java Echo a b c
$ java Echo a "b c"
```

¿Cuál es la diferencia entre el segundo y tercer ejemplo?

[5] https://www.geeksforgeeks.org/java/compilation-execution-java-program/

4.1 Implementando grep

En esta sección se pide implementar un programa en Java que replique el funcionamiento básico del programa `grep`. En concreto, trate de replicar los siguientes ejemplos:

```
$ cat /etc/services | grep web
$ find /etc | grep java
$ strings /bin/* | grep 'Free Software Foundation'
```

En estos ejemplos el programa leerá texto de la entrada estándar e imprimirá por pantalla aquellas líneas que contengan el texto indicado como argumento.

En Java, para leer texto de la entrada estándar puede utilizar la clase `Scanner`[6]. Se proporciona el siguiente fragmento de código a modo de ayuda:

```
import java.util.Scanner;
Scanner scanner = new Scanner(System.in);
while (sc.hasNextLine() {
  String line = scanner.nextLine();
}
```

Deberá poder ejecutar su programa de la siguiente manera:

```
$ cat /etc/services | java Grep web
$ find /etc | java Grep java
$ strings /bin/* | java Grep 'Free Software Foundation'
```

Punto de control 1 (40%): Muestre al profesor el programa indicado.

[6] https://docs.oracle.com/javase/8/docs/api/java/util/Scanner.html

4.2. Implementando find

En esta sección se pide implementar un programa en Java que replique el funcionamiento básico del programa `find`. En concreto, trate de replicar los siguientes ejemplos:

```
$ find /etc
$ find /var/log
$ find .
```

El programa `find`, cuando se ejecuta sin argumentos como en el ejemplo, lista de forma recursiva[7] los ficheros y carpetas que se encuentran a partir de la ruta indicada.

En Java, para realizar operaciones relacionadas con el sistema de ficheros se puede utilizar la clase `File`[8]. En la documentación puede encontrar todas las funcionalidades que ofrece esta clase, pero a continuación se muestran algunos ejemplos que pueden serle de utilidad.

```
File file = new File("/home");
String path = file.toString();
String name = file.getName();
boolean canRead = file.canRead();
boolean isDir = file.isDirectory();
boolean isFile = file.isFile();
String[] content = file.list();
```

Deberá poder ejecutar su programa de la siguiente manera:

```
$ java Find /etc
$ java Find /var/log
$ java Find .
```

Punto de control 2 (40%): Muestre al profesor el programa indicado.

[7] https://es.wikipedia.org/wiki/Recursi%C3%B3n_(ciencias_de_computaci%C3%B3n)

[8] https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/io/File.html

5. Un programa complejo

Como se ha podido ver, gran parte de la flexibilidad que ofrece Linux/UNIX es gracias a la concatenación de programas simples mediante *pipes*. Estos nos permiten realizar acciones complejas mediante la unión de funcionalidades más simples que ofrecen algunas herramientas que vienen por defecto.

Para ver un ejemplo, vamos a suponer que tenemos una entrada de texto y queremos obtener de aquellas líneas que contienen una cadena de texto dada, las 5 que más veces se repiten.

Con herramientas estándar, podemos realizar esta tarea de la siguiente forma:

```
$ strings /bin/* | grep 'Free Software Foundation' | sort | uniq -c | sort
-nr | head –n 5
```

Paso a paso, lo que está realizando es lo siguiente:

- `strings /bin/*`: Es el texto origen sobre el que se pide operar. En concreto, se extraen los carácteres imprimibles de los binarios del sistema.
- `grep 'Free Software Foundation'`: Se filtra buscando únicamente aquellas líneas que contienen el texto `'Free Software Foundation'`.
- `sort`: Se ordenan las líneas alfabéticamente.
- `uniq –c`: Se cuentan el número de veces que aparece la misma línea en el texto origen (requiere que el texto a la entrada esté ordenado).
- `sort –nr`: Se ordenan las líneas en base al número de veces que han aparecido, en orden descendiente.
- `head –n 5`: Se limita la salida a las 5 primeras líneas.

Ahora bien, si el volumen de datos a tratar es muy grande es posible que la utilización de estas funcionalidades simples no sea lo más eficiente posible y esto no nos permita realizar la tarea necesaria de forma efectiva. El problema a nivel de rendimiento que tiene esta concatenación de programas reside principalmente en la primera llamada al comando `sort`. Tal y como funciona `uniq –c`, resulta necesario ordenar previamente las líneas para obtener el resultado deseado.

Sin embargo, desde el punto de vista de los datos de entrada y la salida deseada, no es estrictamente necesario ordenar todas las líneas de texto.

Se pide replicar el funcionamiento de esta serie de programas, pero evitando ordenar el texto alfabéticamente. El programa deberá replicar el resultado a la salida, y la ejecución deberá ser de la siguiente forma:

```
$ strings /bin/* | java Mixed 'Free Software Foundation'
```

Punto de control 3 (20%): Muestre al profesor el programa indicado.