Introducción a Sistemas Operativos: Usando el shell

Clips xxx
Francisco J Ballesteros

1. Usando expresiones regulares

Vamos a resolver el problema que teníamos. Supongamos que tenemos nuestra aplicación `myapp` con su fichero de configuración en `$HOME/.myapp` para cada usuario, y que su aspecto es como habíamos visto:

```
shell /bin/sh
libdir /usr/lib
bindir /usr/bin
logdir /log
```

Vamos a hacer un script llamado `chkappconf` que compruebe la configuración para todos los usuarios del sistema. Por el momento vamos a usar un fichero llamado `config` que tiene justo el contenido que mostramos arriba como ejemplo de fichero de configuración. Luego es fácil trabajar con los ficheros de verdad. La idea es procesar los ficheros *enteros*, poco a poco, e ir generando nuevos "ficheros" como salida de comandos que procesan los datos que tenemos. En realidad, generamos bytes que fluyen por un pipe.

En este caso, vamos a necesitar utilizar *grep(1)*, o, concretamente *egrep(1)*. Este comando lee su entrada, o ficheros indicados en la línea de comandos, línea a línea y escribe aquellas que encajan con la expresión que le damos. Por ejemplo,

```
unix$ egrep dir config
libdir /usr/lib
bindir /usr/bin
logdir /log
```

El primer parámetro es la expresión que buscamos y el segundo es el fichero en que estamos buscando.

Otro ejemplo:

```
unix$ seq 15 | egrep 1
1
10
11
12
13
14
15
unix$
```

En este caso `egrep` lee de la entrada estándar, dado que no hemos indicado fichero alguno.

Las expresiones de `egrep` son muy potentes. Son de hecho un lenguaje denominado **expresiones regulares** que se utiliza en diversos comandos de UNIX, por lo que resulta muy útil aprenderlo. Mirando la página *egrep(1)* vemos hacia el final...

```
unix$ man egrep
...
SEE ALSO
     ed(1), ex(1), gzip(1), sed(1), re_format(7)
...
```

Y precisamente *re_format(7)* en este sistema documenta las expresiones regulares. Esto lo hemos hecho en un sistema OS X. Si usamos un sistema Linux podemos hacer la misma jugada

```
unix$ man egrep
...
SEE ALSO
awk(1),  cmp(1), diff(1), find(1), sed(1), sort(1),
       glob(7), regex(7).
...
```

y vemos que *regex(7)* documenta sus expresiones regulares. Las que usamos en este curso funcionan en ambos.

Una expresión regular es un string que describe otros strings. Decimos que un string encaja con la expresión si *contiene* uno de los strings que describe la expresión. Podemos definir las expresiones recursivamente como sigue:

- Cualquier carácter normal que no forma parte de la sintaxis de expresiones regulares encaja con el mismo. Por ejemplo, `a` describe el string `a`.
- La expresión "`.`" describe con cualquier carácter, pero sólo uno. Por ejemplo, `.` puede ser tanto `a` como `b`, pero no el string vacío ni tampoco on `ab`.
- Una expresión regular *r0* seguida de otra *r1* describe los strings que tienen un prefijo descrito por *r0* seguido de otro descrito por *r1*. Por ejemplo `ab` describe el string `ab` puesto que `a` puede ser `a` y `b` puede ser `b`. Igualmente, `.b` describe `ab` `bb` pero no `ba`.
- Si *r0* y *r1* son dos expresiones regulares, La expresión *r0*`|`*r1* describe los strings que describe alguna de las expresiones *r0* y *r1* (o que describen las dos).
- Si tenemos una expresión *r*, entonces *r*`*` describe los strings "" (la cadena vacía), los que describe *r*, los de *rr*, los de *rrr*, etc. (Repetimos la expresión cualquier número de veces, posiblemente ninguna).
- Si tenemos una expresión *r*, entonces *r*`+` es lo mismo que *r*`(`*r*`)*`. Esto es, *r* una o más veces.
- Si tenemos una expresión *r*, entonces *r*`?` es lo mismo que `(`*r*`)|()`. Esto es, *r* una o ninguna vez.
- `^` representa el principio del string en que buscamos encajes de la expresión regular. Por ejemplo, `^a` encaja con `ab` pero no con `ba`.
- `$` representa el final del string en que buscamos encajes de la expresión regular. Por ejemplo, `a$` encaja con `ba` pero no con `ab`.
- `\`*c* quita el significado especial a *c*, de tal modo que podemos utilizar caracteres que forman parte de la sintaxis de expresiones regulares como caracteres normales. Por ejemplo, `\\` encaja con `\` y `\.c` encaja con `.c` pero no con `ac`. En cambio `.c` encaja también con `ac`.
- `(`*r*`)` permite agrupar una expresión y describe los strings descritos por *r*. Por ejemplo, `(a|b)(x|y)` describe `ax` pero no `ab`.
- `[...]` describe cualquiera de los caracteres entre los corchetes. Y es posible escribir rangos como en `[a-c]` (de la `a` a la `c`). Por ejemplo, `[a-zA-Z0-9_]` es cualquier letra minúscula o mayúscula o dígito o bien "_". ¡Pero cuidado aquí con caracteres como "ñ"!
- `[^...]` describe cualquiera de los caracteres no descritos por `[...]`. Por ejemplo, `[0-9]` es cualquier carácter que no sea un dígito.

Veamos algunos ejemplos utilizando `seq` para usar `egrep` en su salida. Primero, buscamos un `1` seguido de un carácter:

```
unix$ seq 15 | egrep 1.
10
11
12
13
14
15
unix$
```

Ahora un carácter seguido de un `1`:

```
unix$ seq 15 | egrep .1
11
unix$
```

O bien `11` o bien `12`:

```
unix$ seq 15 | egrep '11|12'
11
12
unix$
```

Y tambien podemos combinar expresiones más complejas del mismo modo:

```
unix$ seq 15 | egrep '.2|1.'
10
11
12
13
14
15
unix$
```

Un `1` y cualquier cosa:

```
unix$ seq 15 | egrep '1.*'
1
10
11
12
13
14
15
unix$
```

¡Ojo al `.*`! Si usáramos `1*` entonces veríamos todas las líneas dado que todas contienen el string vacío y que `1*` encaja con el string vacío. Pero podríamos pedir las líneas que son cualquier número de unos:

```
unix$ seq 15 | egrep '^1*$'
1
11
unix$
```

Un `1` o el principio del texto y luego un `2` o un `3`:

```
unix$ seq 15 | egrep '(1|^)(2|3)'
2
3
12
13
unix$
```

Aunque tal vez sería mejor

```
unix$ seq 15 | egrep '(1|^)[23]'
2
3
12
13
unix$
```

para conseguir el mismo efecto.

Ahora cualquier línea que use sólo cualquier carácter menos los del 2 al 8:

```
unix$ seq 15 | egrep '^[^2-8]*$'
1
9
10
11
```

Líneas que tengan 3 una o más veces:

```
unix$ seq 15 | egrep '3+'
3
13
unix$
```

Quizá un 1 y un 3:

```
unix$ seq 15 | egrep '1?3'
3
13
unix$
```

Bueno, ya conocemos las expresiones regulares y podemos utilizar `egrep` para obtener las líneas de nuestro fichero de configuración que se refieren al shell:

```
unix$ egrep '^shell ' config
shell /bin/sh
unix$
```

Y también las que se refieren a directorios:

```
unix$
unix$ egrep '^[a-z]+dir ' config
libdir /usr/lib
bindir /usr/bin
logdir /log
unix$
```

Nótese que utilizar aquí "`dir`" como expresión habría sido seguramente un error. Líneas que contengan

algo como "/opt/bindir/ksh" habrían salido y no tienen por qué ser las que buscamos en este caso.