Taller Sistemas Operativos 08 a Noviembre 2017

8.1 - awk [1]

`awk` se parece, un poco, a `sed` .

La unidad de datos es el *registro* , que normalmente coicide con la *línea* .

Unas instrucciones para `awk` forman una secuencia de:

parte dónde { *programita* }

Y la secuencia puede ir precedida de un *prólogo* y seguida de un *epílogo* .

```
  cd ; cd tso/01/awks
  ln ../palabras/dias .
ls -l dias
awk '{print; print}'      dias                                    # 120
awk '{print
      print}'             dias
awk '/es/{print; print}'  dias     ;    awk '/es/ ; /es/'  dias
awk '/es/'                dias     ;    awk 'NR > 3'       dias
awk -f pawk01             dias
awk 'NR>3 {print $0,$0}' dias
awk 'NR>3 {print $0,$0} ; END {print "leidas", NR, "lineas"}' dias
```

Un *registro* se considera formado por una secuencia de *campos* .

```
awk '{print $1, "+", $2, "  ", $1 $2}' ../parejas/juntas
awk '{print $2, "+", $1, "  ", $2 $1}' ../parejas/juntas          # 220
awk '{print $2, $1}'                   ../parejas/juntas
awk '{print $3, $2, $1}'               ../ntuplas/simpson
awk '{printf ("%9s %9s %9s\n",
               $3, $2, $1)   }'        ../ntuplas/simpson
```

8.2 - bash

a) [2] Podemos considerar los comandos de UNIX como *expresiones booleanas* .

Tenemos operadores *and* y *or* . Son `&&` y `||` , respectivamente.

El resultado se pueda usar en un `if` o en un `while`.

La *ejecución* de ambos operadores es *secuencial*, y *de izquierda a derecha* .

⇒ *si el resultado está decidido, no se continua.*

```
  cd ; cd tso/01/progs
  mkdir bashes ; cd bashes; rm -f ? ??
echo hola >a ; echo hello >c
cp a c || echo paso por A
cp b c || echo paso por B                                          # 330
  echo adios >a
cp a c && echo paso por C
cp b c && echo paso por D
cp g c || cp f c || echo paso por E
```

[1] apuntes SSAA, cap. 17, pag. 143-147, 149,158

[2] apuntes SSAA, cap. 22, pag. 234-235

b) [3] (doble) evaluación

```
date    ;    date +%M:%S
man date
echo la fecha es: ‘date‘
```

Cuando el *intérprete de comandos* (o `bash`), encuentra un par de comillas inversas, ejecuta el comando encerrado y coloca en ese lugar la *salida estandar* de ese comando.

```
who | head -2
who | head -2 > who_sal_‘date +%M:%S‘                                    # 520
sleep 30
who | head -2 > who_sal_‘date +%M:%S‘
more  who_sal_*
```

Guardamos dos *instantaneas* en sendos ficheros que tienen la fecha (parte) en su nombre.

El mismo efecto se consigue con `... $(date +%M:%S)` .

8.3 - make [4] [5]

```
  cd ; cd tso/01/progs/proy32
make codp.o
make -n fi32
cc fi32.c -o fi32 codp.o                                                 # 620
ls -l
./fi32
hola, que tal
echo hola, que tal | ./fi32
```

Hemos compilado y ejecutado una aplicación con código en tres ficheros. Es un poco chapuza. Vamos a apoyarnos en `Makefile` .

```
  rm fi32 codp.o
./fi32
cp pmk-1 Makefile
make fi32                                                                # 720
echo otra vez aqui | ./fi32
echo otra vez aqui > prueba01.entr
./fi32 < prueba01.entr | tee prueba01.sald
cat prueba01.sald
```

Usaremos un fichero para `make` : `Makefile` .

La entrada y salida de una prueba la guardamos, para que no se olvide ... .

`tee` guarda copia y reenvía a salida (... estandar).

```
cat pmk-2 >> Makefile
  rm prueba01.sald
make prueba01.sald
cat prueba01.sald                                                        # 820

  rm prueba01.sald fi32 codp.o
make prueba01.sald
cat prueba01.sald
```

Vemos que `make` no sólo se encarga de ejecutar `fi32`, sino que si hace falta recompila.

---

[3] apuntes SSAA, cap. 22, pag. 231

[4] apuntes SSAA, cap. 18, pag. 167-

[5] apuntes SSAA, cap. 23, pag. 265