UNIVERSIDAD NACIONAL DE LA MATANZA

Departamento de Ingeniería e Investigaciones Tecnológicas

Sistemas Operativos (Plan 2009)

Jefe de Cátedra: Fabio E. Rivalta
Equipo de Docentes: Boettner F., Catalano L., de Lizarralde R, Villamayor A.
Auxiliares docentes: Loiacono F., Hirschfeldt D., Piubel F., Rodriguez A., Segura L., Fernandez Piñeiro, Radice A.

SCRIPTING -> AWK

SISTEMAS OPERATIVOS UNLAM.

OBJETIVOS

- Poder definir AWK
- Saber cuando usarlo
- Comprender como funciona
- Poder construir un script

ROADMAP

- Definimos AWK
- Entendemos como funciona AWK
- Codeamos
- Sacamos conclusiones

QUÉ ES AWK?

AWK

-:$ man awk

- Lenguaje dinámico de programación.
- Creado en 1977 por **A**ho **W**einberg y **K**ernighan.
- Se usa para procesar datos basados en texto (ficheros o flujos de datos).
- Entrada por default -> STDIN
- Salida por default -> STDOUT

Mas info: ejecutar en terminal -> man awk

CÓMO TRABAJA?

Comportamiento

- Lee la entrada de a un registro a la vez
- Compara el contenido del registro con un patrón
- Si coincide ejecuta una acción

```
mientras (  entrada.RegistrosPorLeer() ) {
   Registro = entrada.LeerRegistro()
   si ( CumplePatron (registro) )
        acción()
}
```

AWK en acción

cat introawk | awk

STDOUT
cat

STDIN
AWK

Aho Weinberg Kernighan
1977
Instrucción scripting

AWK

NR	NF	$ 0		
		$ 1	$ 2	$ 3
1	3	Aho	Weinberg	Kernighan
2	1	1977		
3	2	Introducción	Scripting	

STDOUT
AWK

- NR: Number Record. Numero de fila actual.
- NF: Number of Fields. Cantidad de campos de la fila
- $0: Fila actual
- $1: Primer campo
- $2: Segundo campo

Sintaxis

Comando simple

- awk ‘<patrón$_1$>{<acción$_1$>}…<patrón$_n$>{<acción$_n$>}’ <entrada$_1$>… <entrada$_n$>
- Patrón: Permite saber cuando aplicar la acción.
    - Es Opcional -> Si no se especifica se aplican las acciones a toda la entrada
- Acción: acciones a realizar sí se cumple el patrón.
    - Es Opcional -> Por defecto imprime el registro
- Entradas
    - Es Opcional -> Si no se especifica se tomará el stdin

NOTA: cada par patrón acción se evalúa independientemente del otro, así que un registro podría cumplir con mas de un patrón. Sería equivalente a if(patron) acción2; if (patron2) accion2 …..

Variables predefinidas

VARIABLE	DESCRIPCIÓN
ARGC	Número de argumentos pasados en la linea de comandos
ARGV	Arreglo de argumentos de la linea de comandos de awk
FILENAME	Nombre del archivo de entrada actual
RS	Separador de lineas o registros (Record Separator). Si no se especifica asume por default NewLine.
ORS	Separador de lineas o registros de salida (Output Record Separator). Si no se especifica asume por default NewLine.
FS	Separador de campos (Field Separator)
OFS	Separador de campos de salida (Output Field Separator)
OFMT	Formato de salida para números
NR	Número de registro actual (Number record)
NF	Cantidad de campos del registro actual (Number Fields)

A Jugar

EJ1 - Procesos

Mostar un proceso del sistema

NR=1

```
[adrianradice@MacBook-Air-de-Adrian borrar % ps -all
  UID   PID  PPID       F CPU PRI NI       SZ    RSS WCHAN     S             ADDR TTY           TIME CMD
    0 15221 15220    4106   0  31  0  4304888      8 -       Ss               0 ttys000    0:00.06 login -pf adrianradice
  501 15222 15221    4006   0  31  0  4297216      8 -       S                0 ttys000    0:00.11 -zsh
  501 15341 15222    4006   0  31  0  4297216   1204 -       S                0 ttys000    0:00.52 zsh
    0 16635 15341    4106   0  31  0  4268788    944 -       R+               0 ttys000    0:00.01 ps -all
    0 15600 15220    4106   0  31  0  4304888      8 -       Ss               0 ttys001    0:00.05 login -pfl adrianradice /bin/bash -c exec -la zsh /bin/zsh
  501 15601 15600    4006   0  31  0  4297216      8 -       S+               0 ttys001    0:00.07 -zsh
  501 16535 16502    4006   0  31  0  4297300   2032 -       Ss+              0 ttys002    0:00.15 /bin/zsh -l
```

```
ps -all | awk ‘NR == 2’
```

El resultado de ps -all posee un encabezado (fila 1), así que debemos evitar el primer registro

EJ2 - Procesos

Mostar dos proceso del sistema

```
[adrianradice@MacBook-Air-de-Adrian borrar % ps -all
  UID   PID  PPID        F CPU PRI NI       SZ    RSS WCHAN     S             ADDR TTY           TIME CMD
    0 15221 15220     4106   0  31  0  4304888      8 -       Ss               0 ttys000    0:00.06 login -pf adrianradice
  501 15222 15221     4006   0  31  0  4297216      8 -       S                0 ttys000    0:00.11 -zsh
  501 15341 15222     4006   0  31  0  4297216   1204 -       S                0 ttys000    0:00.52 zsh
    0 16635 15341     4106   0  31  0  4268788    944 -       R+               0 ttys000    0:00.01 ps -all
    0 15600 15220     4106   0  31  0  4304888      8 -       Ss               0 ttys001    0:00.05 login -pfl adrianradice /bin/bash -c exec -la zsh /bin/zsh
  501 15601 15600     4006   0  31  0  4297216      8 -       S+               0 ttys001    0:00.07 -zsh
  501 16535 16502     4006   0  31  0  4297300   2032 -       Ss+              0 ttys002    0:00.15 /bin/zsh -l
```

ps -all | awk ‘NR > 1 && NR < 4’

AND = &&

EJ 3 - contador de palabras

Hacer un script awk para contar palabras de una entrada

```
awk { cant+=NF } END {print cant} < Entrada
```

EL BLOQUE END SE EJECUTA UNA VEZ
AL FINALIZAR CON LA ENTRADA

El bloque END se ejecuta luego de procesar toda la entrada. Por ejemplo se utiliza para devolver resultados / totales, resúmenes, pies, etc

EJ 4 - Contando palabras repetidas

Hacer un script awk para contar palabras repetidas en una entrada

```
awk '{ for( i=1; i<=NF; i++ ) cant[$i]++ } END { for ( palabra in cant ) print cant[palabra]}’
```

Array asociativo
-> key: palabra value: cant

Foreach

Los espacios separan cada palabra, así que si el FS es el espacio, cada campo seria una palabra.
Empleamos un array asociativo donde la clave es la palabra y el valor es el contador.

EJ 5 - Capitalizar entrada

Hacer un script awk para poner en mayúscula la primer letra de cada palabra. Cada registro es una palabra

```
awk ‘{ sub( /^[a-z]/, toupper( substr( $0,1,1 ) ),$0 );  print $0  }' entrada
```

Obtener un segmento del string.
Substr (<cadena>, <start>, <cant>)

Convertir a mayúscula
toupper(<cadena>)

Expresión regular. (VER ANEXO)
En este caso indica primer carácter (^) minúscula [a-z]

Sustituir un elemento de una cadena por otro
sub(<regexp>, <replacement>, <target>)

EJ 6 - Parsear número de teléfono

Obtener los token de un número telefónico

```
echo "+54(011)4782-0991" |
    awk '
            {
                split($0,codPaisResto,"(");
                split(codPaisResto[2],areaNumero,")")
            }
            END {
                print "pais:" codPaisResto[1];
                print "area: " areaNumero[1];
                print "numero: " areaNumero[2]
            }
    '
```

Split: toma una cadena y la fracciona en un array de acuerdo a un delimitador

<tamVec> = split (<cadena>, <arrayDestino>, <delimitador>)

Primero separamos el código de país del numero telefónico con el “(“
Luego separamos el código de area del numero telefónico con el carácter “)”
Notar que los caracteres delimitadores se pierden, y que los arreglos se indexan desde el 1

EJ 7 - Más de una entrada

A fines didácticos concatenar 3 entradas

```
awk ‘{ print NR $0 }’ entrada1 entrada2 entrada3
```

entrada1
A
B

entrada2
C

entrada3
D
E
F

Salida
1 A
2 B
3 C
4 D
5 E
6 F

Note que el número de registro hace referencia al número de registro procesado y no del archivo

EJ 8 - BASH + AWK

Realizar un script bash que permita obtener los autos deudores del 2018 dada la siguiente entrada

ENTRADA
AA122DD2019SI
AA123DD2018SI
AB912DD2020SI
CA124DD2019NO

```
#!/bin/bash
modelo="2018"
debe="SI"
awk -v mod=$modelo
    -v deb=$debe
    '
        BEGIN {
            FIELDWITHS = "7 4 2"
        }
        $2 == mod && $3 == deb {
            print $1 " " $2
        }
    ' "$1"
```

Si no tenemos caracteres delimitadores de campos, y los campos son de longitud fija, podemos usar su longitud para diferenciar los campos. La longitud de los campos (FIELDWITHS) debemos indicarlos en el bloque BEGIN que se ejecuta una vez antes de comenzar a procesar la entrada.
Note que para compartir las variables de bash en AWK tuvimos que enviarlas como parámetros de awk (-v)

EJ 9 - Total de compra

Dado el siguiente archivo que representa el detalle de una compra, indicar el total de la misma

Producto Cantidad PU descuento
CPU 1 8000
Memoria 2 4000
Disco 1 7500

EJ 9 - Total de compra

ANALIZEMOS

- El total de la compra es Cantidad * PU
- Separador de campos (espacio)
- Separador de registros (salto de linea)
- Campos de interés 2 (Cantidad) y 3 (PU)

Producto Cantidad PU
CPU 1 8000
Memoria 2 4000
Disco 1 7500

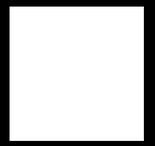
EJ 9 - Total de compra

Código

```
#!/usr/bin/awk -f
BEGIN {
    # inicializamos las variables
    total = 0
}
# Si no es el encabezado y una linea vacia
NR > 1 && length($0) {
    #recuperamos los campos
    cantidad = $2
    pu = $3
    # calculamos el subtotal del detalle
    total += cantidad * pu
}
END {
    # imprimimos el total de la compra
    print total
}
```

Producto	Cantidad	PU
CPU	1	8000
Memoria	2	4000
Disco	1	7500

- Inicializamos la variable para acumular el total
- Evitamos el encabezado y lineas en blanco
- Imprimimos el total



Note que esta vez construimos un script awk que puede ser invocado desde cualquier script bash. Nos permite reutilizarlo y separar BASH de AWK. También note que de esta forma no son necesarias las ‘ ‘

ÚLTIMO

EJERCICIO

Dado el siguiente archivo:

```
APYN|DOC|P1|P2|SEXO|ACTIVO:Pablo Perez|12|7|7|M|1:Oscar Diaz|2|7|2|M|1:Pablo Escobar|22|4|7|F|0:
```

Obtener el total de alumnos activos distinguiendo cantidad por sexo, mostrar los alumnos activo, y informar un resumen de la cantidad de alumnos promocionados, aprobados y que recursan.
Se desea ademas que el reporte de alumnos activos sea un script independiente al del resumen.

PROCESAMIENTO ENTRADA

```
#!/usr/bin/awk -f
BEGIN {
    RS=":"
    FS="|"
    ORS_OLD=ORS
    OFS_OLD=OFS
    ORS=":"
    OFS="|"
    cont=0
    res["M"]=0
    res["F"]=0
    res["PROM"]=0
    res["APRO"]=0
    res["REC"]=0
}
$6 == 1 {
    cont++
    res[$3>=7 && $4>=7?"PROM":$3>=4&&$4>=4?"APRO":"REC"]++
    print cont,$1,$2 > "alumnosActivos"
    res[$5]++
}
END {
    ORS=ORS_OLD
    OFS=OFS_OLD
    print "Total Alumnos Activos: ", cont
    print "|-> Mujeres: ", res["F"]
    print "|-> Hombres: ", res["M"]
    print "|-> Promocionados: ", res["PROM"]
    print "|-> Aprobados: ", res["APRO"]
    print "|-> Recursa: ", res["REC"]
}
```

- Utilizamos un array asociativo a modo de tabla (RES) para llevar el conteo que nos piden.
- La modificación de ORS y OFS es solo con fin didáctico. Podría evitarse y trabajarse con los default

INFORME DE ALUMNOS ACTIVOS

```
#!/bin/bash
echo "Resumen: "
./HolaAWK.awk < $1
awk -v RS=':' -v FS='|' -v OFS='\t' 'BEGIN {printf( "Id\tAPYN\t\tDNI\n" ) }{ print $1,$2,$3 }' < "alumnosActivos.txt"
```

SAQUEMOS CONCLUSIONES

GRACIAS

ANEXO

Repaso expresiones regulares

OPERADORES

- `.` Significa cualquier caracter.
- `^` Indica el principio de una línea.
- `$` Indica el final de una línea.
- `*` Indica cero o más repeticiones del caracter anterior.
- `+` Indica una o más repeticiones del caracter anterior.
- `\<` Indica el comienzo de una palabra.
- `\>` Indica el final de una palabra.
- `\` Caracter de escape. Da significado literal a un metacaracter.
- `[ ]` Uno cualquiera de los caracteres entre los corchetes. Ej: [A-Z] (desde A hasta Z).
- `[^ ]` Cualquier caracter distinto de los que figuran entre corchetes: Ej: [^A-Z].
- `{ }` Nos permiten indicar el número de repeticiones del patrón anterior que deben darse.
- `|` Nos permite indicar caracteres alternativos: Ej: (^|[?&])
- `( )` Nos permiten agrupar patrones. Ej: ([0-9A-F]+:)+