+Laboratorio de
Sistemas Operativos

Práctica 3: AWK.

Tecnológico de Monterrey

Objetivos

Aprender y aplicar la herramienta **AWK** que fue una gran herramienta antes de los 90, y hoy en día retoma su importancia para trabajar con archivos tales como enormes logs de bases de datos, *sensor data, networking streams*, XML y otros (además de los convencionales). Es de gran poder y utilidad cuando se requiere examinar o transformar archivos de texto grandes.

ÍNDICE

AWK
Imprimir (o más precisamente “enviar a stdout”):
Patrones:
Variables:
Funciones estándar en awk:
Manejo de cadenas (string)
Algunas de las funciones para el manejo de strings con que cuenta el awk son:
Manejo de ciclos y condicionales
Condicional 3: Operador ? (ternary)
Ciclo 2: Do-While
Ciclo 3: For
Paso de parámetros a awk:
= = = = Laboratorio = = = =||

AWK

Awk es una herramienta similar a grep en el sentido de que ambos son herramientas de *pattern matching*. Recuerdas que grep te encuentra y despliega líneas que incluyen un patrón tal como la palabra Navajos, o que "matchean" alguna expresión regular como grep ‘[0-9]*’, que te encontraría todas las líneas que incluyen algún número. Es lo mismo que hace Awk; pero además, por cada línea que cumple con la expresión regular correspondiente, se puede ejecutar una *acción*: un programa escrito en el lenguaje de Awk:

$ **cat** finalDeElVertigo

¡Conciencia nunca dormida,
mudo y pertinaz testigo
que no dejas sin castigo
ningún crimen en la vida!
La ley calla, el mundo olvida
mas, ¿quien sacude tu yugo?
Al sumo hacedor le plugo
que a solas con el pecado,
fueses tú para el culpado
delator, juez y verdugo!
-- de “El Vertigo”, poema/drama de Gaspar Núñez de Arce

$ cat finalDeElVertigo | **awk** ‘/ida/ {print $1} ‘

Conciencia
ningún
La

El programa awk, encerrado entre ‘ ‘ dice: por cada línea de stdin que contenga el patrón “ida”, saca para stdout el primer campo de la línea ($1). Ojo ¿por qué necesitamos los ‘ ‘? ¿Por qué no podemos usar dobles comillas? Respuesta: en la práctica anterior…

Awk divide cada renglón de entrada en campos automáticamente. Como default, cada campo se considera separado del siguiente por uno o más espacios en blanco. Cada campo se referencía dentro de la *{acción}* utilizando las variables $1, $2, $3, $4, $5 …; $0 representa toda la línea. Es posible definir otros caracteres que no sean "espacios en blanco" para separar campos; y otros diferentes de "newline" para separar renglones**.** Mmm ¿como le harías para que se imprimiera no el primer campo sino el enésimo… pero el valor de “n” se lo quisieras pasar a awk por medio de un script del shell… ?

Otros Ejemplos:

```
user@localhost home $ who
juan   tty5    Sep   29   17:25
pepe  tty2    Ago   27   08:23

[user@localhost home]$ who |  awk ‘{print $1}’
juan
pepe
[user@localhost home]$ who | awk ‘{print $5}’
17:25
08:23
```

En los dos ejemplos anteriores se está omitiendo el patrón; eso significa que la acción se ejecuta

para todas las líneas, mientras que en el ejemplo de más arriba sólo se ejecutaba para aquellas líneas que tuvieran “ida”.

AWK puede hacer cálculos sobre los datos de entrada numéricos. *Ejemplo*:

```
user@localhost home $ cat datos
10
10
10
10
10

[user@localhost home]$ cat suma.awk

BEGIN {s=0}
      {s=s+$1}
END   {print s, NR}

[user@localhost home]$ cat datos | awk -f suma.awk
50 5
[user@localhost home]$
```

En el ejemplo anterior, el programa awk se ha puesto en un archivo suma.awk, en lugar de ponerlo entre ‘ ‘ en la línea de comando. El programa consiste de tres líneas patrón-acción. Se utilizaron dos patrones predefinidos, **BEGIN y END**, una variable predefinida en AWK: NR (número de renglones) y una variable definida por el usuario, "s". La acción que sigue al BEGIN se ejecuta una sola vez al principio, antes de considerar todos los demás patrones. La acción que sigue al END se ejecuta una sola vez al final. En el patrón/acción intermedio, {s=s+$1} obsérvese que se ha omitido el patrón: sólo está la acción. Entonces esa acción se ejecuta para todos los renglones de entrada, dando como resultado que se acumula en “s” el primer campo de cada renglón de entrada.

Nota la indentación. *Es buen estilo* en programas awk que el patrón comience en la columna 1, y que las acciones correspondientes al patrón estén alineadas a la derecha del mismo.

Los renglones se consideran separados unos de otros por el valor de la variable RS (“record separator”) --default, newline. Los campos de cada renglón están separados por el FS (“Input field separator” --default, espacios. Estas variables podrían redefinirse en el BEGIN si es necesario. Tanto el valor de FS como el de RS puede ser cualquier expresión regular encerrada entre comillas, por ejemplo FS = “, +” (coma seguida de uno o más espacios - consulta man awk para la sintaxis completa de expresiones regulares). El algoritmo que ejecuta el comando AWK es entonces, de manera más detallada:

Si está el patrón BEGIN, ejecuta la acción que sigue al BEGIN
Para cada renglón del archivo de entrada separado del siguiente renglón por el valor de RS:
    Separa los campos del renglón según el FS, y asignarlos a las variables $1, $2...
    Para cada patrón del programa AWK
        Si el patrón concuerda con la línea, ejecuta la acción correspondiente
    End Para cada
End Para cada
Si está el patrón END, ejecuta la acción que sigue al END

Observa en el algoritmo anterior que se pueden tener varios patrones, no solo uno, que se ejecutan en secuencia por cada línea: ¿que te desplegaría el siguiente programa awk? si se ejecuta contra el archivo finalDeElVertigo? Primero, considéralo siguiendo el algoritmo anterior, y **después**, pruébalo!

BEGIN { print "¿Que le plugo --es decir se le antojó-- al sumo hacedor (Dios) ?" }
/plugo/ {print $0 }
/ado/ {print $0}
/juez/ {print $0}

#ojo en este ejemplo no hace falta el patrón END. BEGIN y END son patrones especiales; no se utilizan para agrupar statements como en Java. Y pueden ser innecesarios. {print $0} es un programa completo awk que copia stdin a stdout. Pruébalo!

Ejemplo: Calcule la estimación de la varianza de los siguientes números del archivo y.x:

```
[user@localhost home]$ cat y.x
0.0
10.0
0.0
10.0
0.0
10.0
0.0
10.0
```

El programa awk:

```
BEGIN {
        n=0
      }
      {
        n++
        val[n]=$1
        s+=$1
      }

END   {
        mu=s/n
        for (i in val) {
              d=val[i]-mu
              s2+=d*d
             }
        print sqrt(s2/n)
        print sqrt(s2/(n-1))
      }
```

En este ejemplo se ha hecho uso de un arreglo, "val" para acumular cada uno de los valores del primer campo de la entrada. El arreglo no es necesario declararlo como tal, como ocurriría en otros lenguajes de programación. La proposición "***For i in val***" recorre todos los elementos de ese arreglo. Si este archivo tiene el nombre sd_awk, entonces la forma de invocarlo es la siguiente:

```
[user@localhost home]$ awk -f sd_awk y.x
```

Por cierto, las variables se inicializan en cero automáticamente, por tanto se hubiera podido omitir el “BEGIN {n=0}”

Ejemplo: Calcula el factorial de cada uno de los números enteros del siguiente archivo de datos:

```
[user@localhost home]$ more x
3
6
5
7
```

Creamos el archivo “fac” en cualquier editor, tal que contenga el siguiente programa:

```
[user@localhost home]$ cat fac
{
   f=1
   for (i=$1; i>1; i--) {
   f=f*i
   }
printf "%d \n", f
}
```

El programa incluye una sola “acción” que se ejecuta para cada renglón de la entrada. Este es el resultado:

```
[user@localhost home]$ awk -f fac x
6
720
120
5040
```

Imprimir (o más precisamente “enviar a stdout”):

Awk cuenta con varias funciones para imprimir a la salida estándar. La primera, **print** tiene la siguiente sintaxis:

```
print <lista-de-expresiones>
```

La segunda, **printf** tiene la siguiente sintaxis:

```
printf <formato> , <lista-de-expresiones>
```

El **formato** puede ser alguno de los siguientes:

Formato:	**Imprime:**
%d	Números enteros
%f	Números flotantes

%c Carácter
%s Cadenas

A continuación se muestran algunos ejemplos:

```
[user@localhost home]$ cat entrada.txt
blanco          a             1             0.8126
rojo            b             2            0.4911
verde           c             3             0.2315
azul            d             4             0.7966
amarillo        e             5             0.2366
gris            f             6             0.3291
negro           g             7             0.9416
café            h             8             0.9584
naranja i              9              0.1502
morado          j             10             0.4887
[user@localhost home]$ awk '{print "hola mundo"}' entrada.txt
hola mundo
hola mundo
… (y asi diez veces… se está ignorando todo el contenido de la entrada!)

[user@localhost home]$ awk '{n=2; print "columna " n ": " $n}' entrada.txt
columna 2: a
columna 2: b
columna 2: c
columna 2: d
[user@localhost home]$ awk '{ printf "color %s, letra %c, numero %d, valor %f \n", $1,$2,$3,$4 }'
entrada.txt
color blanco, letra a, numero 1, valor 0.812600
color rojo, letra b, numero 2, valor 0.491100
color verde, letra c, numero 3, valor 0.231500
color azul, letra d, numero 4, valor 0.796600
```

Patrones:

Como indicamos, se pueden utilizar varias líneas patrón {acción}, donde cada patrón es evaluado cada vez que se procesa un nuevo renglón. Los distintos tipos de patrones son:

Patrón	Ejemplo
BEGIN	`BEGIN { contador=0 }`
END	`END { promedio=suma/elementos; print promedio }`
expresión regular	`/tec.*/ { print “contiene palabra que comienza con tec: ” $0}`
expresión relacional	`$1 ~ /Maria/ {print “primer columna igual a Maria: ” $0}`
expresión lógica	`/Pedro/ | /Paramo/ {print “linea contiene Pedro o Paramo: ” $0}`
rango	`/voz/,/madre/ {print “contiene en ese orden, voz y luego madre: ” $0}`

La siguiente tabla muestra los distintos tipos de patrones y sus expresiones (obtenidos del man de AWK):

Patrón:	Expresión:
especial	BEGIN , END
/expresión-regular/	c matches the non-metacharacter c. \c matches the literal character c. . matches any character including newline. ^ matches the beginning of a string. $ matches the end of a string. [abc...] character list, matches any of the characters abc.... [^abc...] negated character list, matches any character except abc.... r1|r2 alternation: matches either r1 or r2. r1r2 concatenation: matches r1, and then r2. r+ matches one or more r's. r* matches zero or more r's. r? matches zero or one r's. (r) grouping: matches r. r{n} r{n,} r{n,m} One or two numbers inside braces denote an interval expression, r is repeated n to m times.
expresión-relacional	~ , ~!
expresión-lógica	&& , || , !
rango	/expresion1/,/expresion2/

Ejemplo: Imprime las líneas donde aparece la palabra “Pedro” o la palabra “Paramo”, toma como entrada el archivo novela que se encuentra más adelante en la sección Laboratorio.

```
[user@localhost home]$ awk '/Pedro/|/Paramo/{print $0}' novela
Vine a X porque me dijeron que aca vivia mi padre, un tal Pedro Paramo.
Y de este modo se me fue formando un mundo alrededor de la esperanza que era aquel señor llamado Pedro
Paramo, el marido de mi madre.
```

Variables:

Las variables en awk pueden tener un valor numérico o de texto y no es necesario declararlas. Un nombre de variable se define por una secuencia de letras, dígitos o guiones bajos, y nunca comienza con un número; los nombres de variable son sensibles a las mayúsculas y minúsculas. La sintaxis para asignar algún valor a una variable es la siguiente:

```
variable=valor
```

Para recuperar el valor que tiene almacenado una variable únicamente se indica su nombre, por ejemplo:

```
[user@localhost home]$ awk '{ n=5; print “n=” n }' archivo
n=5
```

```
n=5
```

Existen variables predefinidas, como:

NF Contiene el número de campos en el renglón actual.
NR Indica el número total de renglones procesados hasta el momento.
FS El separador de campos (field separator), es un string que por default es un espacio, pero puede redefinirse en el BEGIN y puede ser cualquier expresión regular entre comillas (no entre / /)

RS El separador de renglones (record separator); por default un "newline"

Ejemplo: Imprime de dos formas el valor de la columna número dos:

```
[user@localhost home]$ cat datos
set-a              0.1945              0.9889              0.5444
set-b              0.5752              0.9445              0.9552
set-c              0.1684              0.0983              0.4060
set-d              0.8089              0.1956              0.1082
set-e              0.9621              0.5971              0.2042
set-f              0.5131              0.4257              0.2132
set-g              0.6182              0.1894              0.0910
set-h              0.2514              0.9625              0.0262
set-i              0.1182              0.5078              0.3755
set-j              0.8187              0.2381              0.0061

[user@localhost home]$ awk '{ n=2; print "Columna " n ": " $n " " $2 }' datos
Columna 2: 0.1945 0.1945
Columna 2: 0.5752 0.5752
Columna 2: 0.1684 0.1684
Columna 2: 0.8089 0.8089
Columna 2: 0.9621 0.9621
Columna 2: 0.5131 0.5131
Columna 2: 0.6182 0.6182
Columna 2: 0.2514 0.2514
Columna 2: 0.1182 0.1182
Columna 2: 0.8187 0.8187
```

Funciones estándar en awk:

En la parte de programa de awk se pueden utilizar distintas funciones, ya implementadas, para realizar cálculos sobre los datos contenidos en el archivo a procesar, algunas de estas son:

- **sqrt()** Raíz cuadrada
- **log()** Logaritmo base e
- **exp()** Exponencial.
- **int()** Parte entera de un argumento.

Ejemplo: Calcule el logaritmo de cada número ubicado en la 3er columna numérica, e imprímalo en una cuarta columna nueva.

```
[user@localhost home]$ cat datos
set-a              0.1945              0.9889              0.5444
set-b              0.5752              0.9445              0.9552
set-c              0.1684              0.0983              0.4060
set-d              0.8089              0.1956              0.1082
set-e              0.9621              0.5971              0.2042
set-f              0.5131              0.4257              0.2132
```

```
set-g              0.6182              0.1894              0.0910
set-h              0.2514              0.9625              0.0262
set-i              0.1182              0.5078              0.3755
set-j              0.8187              0.2381              0.0061

[user@localhost home]$ awk '{ a=log($4) ; print $0 "\t" a}' datos
set-a              0.1945       0.9889     0.5444              -0.608071
set-b              0.5752       0.9445     0.9552              -0.0458345
set-c              0.1684       0.0983     0.4060              -0.901402
set-d              0.8089       0.1956     0.1082              -2.22377
set-e              0.9621       0.5971     0.2042              -1.58866
set-f              0.5131       0.4257     0.2132              -1.54552
set-g              0.6182       0.1894     0.0910              -2.3969
set-h              0.2514       0.9625     0.0262              -3.642
set-i              0.1182       0.5078     0.3755              -0.979497
set-j              0.8187       0.2381     0.0061              -5.09947
```

Manejo de cadenas (string)

Algunas de las funciones para el manejo de strings con que cuenta el awk son:

- **length()** longitud de la fila.
- **substr()** extrae un substring de un string.
- **Index()** regresa el punto inicial de un substring en un string.
- **sub()** sustituye la primera ocurrencia de una expresión regular por una cadena.
- **gsub()** igual que **sub()**, pero sustituye todas las ocurrencias.
- **split()** divide una cadena en elementos de un arreglo.

Ejemplo: Imprima el contenido de la variable de entorno PATH, y separe los directorios por espacios. Se muestran dos formas de hacerlo:

```
[user@localhost home]$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/games:/home/usuario/bin:/db2/speech/htk/bin.linux

[user@localhost home]$
echo $PATH | awk '{ n=split($0,arr,/:/); for (ii=1; ii<=n; ++ii) { printf "%s ",arr[ii] } printf "\n"
}'
/usr/local/bin /usr/bin /bin /usr/games /home/usuario/bin /db2/speech/htk/bin.linux

[user@localhost home]$ echo $PATH | awk 'BEGIN {FS=":"}; {for (ii=1; ii<=NF; ++ii) {print $ii} }'
/usr/local/bin /usr/bin /bin /usr/games /home/usuario/bin /db2/speech/htk/bin.linux
```

En el segundo ejemplo se define el "field separator" FS como el caracter dos puntos; de esta manera, cada directorio queda asignado a las variables $1, $2 etc. y basta desplegarlas en orden con un "for". Observa que en el for estamos utilizando la variable predefinida NF (number of fields), y que podemos utilizar una variable pegada al signo de pesos: $ii

Manejo de ciclos y condicionales

Para las condiciones y los ciclos, AWK maneja una sintaxis muy similar a C. Además cuando es una sola instrucción no es necesario utilizar el símbolo ";" para delimitarla aunque **se recomienda su uso**.

Para los condicionales existen 4 estructuras:

1. If simple
2. Cláusula IF-Else
3. Cláusula IF-Else ladder (escalera)
4. Ternary (?)

Condicional 1: **IF simple e IF-ELSE**

```
if (conditional-expression)
{
       action;
}
else
{
               action1;
        action2;
}
```

Condicional 2: **IF-ELSEIF (IF-ELSE Ladderl)**

```
if(conditional-expression1)
       action1;
else if(conditional-expression2)
       action2;
else if(conditional-expression3)
       action3;
       .
       .
else
       action n;
```

Condicional 3: **Operador ? (ternary)**

```
conditional-expression ? action1 : action2 ;
```

La sintaxis de los condicionales también es parecida en C, aunque los símbolos "{" y "}" no son obligatorios se recomienda mantener su uso.

Ciclo 1 - While

```
while(condition)
       actions;
```

Ciclo 2: Do-While

```
do
   actions;
while(condition)
```

Ciclo 3: For

```
for(initialization;condition;increment/decrement)
actions
```

Paso de parámetros a awk:

Awk permite que se inicialice el valor de una variable en la línea de comandos, cuando se invoca **awk**. Esto es, se permite pasar un valor a awk, que será almacenado en una variable de **awk**, en el momento de la invocación del mismo. Este valor puede ser un entero o una cadena. Para hacer esto se utiliza la siguiente sintaxis:

```
awk  -v  <variable>=<valor> ‘<Codigo AWK>’  <archivo-de-entrada>
```

Ejemplo: Programa que permite elegir el número de columnas que se desean imprimir.

```
[user@localhost home]$ cat datos
set-a              0.1945              0.9889              0.5444
set-b              0.5752              0.9445              0.9552
set-c              0.1684              0.0983              0.4060
set-d              0.8089              0.1956              0.1082
set-e              0.9621              0.5971              0.2042
set-f              0.5131              0.4257              0.2132
set-g              0.6182              0.1894              0.0910
set-h              0.2514              0.9625              0.0262
set-i              0.1182              0.5078              0.3755
set-j              0.8187              0.2381              0.0061

[user@localhost home]$
awk -v columnas=2 '{ for (ii=1; ii<= columnas; ++ii) printf "%s ",$ii ; printf "\n" }' datos
set-a 0.1945
set-b 0.5752
set-c 0.1684
set-d 0.8089
set-e 0.9621
set-f 0.5131
set-g 0.6182
set-h 0.2514
set-i 0.1182
set-j 0.8187


[user@localhost home]$
awk -v columnas=4 '{ for (ii=1; ii<= columnas; ++ii) printf "%s ",$ii ; printf "\n" }' datos
set-a 0.1945 0.9889 0.5444
set-b 0.5752 0.9445 0.9552
set-c 0.1684 0.0983 0.4060
set-d 0.8089 0.1956 0.1082
set-e 0.9621 0.5971 0.2042
set-f 0.5131 0.4257 0.2132
set-g 0.6182 0.1894 0.0910
set-h 0.2514 0.9625 0.0262
set-i 0.1182 0.5078 0.3755
set-j 0.8187 0.2381 0.0061
```

= = = = Laboratorio = = = =

- Algunas de las actividades a realizar en esta práctica se encuentran descritas en este documento, sus respuestas deben registrarse en :

- **Material de Apoyo**: Lab-03: AWK
- **Enlace al formulario:** Práctica 3: AWK .

Los siguientes archivos se utilizan en el formulario:

- Crea el directorio **awk_dir** y entra en él. En ese directorio:

Cree un archivo, ***numeros.dat*****,** con la siguiente información:

```
4       2
5       3
6       4
7       6
8       9
```

- Además, crea el archivo de nombre **novela** que contenga el siguiente texto:

Don Quijote de la Mancha, Cervantes

Capítulo II

Que trata de la notable pendencia [*] *que Sancho Panza tuvo con la sobrina y ama de don Quijote, con otros sujetos graciosos*

Cuenta la historia que las voces que oyeron don Quijote, el cura y el barbero eran de la sobrina y ama, que las daban diciendo a Sancho Panza, que pugnaba por entrar a ver a don Quijote, y ellas le defendían la puerta:

-¿Qué quiere este mostrenco en esta casa? Idos a la vuestra, hermano, que vos sois, y no otro, el que destrae y sonsaca a mi señor, y le lleva por esos andurriales.

A lo que Sancho respondió:

-Ama de Satanás, el sonsacado, y el destraído, y el llevado por esos andurriales soy yo, que no tu amo; él me llevó por esos mundos, y vosotras os engañáis en la mitad del justo precio: él me sacó de mi casa con engañifas, prometiéndome una ínsula, que hasta agora la espero.

-Malas ínsulas te ahoguen -respondió la sobrina-, Sancho maldito. Y ¿qué son ínsulas? ¿Es alguna cosa de comer, golosazo, comilón, que tú eres?

-No es de comer -replicó Sancho-, sino de gobernar y regir mejor que cuatro ciudades y que cuatro alcaldes de corte.

-Con todo eso -dijo el ama-, no entraréis acá, saco de maldades y costal de malicias. Id a gobernar vuestra casa y a labrar vuestros pegujares, y dejaos de pretender ínsulas ni ínsulos.

Grande gusto recebían el cura y el barbero de oír el coloquio de los tres; pero don Quijote, temeroso que Sancho se descosiese y desbuchase algún montón de maliciosas necedades, y tocase en puntos que no le estarían bien a su crédito, le llamó, y hizo a las dos que callasen y le dejasen entrar. Entró Sancho, y el cura y el barbero se despidieron de don Quijote, de cuya salud desesperaron, viendo cuán puesto estaba en sus desvariados pensamientos, y cuán embebido en la simplicidad de sus malandantes caballerías; y así, dijo el cura al barbero:

-Vos veréis, compadre, cómo, cuando menos lo pensemos, nuestro hidalgo sale otra vez a volar la ribera.

No pongo yo duda en eso -respondió el barbero-, pero no me maravillo tanto de la locura del caballero como de la simplicidad del escudero, que tan creído tiene aquello de la ínsula, que creo que no se lo sacarán del casco

cuantos desengaños pueden imaginarse.

-Dios los remedie -dijo el cura-, y estemos a la mira: veremos en lo que para esta máq0uina de disparates de tal caballero y de tal escudero, que parece que los forjaron a los dos en una mesma turquesa, y que las locuras del señor, sin las necedades del criado, no valían un ardite.

-Así es -dijo el barbero-, y holgara mucho saber qué tratarán ahora los dos.

-Yo seguro -respondió el cura- que la sobrina o el ama nos lo cuenta después, que no son de condición que dejarán de escucharlo.

En tanto, don Quijote se encerró con Sancho en su aposento; y, estando solos, le dijo:

-Mucho me pesa, Sancho, que hayas dicho y digas que yo fui el que te saqué de tus casillas, sabiendo que yo no me quedé en mis casas: juntos salimos, juntos fuimos y juntos peregrinamos; una misma fortuna y una misma suerte ha corrido por los dos: si a ti te mantearon una vez, a mí me han molido ciento, y esto es lo que te llevo de ventaja.

-Eso estaba puesto en razón -respondió Sancho-, porque, según vuestra merced dice, más anejas son a los caballeros andantes las desgracias que a sus escuderos.

-Engáñaste, Sancho -dijo don Quijote-; según aquello, quando caput dolet..., etcétera.

-No entiendo otra lengua que la mía -respondió Sancho.

-Quiero decir -dijo don Quijote- que, cuando la cabeza duele, todos los miembros duelen; y así, siendo yo tu amo y señor, soy tu cabeza, y tú mi parte, pues eres mi criado; y, por esta razón, el mal que a mí me toca, o tocare, a ti te ha de doler, y a mí el tuyo.

- Un tercer archivo a crear será **tiempos** con la siguiente información:

```
Columna:    "a"     "b"      "c"     "d"   "e"
gcc          10      1       12       2    19
frozenball   12      11       10      13   20
firefox       5      14       17       5    0
terminal      7     3         1     4   10
xilinx       18      20       6       12    12
```

El archivo **tiempos** muestra el tiempo de ejecución de diferentes programas. Cada columna de números representa los resultados de un conjunto de pruebas diferente, es decir la primer columna corresponde a la prueba “a”, la segunda al conjunto “b”, y así. .

Código ejemplo

Listado de correo

```
# Este es el mismo ejemplo que en la práctica anterior, donde cada línea de entrada puede ser
una dirección de correo itesm, o una matrícula con o sin la "A". Hecho en awk toma 5 líneas!!

# Recuerda que por cada línea, los patrones se examinan en secuencia. "next" ignora los
patrones que siguen.

/A[0-9]+@itesm.mx/ {print $1; next}
```

```
/A[0-9]+/          {print $1 "@itesm.mx"; next} #ojo dos strings juntos se concatenan
/[0-9]+/           {if (length($1) == 6) {print "A00" $1 "@itesm.mx"} \
                    else                    {print "A0" $1 "@itesm.mx"}; next}

                   {print $1 " wrong format"}
```