Métodos de gran escala

Adolfo De Unánue T.

Instituto Tecnológico Autónomo de México

nanounanue@gmail.com

30 de enero de 2013

Notas sobre el curso

- Tareas: 40 %
- Proyecto Final: 60 %

Usando `git`: ¿Por qué Control de versiones?

- Sincronizar cambios con tus compañeros de trabajo.
- Respaldos regulares (muy regulares) de tu trabajo.

  Enviar correros electrónicos no está bien, no escala, no es *cool*, punto.

- *Configuration Management* $\rightarrow$ líneas de desarrollo en paralelo.

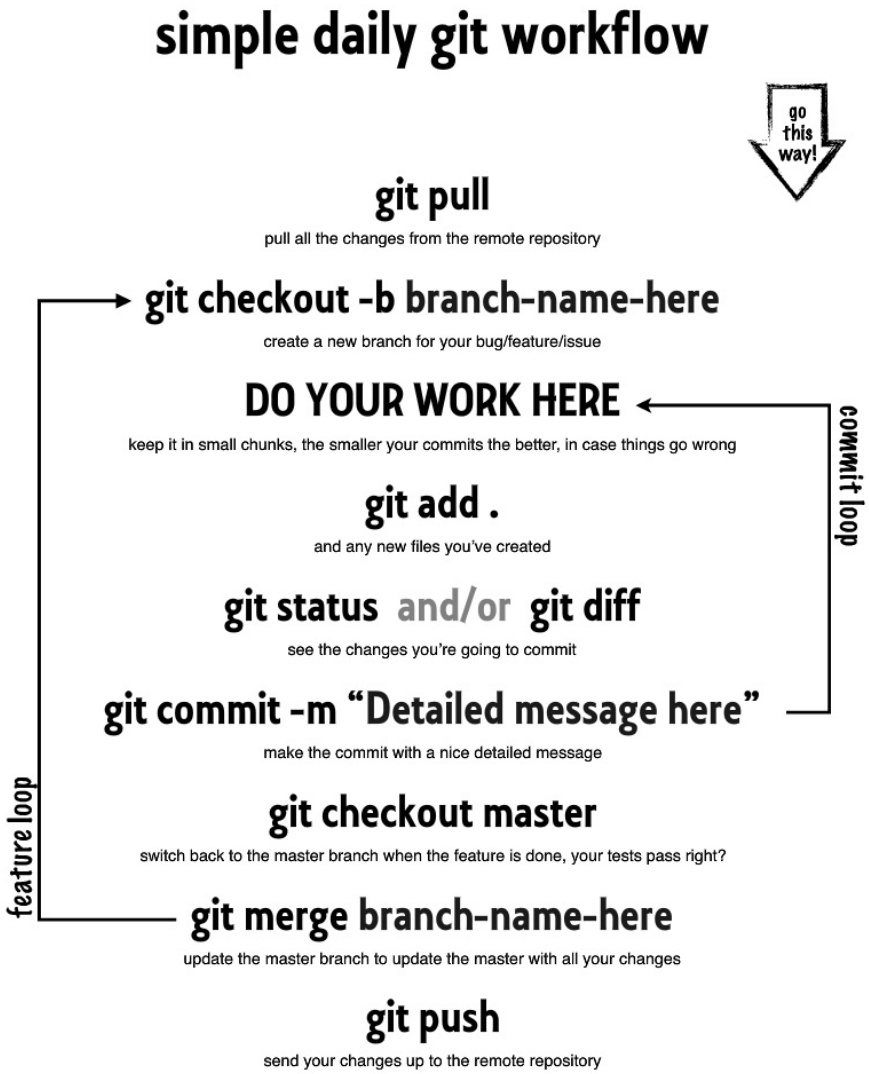
  Eso no lo veremos en este curso, pero vale la pena averiguar.

Usando `git`: Filosofía

Michael Crichton

Los mejores libros no se escriben– se *reescriben*.

Usando git: Flujo



http://nakedstartup.com/2010/04/simple-daily-git-workflow

Usando git: Configuración básica

```
$ git config --global user.name "Nombre Apellido"
$ git config --global user.email "correo@buzon.com"
```

Usando git: Ejemplo

Vamos a crear sus carpetas personales en `github`, crearemos una carpeta `test`, dos archivos, modificaremos esos archivos y luego incorporaremos todo a la rama *master*.

```
$ git clone https://github.com/nanounanue/itam-big-data.git
$ cd itam-big-data
$ git branch   # veo que ramas hay
$ git checkout -b ejemplo-username # Creo una rama para trabajar
$ touch uno.txt
$ touch dos.txt
$ git add .
$ git status
$ git commit -m "Archivos iniciales"
$ echo "Hola" > uno.txt
$ echo "Adios" > dos.txt
$ git diff uno.txt
$ git status
$ git add uno.txt
$ git commit -m "Agregamos un hola"
$ git status
$ git add dos.txt
$ git commit -m "Agregamos un adios"
$ git status
$ git checkout master
$ git merge ejemplo-username      # Uno la rama con master
$ git push   # NO EJECUTAR
$ git branch -d ejemplo-username # destruye la rama
$ git branch
```

pandoc: Introducción

- Es una extensión de `markdown`.
- Página principal: `http://johnmacfarlane.net/pandoc/`
- `pandoc` es una herramienta que permite la transformación de documentos de texto. E.g.
  - LaTeX$\rightarrow$ `markdown` y viceversa.
  - PDF.
  - HTML5.
  - Beamer
  - HTML $\rightarrow$ `markdown` y vicerversa.
  - Presentaciones web.

pandoc: Instalación

- **GNU/Linux** (Debian-like)

```
sudo apt-get install pandoc
```

- **Windows** y **MacOS** Se requiere ir a la página de pandoc y descargar la versión correspondiente. Además será necesario descargar LaTeX. (ver instrucciones en la página)

pandoc: Ejecución

```
pandoc ejemplo.markdown -s --highlight-style haddock -o ejemplo.pdf
```

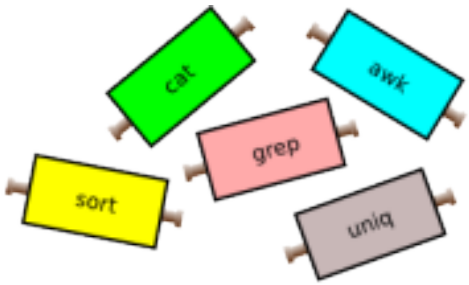
Unix tools para big data: Outline

1. Herramientas básicas de UNIX
   - wc
   - head, tail
   - split, cat
   - cut
   - uniq y sort
   - Expresiones regulares
   - grep
   - awk
   - sed
   - Otros
   - Bash programming
2. Tarea 2

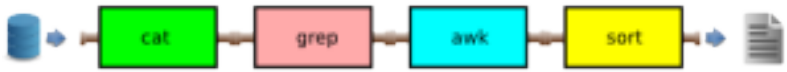
Introducción

En muchas ocasiones, se verán en la necesidad de responder muy rápido y en una etapa muy temprana del proceso de big data. Las peticiones regularmente serán cosas muy sencillas, como estadistica univariable y es aquí donde es posible responder con las herramientas *mágicas* de UNIX.

Lego: Idea poderosa

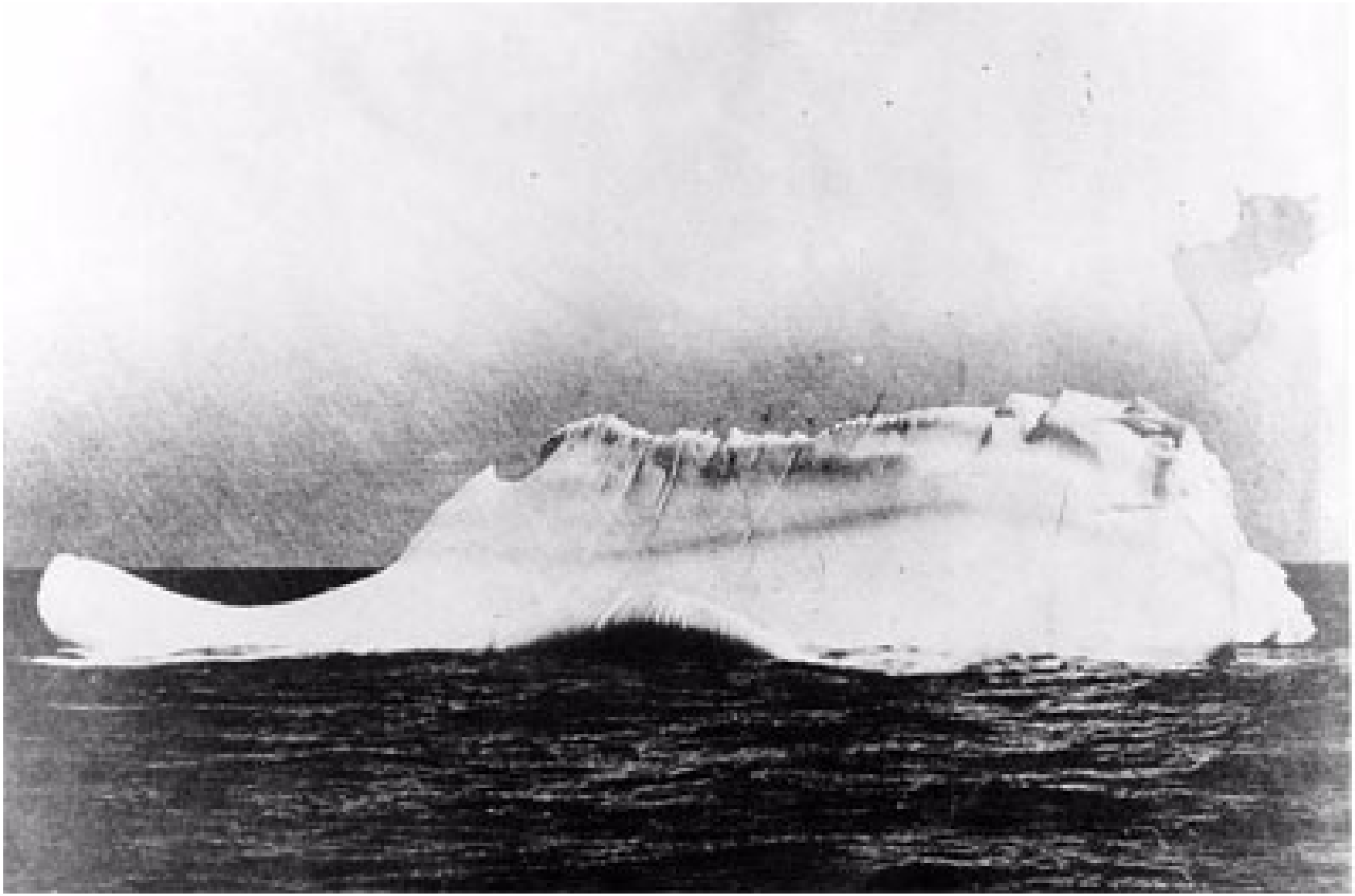


Programs as blocks



Processing pipeline

Datos para jugar



Datos para jugar

- Para los siguientes ejemplos trabajaremos con los archivos encontrados en `data`.
- Estos datos representan la lista de pasajeros del Titanic y si sobrevivieron o no.
- Se incluyen dos archivos de datos `train.csv` y `test.csv`.
- La descripción de las variables está en el archivo `descripcion.txt`.

wc

- `wc` significa *word count* y hace más que eso, puede contar palabras, renglones, bytes, caracteres, etc.
- Es un buen momento para aprender que existe el manual

  `man wc`
- En nuestro caso nos interesa la bandera `-l`, la cual sirve para contar líneas.

```
> cd data
> wc -l train.csv
  892 train.csv
> wc -l *.csv
  419 test.csv
  892 train.csv
 1311 total
```

`head, tail`

- `head` y `tail` sirven para explorar visualmente las primeras diez (default) o últimas diez (default) renglones del archivo, respectivamente.

head, tail

```
> cd data
> head train.csv
survived,pclass,name,sex,age,sibsp,parch,ticket,fare,cabin,embarked
0,3,"Braund, Mr. Owen Harris",male,22,1,0,A/5 21171,7.25,,S
1,1,"Cumings, Mrs. John Bradley (Florence Briggs Thayer)",female,38,1,0,PC 17599,71.2833,C85,C
1,3,"Heikkinen, Miss. Laina",female,26,0,0,STON/O2. 3101282,7.925,,S
1,1,"Futrelle, Mrs. Jacques Heath (Lily May Peel)",female,35,1,0,113803,53.1,C123,S
0,3,"Allen, Mr. William Henry",male,35,0,0,373450,8.05,,S
0,3,"Moran, Mr. James",male,,0,0,330877,8.4583,,Q
0,1,"McCarthy, Mr. Timothy J",male,54,0,0,17463,51.8625,E46,S
0,3,"Palsson, Master. Gosta Leonard",male,2,3,1,349909,21.075,,S
1,3,"Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)",female,27,0,2,347742,11.1333,,S
> tail -3 train.csv
0,3,"Johnston, Miss. Catherine Helen ""Carrie""",female,,1,2,W./C. 6607,23.45,,S
1,1,"Behr, Mr. Karl Howell",male,26,0,0,111369,30,C148,C
0,3,"Dooley, Mr. Patrick",male,32,0,0,370376,7.75,,Q
```

split, cat

- `cat` concatena archivos y/o imprime al `stdout`

```
> echo 'Hola mundo!' >> test
> echo 'Adios mundo cruel!' >> test
> cat test
...
> rm test
> cd data
> cat train.csv test.csv > titanic.csv
> wc -l *.csv
```

- `split` hace la función contraria, divide archivos.
- `split` puede hacer por tamaño (bytes) (`-b`) por líneas (`-l`)

```
> split -l 500 titanic.csv titanic_
> wc -l titanic*
```

- PREGUNTA: ¿Cómo pegaríamos los archivos?

cut

- Con `cut` podemos dividir el archivo pero por columnas.
- Donde columnas puede estar definido como campo (`-f`), en conjunción con (`-d`), carácter (texttt-c) o bytes (`-b`).
- En este curso nos interesa partir por campo.

```
$ echo "Adolfo|1978|Físico" >> prueba.psv  # Por razones que serán obvias
                                          # creemos un nuevo archivo
$ echo "Patty|1984|Abogada" >> prueba.psv  # psv = Pipe separated values
$ cut -d'|' -f1 prueba.psv
$ cut -d'|' -f1,3 prueba.psv
$ cut -d'|' -f1-3 prueba.psv
```

Pregunta: ¿Qué pasa con los datos de Titanic? Por ejemplo: Quisiera las columnas 2, 4, 6 ó si quiero las columnas de sexo, supervivencia, edad y clase.

cut y nuestro primer problema de datos

```
> head train.csv | cut -d',' -f2,4,6
pclass,sex,sibsp
3, Mr. Owen Harris",22
1, Mrs. John Bradley (Florence Briggs Thayer)",38
3, Miss. Laina",26
1, Mrs. Jacques Heath (Lily May Peel)",35
3, Mr. William Henry",35
3, Mr. James",
1, Mr. Timothy J",54
3, Master. Gosta Leonard",2
3, Mrs. Oscar W (Elisabeth Vilhelmina Berg)",27
> head train.csv | cut -d',' -f4,1,5,2
survived,pclass,sex,age
0,3, Mr. Owen Harris",male
1,1, Mrs. John Bradley (Florence Briggs Thayer)",female
1,3, Miss. Laina",female
1,1, Mrs. Jacques Heath (Lily May Peel)",female
0,3, Mr. William Henry",male
0,3, Mr. James",male
0,1, Mr. Timothy J",male
0,3, Master. Gosta Leonard",male
1,3, Mrs. Oscar W (Elisabeth Vilhelmina Berg)",female
```

¿Ven el problema? Para resolver esto, necesitamos comando más poderosos... pero antes dos comando más :)

uniq, sort

- `uniq` Identifica aquellos renglones *consecutivos* que son iguales.
- `uniq` puede contar (`-c`), eliminar (`-u`), imprimir sólo las duplicadas (`-d`), etc.
- `sort` Ordena el archivo, es muy poderoso, puede ordenar por columnas (`-k`), usar ordenamiento numérico (`-g, -h, -n`), mes (`-M`), random (`-r`) etc.

```
sort -t "," -k 2 -k 1 -n train.csv
```

- Combinados podemos tener un group by:

```
cat train.csv | cut -d, -f1,2 | sort -t "," -k 2 -k 1 -n | uniq -c
```

Regexp

Broma

Q: What did one regular expression say to the other?

A: .*

Regexp

Quote

In computing, regular expressions provide a concise and flexible means for identifying strings of text of interest, such as particular characters, words, or patterns of characters. Regular expressions (abbreviated as regex or regexp, with plural forms regexes, regexps, or regexen) are written in a formal language that can be interpreted by a regular expression processor, a program that either serves as a parser generator or examines text and identifies parts that match the provided specification.

Wikipedia: Regular Expressions

Regexp: Básicos

- Hay varios tipos POSIX, Perl, PHP, GNU/Emacs, etc.

  Aquí veremos POSIX
- Pensar en patrones (*patterns*).
- Operadores básicos
  - OR, `gato|gata` hará *match* con gato o gata.
  - Agrupamiento o precedencia de operadores, `gat(a|o)` tiene el mismo significado que `gato|gata`.
  - Cuantificadores, `?` 0 o 1, `+` uno o más, `*` cero o más.
- Expresiones básicas
  - `.` Cualquier carácter.
  - `[ ]` Cualquier carácter incluido en los corchetes, e.g. `[xyz]`, `[a-zA-Z0-9-]`.
  - `[^ ]` Cualquier carácter individual que no esté en los corchetes, e.g. `[^abc]`. También puede indicar inicio de línea (fuera de los corchetes.)
  - `\( \)` ó `( )` crea una subexpresión que luego puede ser invocada con `\n` donde `n` es el número de la subexpresión.
  - `{m,n}` Repite lo anterior un número de al menos `m` veces pero no mayor a `n` veces.
  - `\b` representa el límite de palabra

Regexp: Ejemplos y ejercicios

- username: `[a-z0-9_-]{3,16}`
- contraseña: `[a-z0-9_-]{6,18}`
- IP address: `(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)3(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)`
- fecha (`dd/mm/yyyy`): ???
- email (`adolfo@itam.edu`) : ???
- URL (`http://gmail.com`): ???

¿Y cómo reconocemos el nombre en la fuente de datos de titanic?

Expresiones de caracteres

- `[:digit:]` Dígitos del 0 al 9.
- `[:alnum:]` Cualquier carácter alfanumérico 0 al 9 OR A a la Z OR a a la z.
- `[:alpha:]` Carácter alfabético A a la Z OR a a la z.
- `[:blank:]` Espacio ó TAB únicamente.

Regexp

Espero que ahora si hayan entendido la broma... :)

grep

grep nos permite buscar las líneas que contengan un patrón en específico.

```
> grep "Mrs\." train.csv  # ¿Cuántas señoras hay?
> grep "Miss\." train.csv # ¿Cuántas señoritas había abordo?
> grep "Mr\." train.csv # ¿Hombres?
> grep -v "Mr\." train.csv # ¿Mujeres?
                           #  ¿Coincide si lo hacemos por sexo?¿Por qué?
> grep -E "John|James" train.csv  # Mmmm traemos de más
> grep -E "John|James" train.csv | grep -v -E "Johnston|Johnson"
> grep -c -o -E "John|James" train.csv
> grep -o -E "John|James" train.csv | sort | uniq -c
> grep -o -E "\bJohn\b|\bJames\b" train.csv | sort | uniq -c #Delimitador de palabra
> grep  "^[0-9]\{1,5\}$" numbers.txt # 1 o 6
> grep  "^[0-9]\{5\}$" numbers.txt  # Exactamente 5
> grep "[0-9]\{5,\}" numbers.txt  # Más de 5
> grep "\(aeiou]\).\1" names.txt # Vocal caracter Misma vocal
```

awk

- awk es un lenguaje de programación muy completo, orientado a archivos de texto que vengan en columnas.
- Un programa de awk consiste en una secuencia de enunciados del tipo patrón-acción: `pattern { action statement }`.
- Si hay varios se separan con ‘‘;’’ `[pattern] [command1] ; [command2] ; [command3]`
- `BEGIN , END, [op1] ~ [regular expression]`, operadores booleanos como en C.
- Variables especiales:
  - `$1, $2, $3, ...` – Valores de las columnas
  - `$0` – toda la línea
  - `FS` – separador de entrada data.
  - `OFS` – separador de salida
  - `NR` – número de la línea actual
  - `NF` – número de campos en la línea (record)

awk: sintaxis

Sintaxis

```
awk '/search pattern1/ {Actions}
     /search pattern2/ {Actions}' file
```

awk: Ejemplos

```
> awk 'END { print NR }'  train.csv # Lo mismo que wc -l
> awk 'BEGIN{ FS = "|" }; { if(NF != 22){ print >> "train_fixme.csv"} \
else { print >> "train_fixed.csv" } }' train.csv
# Limpia el archivo con columnas de más
> awk '{ print NF ":" $0 }' train.csv
> awk '{ $2 = ""; print }'
> awk 'NF > 4' train.csv # Imprime las líneas que tengan más de cuatro campos
> awk 'BEGIN{ FS = "," }; {sum += $2} END {print sum}' train.csv
# Suma de una columna (aunque aquí no tiene mucho sentido)
> awk '{sum7+=$7; sum8+=$8;mul+=$7*$8} END {print sum7/NR,sum8/NR,mul/NR}' train.csv
# Promedios de varias columnas
> awk '/Beth/ { n++ }; END { print n+0 }' train.csv
> awk '$1 > max { max=$1; maxline=$0 }; END { print max, maxline }'train.csv
# Imprime el máximo del campo $1
> awk '{ sub(/foo/,"bar"); print }' train.csv
> awk '{ gsub(/foo/,"bar"); print }' train.csv
> awk '/baz/ { gsub(/foo/, "bar") }; { print }' train.csv
> awk '!/baz/ { gsub(/foo/, "bar") }; { print }' train.csv
> awk 'a != $0; { a = $0 }' # Como uniq
> awk '!a[$0]++' # Remueve duplicados que no sean consecutivos
> awk '$4 ~/Technology/' employee.txt
> awk 'BEGIN { count=0;}
$3 ~ /John/ { count++; }
END { print "Número de Johns en el Titanic =",count;}' train.csv
```

awk: más sintaxis

Sintaxis

```
awk '
BEGIN { Actions}
{ACTION} # Action for everyline in a file
END { Actions }
# is for comments in Awk
' file
```

Algo un poco más poderoso...

mean, max, min

```
> awk '{FS="|"}{if(min==""){min=max=$1}; if($1>max) {max=$1};if($1<min) {min=$1}; total+=$1; count+=1} END \
{print "mean = " total/count,"min = " min, "max = " max}' data.txt
```

Algo un poco más poderoso...

Mediana

```
$ gawk -v max=128 '
    function median(c,v,  j) {
       asort(v,j);
       if (c % 2) return j[(c+1)/2];
       else return (j[c/2+1]+j[c/2])/2.0;
    }

    {
       count++;
       values[count]=$1;
       if (count >= max) {
         print  median(count,values); count=0;
       }
    }

    END {
       print  "median = " median(count,values);
    }' data.txt
```

Algo un poco más poderoso...

Desviación estándar

```
> awk '{sum+=$1; sumsq+=$1*$1;} END {print "stdev = " sqrt(sumsq/NR - (sum/NR)**2);}'  data.txt
```

Algo un poco más poderoso...

Todo muy bien, pero ¿Qué pasa cuando hay varias columnas?

```
> awk -F "|" '
{ for (i=1; i<=NF; ++i) sum[i] += $i; j=NF }
END { for (i=1; i <= j; ++i) printf "%s ", sum[i]; printf "\n"; }
' data2.txt
```

EJERCICIO: Modificar los ejemplos anteriores para calcular la estadística de `data2.txt`

sed

- `sed` significa *stream editor*. Permite editar archivos de manera automática.
- El comando `sed` tiene cuatro espacios:
  - Flujo de entrada
  - Patrón
  - Búfer
  - Flujo de salida
- Entonces, `sed` lee el *flujo de entrada* hasta que encuentra `\n`. Lo copia al *espacio patrón*, y es ahí donde se realizan las operaciones con los datos. El *búfer* está para su uso, pero es opcional, es un búfer, vamos. Y finalmente copia al *flujo de salida*.

Algunos ejemplos

Ejemplos

```
> sed 's/foo/bar/' data3.txt   # Sustituye foo por bar
> sed -n 's/foo/bar/' data3.txt # Lo mismo pero no imprime a la salida
> sed -n 's/foo/bar/; p' data3.txt # Lo mismo pero el comando "p", imprime
> sed -n 's/foo/bar/' -e p data3.txt # Si no queremos separar por espacios
> sed '3s/foo/bar/'  data3.txt # Sólo la tercera linea
> sed '3!s/foo/bar/'  data3.txt # Excluye la tercera línea
> sed '2,3s/foo/bar/' data3.txt # Con rango
> sed -n '2,3p' data3.txt  # Imprime sólo las líneas de la 2 a la 3
> sed -n '$p' # Imprime la última línea
> sed '/abc/,/-foo-/d' data3.txt # Elimina todas las líneas entre "abc" y "-foo-"
> sed '/123/s/foo/bar/g'  data3.txt
# Sustituye globalmente "foo" por "bar" en las líneas que tengan 123
> sed 1d data2.txt # Elimina la primera línea del archivo
> sed -i 1d data2.txt  # Elimina la primera línea del archivo de manera interactiva
```

Otros muy importantes

- `file -i` Provee información sobre el archivo en cuestion

```
> file -i train.csv
train.csv: text/plain; charset=us-ascii
```

- `iconv` Convierte entre encodings, charsets etc.

```
> iconv -f iso-8859-1 -t utf-8 train.csv > train_utf8.csv
```

- `od` Muestra el archivo en octal y otros formatos, en particular la bandera `-bc` lo muestra en octal seguido con su representación ascii. Esto sirve para identificar separadores raros.

```
> od -bc train.csv  | head
0000000 163 165 162 166 151 166 145 144 054 160 143 154 141 163 163 054
          s   u   r   v   i   v   e   d   ,   p   c   l   a   s   s   ,
0000020 156 141 155 145 054 163 145 170 054 141 147 145 054 163 151 142
          n   a   m   e   ,   s   e   x   ,   a   g   e   ,   s   i   b
0000040 163 160 054 160 141 162 143 150 054 164 151 143 153 145 164 054
          s   p   ,   p   a   r   c   h   ,   t   i   c   k   e   t   ,
```

Otros muy importantes

- `find` Búsquedas poderosas de archivos y directorios (por tiempo, acceso, tamaño, nombre, etc.)

  Excelentes ejemplos de uso:

  `http://www.thegeekstuff.com/2009/03/15-practical-linux-find-command-examples/`

- `xargs` Maneja los argumentos para comandos. Se usa principalmente con `find`

  Ver aquí: `http://www.cyberciti.biz/faq/linux-unix-bsd-xargs-construct-argument-lists-utility/`

- `parallel` Ejecuta comandos en paralelo. Es una versión mejorada de `xargs` también.
  NOTA: Necesita instalarse

  `http://www.gnu.org/software/parallel/`

Bash programming

- Hasta ahora hemos visto *one-liners*, pero ...
- ¿Qué pasa si quiero hacerlo en varios archivos?
- ¿Y si quiero usar varios comandos?

Bash programming: Redireccionando |, >, >>, &&

- '|' (pipe) **"Entuba"** la salida de un comando al siguiente (no se puede con todos los comandos, pero con la mayoría sí)
- '>', '>>', **Redirecciona** la salida de los comandos a un *sumidero*.

```
ls >> prueba.dat
```

- '<' **Redirecciona** desde el archivo

```
sort < prueba.dat  # A la línea de comandos acomoda con sort,
sort < prueba.dat > prueba_sort.dat # Guardar el sort a un archivo.
```

- '&&' es un AND, sólo ejecuta el comando que sigue a '&&' si el primero es exitoso.

```
> ls && echo "Hola"
> lss && echo "Hola"
```

Bash programming: *loops*

```
for var in ‘comando‘
do
instrucción
instrucción
...
done
```

Bash programming: condicionales

```
if TEST-COMMANDS; then CONSEQUENT-COMMANDS; fi
```

Ver esta página: `http://pinehead.tv/linux/conditions-in-bash-scripting-if-statements/`

Bash programming: Shebang #!

- Al final hay que poner esto en un archivo, ponerlo a correr e irnos a pensar...
- Para cualquier programa *script* es importante que la primera línea del archivo le diga a bash que comando usar para ejecutarlo.
- Esta linea se conoce como `shebang` y se representa por `#!` seguido de la ruta al ejecutable.

Tarea

1. Generar un reporte dentro de la carpeta `tarea_2` sobre su fuente de datos seleccionada.
2. Descripción de los campos (esto era de la tarea pasada).
3. Debe de incluir estadisticas simples como conteos (para variables categóricas), promedios, máximos, mínimos, faltantes por columna.
4. Y muestren su curiosidad, cualquier cosa interesante de los datos.

Fin