Parámetros pasados a un script

```
$0  Contiene el nombre del script tal como es invocado
$*  El conjunto de todos los parámetros en un solo argumento
$@  El conjunto de argumentos, un argumento por parámetro
$#  El número de parámetros pasados al script
$?  El código de retorno del último comando
$$  El PID del shell que ejecuta el script
$!  El PID del último proceso ejecutado en segundo plano
```

```
basename $0 devuelve el nombre del script
dirname $0  devuelve el nombre del directorio
```

Mostrar todos los parámetros pasados a un script:

```
#!/bin/bash
for i in $@
do
  echo $i
done
```

O metiéndolos en un vector:

```
#!/bin/bash
argv=("$@")
let narg=("$#")-1
for i in `seq 0 $narg`
do
  echo ${argv[$i]}
done
```

Coger opciones de parámetros pasados a un script:

http://stackoverflow.com/questions/192249/how-do-i-parse-command-line-arguments-in-bash

https://github.com/oracle/docker-images/blob/master/OracleWebLogic/dockerfiles/buildDockerImage.sh

Leer variables

Vector

Se define un vector

```
array=( hola adios )
```

Tambien se puede definir:

```
variable=$(echo hola adios)
array=( $variable )
```

Se muestra un número de elemento en concreto:

```
# echo ${array[0]}
hola
```

echo ${array[1]}

```
adios
```

Número de elementos:

```
# echo ${#array[@]}
2
```

Todos los elementos:

```
# echo ${array[@]}
hola adios
```

Método dos

array[0]='cero'

echo ${array[0]}

Variable en el nombre de un vector

Funciona incluso con elementos con espacios

```
#!/bin/bash
var=Error
vector_Error=( "hola que tal" "como estamos" "bastante bien" )

eval name=(\"\${vector_$var[@]}\")

echo ${name[0]}
hola que tal

echo ${name[1]}
como estamos

echo ${name[2]}
```

```
bastante bien
```

Duplicar vector

```
prueba=(
'hola que tal como estamos'
'pues bastante bien'
'¿para la edad que tienes?'
)

Funciona:
nuevo=("${prueba[@]}")

echo ${prueba[1]}
pues bastante bien

echo ${nuevo[1]}
pues bastante bien
```

Convertir linea en vector con un separador

Si tenemos la linea:

```
linea="hola;que tal;como estamos;bien"
```

Y queremos pasar a un vector cortando por el caracter “;” y meterlo en el vector “array”

```
IFS=';' read -a array <<< "$linea"
```

Lo comprobamos recorriendo el vector:

```
let n=${#array[@]}-1
for i in `seq 0 $n`
do
  echo ${array[$i]}
done
```

Otra forma de recorrer el vector es esta. Muy importante las **“** por que si no coge mal los valores con espacios:

```
for i in "${array[@]}"; do echo $i; done
```

El resultado es:

```
hola
que tal
como estamos
```

```
bien
```

while read linea

```
while read linea
do
 texto=$texto" "$linea
done < <(cat fichero)
echo $texto
```

read dentro de bucle

```
while read linea
do
  read input </dev/tty
done < <(cat fichero)
```

Asignar Variable

Asignar variable a un nombre de variable:

```
#!/bin/bash
var=variable2
variable2="hola"
eval echo \$${var}
#o lo que es lo mismo
echo $variable2
```

Asignar variable a la misma variable con nombre variable :P

```
#!/bin/bash
var=variable2
for i in 3 2 1
do
  eval $var='`echo ${!var}`" "$i'
done
eval echo \$${var}
echo ${!var}
echo $variable2
```

Bucle

if

```
if [ $var -eq 1 ]
then
  echo "var=1"
elif [ $var -eq 2 ]
then
  echo "var=2"
else
  echo "var no vale 1 ni 2"
fi
```

for

```
for ((i=1;i<=9;i+=1)); do echo $i; done
for i in {1..9}; do echo $i; done
for i in `seq 1 9`; do echo $i; done
```

case

```
case $var in
  0 ) echo "Salir del menu"; exit;;
  1 ) echo "Opcion 1";;
  2 ) echo "Opcion 2" ;;
  * ) echo "Opción incorrecta" ;;
esac
```

Comprobar si un puerto está abierto o levantado:

En local:

```
#!/bin/sh
while true
  do
  if ! netstat -an | grep 78889 > /dev/null
  then
```

```
    echo `date` >> /var/log/port8889.log
  fi
done

  lsof -Pni:[puerto]

  netstat -putan | grep [puerto]
```

Servidor externo

Como alternativa a telnet se pueden usar estos comandos. A veces hay que poner el nombre del servicio, por ejemplo una VIP de oracle

```
cat < /dev/null > /dev/tcp/<ip>/<puerto>
```

Conexión correcta, no da error:

```
# cat < /dev/null > /dev/tcp/oracle-scan/1521
# echo $?
0
```

Conexión fallida, devuelve error:

```
# cat < /dev/null > /dev/tcp/oracle-scan/1521
-bash: connect: Conexión rehusada
-bash: /dev/tcp/oracle-scan/1521: Conexión rehusada
```

Lo mismo para

```
echo > /dev/tcp/<ip>/<puerto>
```

Conexión correcta, no da error:

```
# echo > /dev/tcp/oracle-scan/1521
# echo $?
0
```

Conexión fallida, devuelve error:

```
# echo > /dev/tcp/oracle-scan/15212
-bash: connect: Conexión rehusada
-bash: /dev/tcp/oracle-scan/15212: Conexión rehusada
```

Con netcat, la respuesta es parecida al telnet:

```
nc oracle-scan 1521
```

Con curl:

```
curl -v telnet://<ip>:<puerto>
```

en un script con lista y colores:

```
#!/bin/bash

ecco() {
    local text="$1"
    local color="$2"
    case "$color" in
        "rojo") echo -e "\e[31m$text\e[0m";;
        "verde") echo -e "\e[32m$text\e[0m";;
        "azul") echo -e "\e[34m$text\e[0m";;
        *) echo "$text";; # Color por defecto si no se proporciona uno
válido
    esac
}

azul="\e[34mOK\e[0m"
verde="\e[32mOK\e[0m"
rojo="\e[31mFAIL\e[0m"

# Definir la lista de microservicios y puertos
microservicios=(
wzc-authentication 16443
features-matrix 14443
wzc-access-rules 35443
wzc-authorization 26443
wzc-card 24443
wzc-card-payment 28443
wzc-card-reward 31443
wzc-card-transaction 29443
wzc-client-profile 17443
wzc-client-rights 20443
wzc-device 21443
wzc-document 19443
wzc-message 18443
wzc-notification 23443
wzc-registration 25443
)
echo
for ((i=0; i<${#microservicios[@]}; i+=2)); do
    echo "**************************"
    line=("${microservicios[@]:i:2}")
    microservicio="${line[0]}"
    puerto="${line[1]}"
    echo -n "$microservicio $puerto "
    (>/dev/tcp/localhost/$puerto) 2>/dev/null && ecco OK azul || ecco FAIL
rojo
done
```

```
echo "**************************"
```

Demonio

Se crea un fichero kkfichero que mientras este creado se ejecuta el script.
Se ponen las dos condiciones, la que queremos i el fichero para poder parar el script (en este ejemplo contamos hasta 10)

```
#!/bin/sh
touch kkfichero
i=0

while [ $i -le 10 ] && [ -f kkfichero ]
do
  echo $i
    i=$(($i + 1))
  sleep 3
done

if [ -f kkvolei ]
then
  rm kkvolei
fi
```

Para que el fichero sea único se puede crear con el PID:

```
#!/bin/sh
PID=`echo $$`
```

Realiza comprobación

```
test `date +\%w` = 3 && echo "Hoy es miércoles"
```

```
`comando`; [ $? -eq 0 ] && echo "Comando correcto"
```

Leer fichero:

```
cat fichero.txt | while read linea
do
  echo "LEIDO ($linea)"
done
```

Ver si URL existe

Opción 1

```
#!/usr/bin/bash.exe

listaUrls=`cat<<EOF
http://www.google.com/index.html

http://www.voleicat.net/generic/documentsweb0607/resultats/campionats%20catalunya/102/cl_102_16.htm

EOF`
for url in $listaUrls
do
  tot=`wget $url 2>&1| grep -ic 'Not Found'`

  if [ $tot -ne 0 ]
  then
    echo "$url NO EXISTE"
  else
    echo "$url SI EXISTE"
  fi
done
```

Opción 2

```
curl -w %{http_code}   -s  -o fichero http://www.google.com
```

Devuelve el código 404,300, 200, etc....
Parámetros:
-s: silent Para no mostrar el progreso de descarga
-o fichero: Para descargar la salida en un fichero. El fichero puede ser /dev/null

Conocer mi ip

```
miip=`curl -s http://checkip.dyndns.org | awk '{print $6}'  | cut -d "<" -f1`
```

ls i espacio en disco

Muestra lo que ocupan unos ficheros por fecha

```
#!/bin/sh
ultima=`du -a --time | sort -k 2 | head -1 | awk '{print $2}' | cut -b -7`
let total=0
let sum=0
#du -a --time | sort -k 2 | while read linea
IFS=$'\n'
for linea in `du -a --time | sort -k 2`
do
  fecha=`echo $linea | awk '{print $2}' | cut -b -7`
  let sum=`echo $linea | awk  '{print $1}'`
  if [ $ultima == $fecha ]
  then
    let total=total+sum
  else
    echo $ultima $total
    let total=$sum
  fi
  ultima=$fecha
done
echo $ultima $total
```

Muestra lo que ocupan los directorios

```
# du --max-depth=1
```

Búsqueda de ficheros

Ficheros modificados hace mas de 7 días

```
find * -mtime +7 -exec ls -la {} \;
```

Ficheros modificados en los últimos 7 dias

```
find * -mtime -7 -exec ls -la {} \;
```

Mover los ficheros

```
find * -mtime +7 -maxdepth 0 -type f -exec mv '{}' old/ ';'
```

Operaciones

Suma

En bash:

```
let i=$i+1
```

```
expr 1 + 1
```

En sh:

```
i=$(($i + 1))
```

Con decimales:

```
 awk 'BEGIN{printf "%.2f\n", (640/480)}'
 1.33
```

Búcles

comparaciones

```
  -eq is equal to 5 == 6
  -ne is not equal to 5 != 6
  -lt is less than 5 < 6
  -le is less than or equal to 5 <= 6
  -gt is greater than 5 > 6
  -ge is greater than or equal to 5 >= 6
  -f es un archivo (existe el fichero)
  -d es un directorio
```

Para la comparación entre cadenas se usara los siguientes simbolos:

```
  == Realiza la comparación entre cadenas si son iguales
  != Decide si son distintas
  -n Informa si la cadena tiene longitud mayor a cero
  -z Informa si la cadena tiene longitud igual a cero
```

ECHO

Referencia:
https://www.linuxtopia.org/online_books/advanced_bash_scripting_guide/string-manipulation.html

Mostrar una parte de una cadena con echo:

```
palabra=murcielago
```

```
echo ${palabra:2}
rcielago
```

```
echo ${palabra:1:2}
ur
```

```
echo ${palabra:(-2)}
go
```

```
echo ${palabra:2:3}
rci
```

SED

Muestra las coincidencias en un fichero

```
sed  -n '/texto/p' totes_brutes
```

Borra una línea. La número 5

```
sed '5d' fichero.txt
```

Borra líneas con coincidencias

```
sed  -i '/texto/d' totes_brutes
```

Elmina caracteres duplicados en un fichero

```
molon
remar
adios

sed -f <(printf 's/%s//2g\n' {a..z}) <<< cat fichero.txt

moln
rema
```

```
adios
```

No es sed, pero para reemplazar dentro de una variable

```
var="hola que tal"
echo $var
  hola que tal
```

Solo cambia el primero:

```
echo ${var/ /%20}
  hola%20que tal
```

Los cambia todos:

```
echo ${var// /%20}
  hola%20que%20tal
```

Mostrar un parámetro

```
echo "param1=hola param2=que param3=tal" | sed -e 's/^.*param1=\([^
]*\).*$/\1/'
```

Mostrar una parte de un fichero

```
sed -n -e '/<pattern start>/,/<pattern end>/p' <file>
```

Si no ponemos <pattern end> nos muestra hasta el final.
Para mostrar hasta el final:

```
tail -n +`grep  "<patern>" messages -n | head -1 | awk -F: {'print $1'}`
<file>
```

Para mostrar determinadas líneas

Mostrar la segunda línea

```
sed -n '2p' file.txt
```

Mostrar a partir de la segunda línea

```
sed 2p file.txt
```

Mostrar hasta la línea 20:

```
 sed 20q file.txt
```

Mostrar de la 10 a la 33:

```
sed -n '10,33p' file.txt
```

Mostrar de la 10 a la 33, pero mas rápido por si el fichero es muy grande. En la q le dices cuantas líneas quieres que te muestre y luego sobre esas 34 cogemos de a 10 a la 33:

```
sed -n '34q;10,33p' file.txt
```

La linea 15 y la 20

```
sed -n '15p;20p' file.txt
```

Referencias:
http://stackoverflow.com/questions/6022384/bash-tool-to-get-nth-line-from-a-file
http://sed.sourceforge.net/sed1line.txt

Remplazar cadena en un fichero

Fichero base:

```
<input-fields>
   <data-value name="BEAHOME" value="/u01/weblogic/mid1036" />
   <data-value name="USER_INSTALL_DIR" value="/u01/domains/wls" />
   <data-value name="INSTALL_NODE_MANAGER_SERVICE" value="no" />
   <data-value name="COMPONENT_PATHS" value="WebLogic Server" />
</input-fields>
```

Substituir el campo **value** para cada **name** de lo que hay entre comillas.

```
sed -i '/BEAHOME/s/value="[^"]*"/value="{{ bea.home }}"/' silent.xml
```

Resultado:

```
 <data-value name="BEAHOME" value="{{ bea.home }}" />
```

Cambiar por el mismo campo que pone en **name** En \1 guarda la variable y substituye el campo **value** por ella:

```
sed -e '/<data-value/s/name="\(.*\)" *value="[^"]*"/name="\1" value="{{ \1
}}"/' file.xml
```

Substituir después del = todo lo que haya.

```
[ENGINE]
Response File Version=1.0.0.0.0
```

```
[GENERIC]
ORACLE_HOME=/opt/middleware
INSTALL_TYPE=Weblogic Server
MYORACLESUPPORT_USERNAME=
MYORACLESUPPORT_PASSWORD=
DECLINE_SECURITY_UPDATES=true
SECURITY_UPDATES_VIA_MYORACLESUPPORT=false
PROXY_HOST=
PROXY_PORT=
PROXY_USER=
PROXY_PWD=
COLLECTOR_SUPPORTHUB_URL=
```

```
sed -e '/ORACLE_HOME=/s/ORACLE_HOME=\(.*\)/ORACLE_HOME=\/u01\/mid12212/'
response.rsp
```

Resultado:

```
ORACLE_HOME=/u01/mid12212
```

Mostrar un trozo de cadena limitado por dos cadenas

```
echo "hola como estamos todos hoy" | sed -e 's/^.*como\([^*]*todos\).*$/\1/'
```

devuelve (con un espacio al principio)

```
 estamos todos
```

Substituir cadena por variable

```
sed -i "s|cadena|$variable|" fichero
```

Contar veces que aparece una ip en un fichero de acceso

```
cat jur | sort | uniq | while read linea; do echo `cat jur | grep $linea |
wc -l` " " $linea >> jur2;done; cat jur2 | sort -n
```

Funcion

Simple

```
#!/bin/bash
function quit {
exit
}
function e {
echo $1
}
e Hello
e World
quit
echo foo
```

La Salida es “Hello World” y sale sin hacer el foo

Devuelve un vector

```
#!/bin/bash
funcion()
{
  salida[1]=hola
  salida[2]=adios
  echo "${salida[@]}"
}

devuelto=( `funcion` )
echo ${devuelto[0]}
echo ${devuelto[1]}
```

AWK

- Entre 2 delimitadores (-F):

```
echo "holaquetalcomoestamos" | awk -F 'hola|estamos' {'print $2'}
```

-Muestra la última columna

```
cat fichero.txt | awk {'print $NF'}
```

-Muestra toda la cadena

```
cat fichero.txt | awk {'print $0'}
```

-Muestra des de la columna n hasta la última (en la d ponemos el delimitador):

```
#echo "hola que tal como estamos" | cut -d" "  -f2-
que tal como estamos
```

```
# echo "hola-que-tal-como-estamos" | awk -F\- '{for(i=1;i<=NF-2;i++){printf
"%s-", $i}; printf $(NF-1)"\n"}'
hola-que-tal-como
```

-Usa delimitador <>

```
# echo "Hola que tal <como estamos> mas texto <dentro etiqueta>" | awk -F
"[<>]" '{print $3}'
mas texto
```

-Condiciones
Busca la palabra ***cadena*** en la columna dos y muestra toda la linea

```
awk '{if ( $2 == "cadena" ) printf $0"\n"}' fichero
```

Para buscar un texto en una columna y evitar el cutre grep | grep -v grep. Busca que en la columna del proceso este la palabra gnome y muestra todo el proceso:

```
ps -ef | awk '{if (index($8,"gnome") >0 ) printf $0"\n"}'
```

-Variables dentro de awk:

```
root="/webroot"
echo | awk -v r=$root '{ print "shell root value - " r}'
```

Recorre todos los valores uno a uno

```
awk '{ for(i = 1; i <= NF; i++) { print $i; } }' fichero.txt
```

CUT

De la segunda hasta el fin

```
echo "hola que tal como estamos" | cut -d" "  -f2-
que tal como estamos
```

De la cuarta hasta el inicio

```
echo "hola que tal como estamos" | cut -d" "  -f-4
```

```
hola que tal como
```

Muestra número de carácteres después de concurrencia

```
awk 'c-->0;$0~s{if(b)for(c=b+1;c>1;c--)print r[(NR-
c+1)%b];print"";print"***************";print;c=a}b{r[NR%b]=$0}'
b=lineas_antes a=lineas_despues s="concurrencia" fichero.txt
```

Muestra un número de columna guardado en una variable

```
# t=2
# echo "hola que tal estamos" | awk -v i=$t '{print $i}'

que
```

GREP

Cuenta número de letras de un fichero:

```
grep -Eo '[^[:blank:]]' <<<cat 5.txt | sort | uniq -c
   4073 a
    711 b
   1297 c
    612 d
   2027 e
    386 f
    565 g
    412 h
   1714 i
    340 j
```

Cuenta número de palabras:

```
 grep -Eo '[^[:blank:]]+' <<<'this  line this this line' | sort | uniq -c
      2 line
      3 this
```

Texto entre dos cadenas:

```
<span class="title">abaixar </span>
```

```
grep -oP '(?<=class="title">).*?(?=</span)' file.txt
```

Varias coincidencias

```
# grep -E 'hola|adios' fichero.txt
```

Buscar por nombre de proceso, para evitar grep -v grep

```
# ps -fc java
```

Posicion de un caracter en una cadena. La letra o en la cadena bolo. Empieza por 0

```
echo "bolo" | grep -b -o o
```

1:o 3:o

Sacar excepciones en SystemOut.log

Saca toda la excepción hasta que la siguiente linea empiece con el formato corchete fecha:

```
[12/9/11 6:34:05:553 CET]
```

Uso:

```
./script.sh " E "
```

```
#!/bin/bash
while read linea
do
  echo $linea
  let numero=`echo $linea | awk -F\: {'print $1'}`+1
  primer=`sed -n "$numero p" SystemOut.log | grep -v "^\["`
  while [ "$primer" != "" ]
  do
    echo $primer
    let numero=$numero+1
    primer=`sed -n "$numero p" SystemOut.log | grep -v "^\["`
    let numero=$numero+1
  done
echo
echo
done < <(grep -n "$1" SystemOut.log)
```

Eliminar carácteres de un fichero binario

tr -cd '\11\12\15\40-\176' < fichero_binario > limpio.txt

contraseñas aleatorias

```
tr -dc A-Za-z0-9+-_ < /dev/urandom | head -c 8;echo
```

```
strings /dev/urandom | grep -o '[[:alnum:]]' | head -n 8 | tr -d '\n'
```

SSH automático con expect

Los comandos principales son:

```
expect: continua cuando recibe esa cadena
send: envía una cadena
interact: salta al prompt
export: asigna variable
```

Si no responde a un expect, salta a la siguiente linea por el timeout que tenga puesto, por defecto 8-10 segundos.

```
#!/usr/bin/expect -f
spawn ssh usuario@maquina
expect "*?assword:*"
send "password_usuario\r"
expect " > "
interact
```

Con variable:

```
#!/usr/bin/expect -f
export contrasenya password_usuario
spawn ssh usuario@maquina
expect "*?assword:*"
send "$contrasenya\r"
expect " > "
interact
```

Asignar parámetros: set param1 [lindex $argv 0] set param2 [lindex $argv 1]

En los send es recomendable poner - - por si alguna contraseña viene en variables con un guión al principio:

```
send -- "$password\r"
```

Para evitar la validación del fingerprint realizar el ssh con el parámetro:

1. o StrictHostKeyChecking=no

Condiciones en Expect

Por ejemplo, si la máquina no la tenemos en known_hosts nos dirá antes la pregunta:

```
Are you sure you want to continue connecting (yes/no)?
```

Para poner una condición en expect:

```
expect {
   "Are you sure you want to continue connecting (yes/no)?" { send "yes\r";
exp_continue }
   "*?assword:*" {send "password\r"}
}
```

Traza en Expect

exp_internal 1

Bucles con Expect

```
#!/bin/bash
lista=`cat <<EOF
fichero1
fichero2
EOF`
for fichero in $lista; do
/usr/bin/expect <<EOF
spawn scp $fichero usuario@servidor:
expect *assword*
send contrasenya\r
expect *#
EOF
done
```

Login con contraseña encriptada

Mirar http://wiki.legido.com/doku.php?id=informatica:linux:script&#encriptar_contrasenas para ver

como se encripta contraseña

```
#!/bin/bash
password=`echo U2FsdGVkX19VVheWdlKL5do97riAmAUq | openssl enc -base64 -d |
openssl enc -des3 -k 1469 -d`
/usr/bin/expect <<EOF
spawn ssh usaurio@maquina
expect "*?assword:*"
send "$password\r"
send "\r"
expect "*:~$*"
send "touch jurjur\r"
expect "*:~$*"
send "exit\r"
EOF
```

Insertar texto en un fichero después de una linea buscada

```
#!/bin/bash
texto_busqueda="http://guifi.net"
texto_insertar="*********"
inicio=0

grep -n "$texto" $1 | while read linea
do
    num=`echo $linea | awk -F\: {'print $1'}`
    let diff=$num-$inicio
    head -$num $1 | tail -$diff
    echo $texto_insertar
    let inicio=$num
done
```

Tabla de instrucciones dependiendo de la shell ejecutada

http://fringe.davesource.com/Fringe/Computers/Languages/Shell_Conversion

Trucos

Como curiosidad nunca uses:

```
$(cat algo)
```

Utiliza:

```
$(< algo)
```

Es infinitamente más rápido. De hecho es RARA la vez que hay que usar cat en un shell script. Siempre hay mejores alternativas.

Y tampoco uses 'let', es feo. Usa:

```
((CONTADOR++))
```

Escribe la salida del último comando # echo $(!!)

Liberar espacio fichero borrado pillado por un proceso

A veces borramos sin querer un fichero pillado por un proceso y vemos que no libera el espacio:

```
# df -h /tmp
/tmp 4.9G  4.6G     0 100% /tmp
```

```
# ls -la /tmp

-rw-r----- 1 ruth ruth 1.2G Feb 27 10:36 005c81d2-1a5b-449a-
aa0b-13a36509bd6a.zip
```

Borramos el fichero pero no se libera espacio:

```
# df -h /tmp
/tmp 4.9G  4.6G     0 100% /tmp
```

Vemos que proceso tiene pillado el fichero

```
# /usr/sbin/lsof  | grep 005c81d2-1a5b-449a-aa0b-13a36509bd6a.zip
```

```
COMMAND     PID     USER   FD      TYPE             DEVICE   SIZE/OFF
NODE NAME
java      17923 ruth  777w      REG              253,2          0      98310
/tmp/005c81d2-1a5b-449a-aa0b-13a36509bd6a.zip (deleted)
```

Si no podemos parar el proceso por ser crítico, truncamos el fichero. El primer número es el PID del proceso el segundo el FD:

```
# > /proc/17923/fd/777
```

Ahora ya está libre el espacio

Fuente: http://www.unixwerk.eu/linux/deleted_files.html

XARGS

```
ls | xargs -i mv {} {}_old
```

Date

http://www.cyberciti.biz/tips/linux-unix-get-yesterdays-tomorrows-date.html

Formato fecha custom:

```
date +%Y%m%d_%H%M%S
20190420_225307

date +'%d/%m/%Y %H:%M:%S'
10/10/2023 20:11:31
```

```
date --date='tomorrow'
date --date='1 day'
date --date='10 day'
date --date='10 week'
date --date='10 month'
date --date='10 year'
```

```
date --date='yesterday'
date --date='1 day ago'
date --date='10 day ago'
date --date='10 week ago'
date --date='10 month ago'
date --date='10 year ago'
```

Pasar una fecha a unixtime:

```
date "+%s" -d "01/02/2021 13:11:04"
```

Coge formato americano MM/DD/YYY. Si probamos esto falla:

```
date "+%s" -d "31/01/2021 13:11:04"
```

Convertir unixtime a formato texto:

```
date -d @1448488800000
```

Thu Oct 13 06:00:00 PM CEST 47870

Reordenamos las fechas:

```
echo 31/01/2021 14:12:30 | awk 'BEGIN{FS=OFS="/"}{print $2,$1,$3}'|xargs -i
date -d"{}" +"%s"
```

History

Poner fecha en historial history

export HISTTIMEFORMAT='%d/%m/%Y %H:%M:%S: '

Salir terminal sin grabar history

```
kill -9 $$
```

Historial infinito

export HISTSIZE=””

Unixtime

Escribir unixtime:

```
# date +%s

1384245790
```

Pasar unixtime a fecha texto

```
# date -d '@1384245790' +%d

date -d '@1384245790' +%d/%m/%Y
```

Pasar de formato texto a unixtime:

```
# date +%s -d"Jan 1, 1980 00:00:01"
315529201
```

```
# date +%s -d"Jan  3 08:22:12 2015 GMT"
1420273332
```

Para usar **strftime** hay que tener instalado el paquete **gawk**

convertir fecha:

```
sed -r 's/(\[|])//g' | awk ' { $1=strftime("%D %T",$1); print }'
```

Ejemplo:

```
# echo 1339506985|awk '{print strftime("%D %T",$1)}'

06/12/12 15:16:25
```

http://influencd.co.uk/blog/2011/03/converting-unix-epoch-time-in-bash-history/

Encriptar contraseñas

Para encriptar una contraseña. mi_salt puede ser numero o palabra. Sin ese valor no se puede desencriptar:

```
# echo -n "mi_contraseña" | openssl enc -des3 -k <mi_salt> | openssl enc -
base64

U2FsdGVkX1/js2CVDYiDj4H4MAt4TETG
```

Para desencriptar:

```
# echo "U2FsdGVkX19bIX1uNHDlZniiqz51ggEW" | openssl enc -base64 -d | openssl
enc -des3 -k <mysalt> -d
```

Por ejemplo:

```
# echo -n "jurjur" | openssl enc -des3 -k 14 | openssl enc -base64
U2FsdGVkX1+kzYuThdOeyeUnyXWpj9o0
```

```
# echo "U2FsdGVkX19bIX1uNHDlZniiqz51ggEW" | openssl enc -base64 -d | openssl
enc -des3 -k 14 -d
jurjur
```

Cadenas de texto

Sacar la posición de una subcadena:

Con variable:

```
string="hola que tal como estamos"
# echo | awk '{ print index("'"${string}"'", "tal")}'
10
```

```
# echo | awk '{ print index("'"hola que tal como estamos"'", "como")}'
14
```

Pedir variable que no se ve en el prompt

Con esta opción no escribe nada:

```
read -s variable
```

Script que escribe asteriscos:

```
#!/bin/bash

unset PASSWORD
unset CHARCOUNT

echo -n "Enter password: "

stty -echo

CHARCOUNT=0
while IFS= read -p "$PROMPT" -r -s -n 1 CHAR
do
    # Enter - accept password
    if [[ $CHAR == $'\0' ]] ; then
        break
    fi
    # Backspace
    if [[ $CHAR == $'\177' ]] ; then
        if [ $CHARCOUNT -gt 0 ] ; then
            CHARCOUNT=$((CHARCOUNT-1))
            PROMPT=$'\b \b'
            PASSWORD="${PASSWORD%?}"
        else
            PROMPT=''
        fi
    else
        CHARCOUNT=$((CHARCOUNT+1))
        PROMPT='*'
        PASSWORD+="$CHAR"
    fi
done
```

```
stty echo

echo
echo $PASSWORD
```

Repositorio

Para evitar que valide el certificado de un repositorio, por ejemplo si estamos detrás de un proxy:

```
echo "Acquire::https::pkg.jenkins.io::Verify-Peer "false";" >
/etc/apt/apt.conf.d/jenkins
```

ip a s

```
ip addr add 192.168.56.101/24 dev eth0

ip link set eth1 up
```

Generar un número aleatorio random en un rango

```
awk -v min=0 -v max=100000 'BEGIN{srand(); print int(min+rand()*(max-
min+1))}'
```

Entre 4 y 10

```
shuf -i4-10 -n1
```

vim vi

```
yy copia línea
p pega línea
```

```
v selecciona
y copia
p pega
```

Formatear yaml json

```
:%!python -m json.tool
```

Formatear xml

```
:%!xmllint --encode UTF-8 --format -
```

Carácteres especiales

Para ver los carácteres especiales:

```
:set list
```

Vi añade newline al final de fichero. Si obtenemos el error **No newline at end of file** al hacer un diff por ejemplo, podemos hacer esto para verlo, aparecerà un \n en uno de los ficheros al final:

```
od -c fichero
```

Buscar paquete debian

Instalamos apt-file y hacemos apt-file update. Buscamos:

```
apt-file search --regexp '/identify$'
```

Que lo que hace es esta búsqueda:

```
# apt-file search /usr/bin/identify
graphicsmagick-imagemagick-compat: /usr/bin/identify
```

O desde la web:
https://www.debian.org/distrib/packages#search_contents

Fecha certificado

```
SITE_URL="dominio.com"
SITE_SSL_PORT="443"

openssl s_client -connect ${SITE_URL}:${SITE_SSL_PORT} \
```

```
  -servername ${SITE_URL} 2> /dev/null |  openssl x509 -noout  -dates
```