

1. Gaia

Komando-interpretatzailea

Skriptak (Gidoiak)

Juanan Pereira <juanan.pereira@ehu.es

Kepa Bengoetxea <kepa.bengoetxea@ehu.es>

Erreferentziak

The Linux Documentation Project

<http://tldp.org/LDP/abs/html> (Advanced Bash Scripting)

Bash interpretatzailearen laguntza

man bash

LINUX: SISTEMAREN ETA SAREAREN ADMINISTRAZIOA

Udako Euskal Unibertsitatea, Iñaki Alegria , 2003

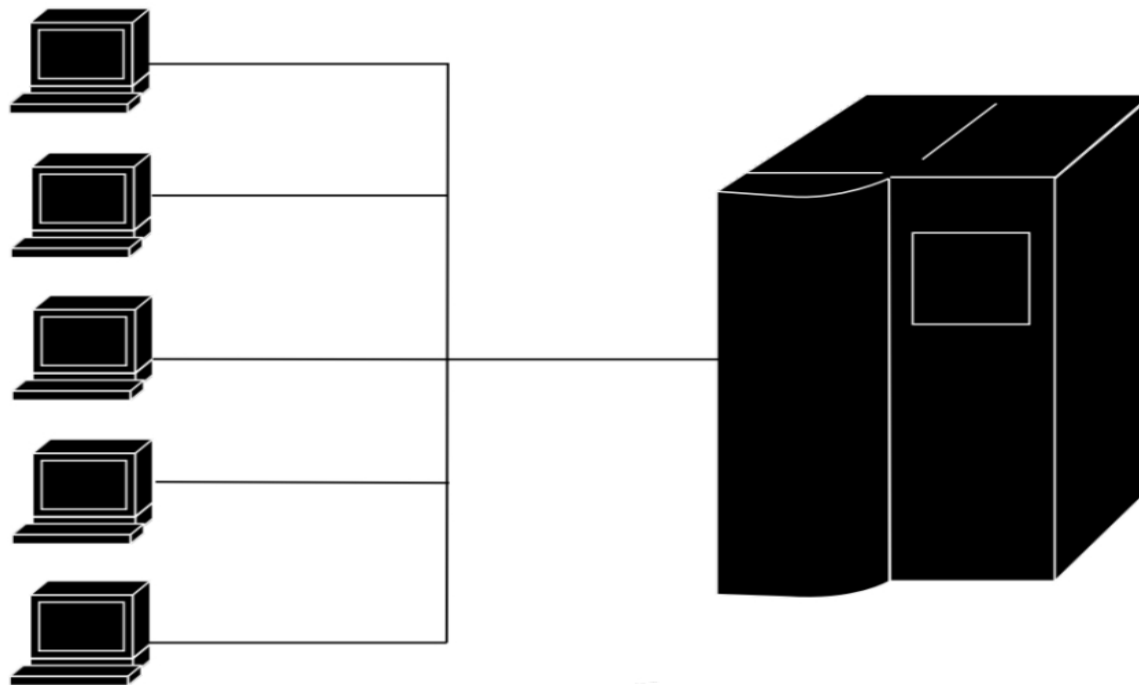
http://www.buruxkak.org/pdf/165_LinuxSistemaren.osoa.pdf

Free eBook: Introduction to the Command Line

<http://dontfearthecommandline.blogspot.com/>

Terminal emulatzailea (tty)

Terminal emulatzailea (tty): terminal-emulatzaile bat zerbitzari batek izaten dituen terminal fisikoak emulatzeko programa informatiko bat da.



Terminal emulatzailea (tty)

Hainbat terminal-emulatzaile ditugu eskuragarri:

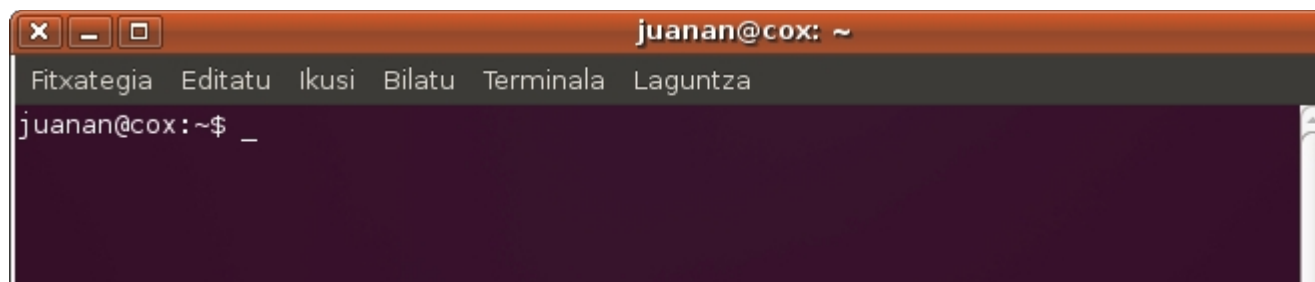
- Idazmahai grafikoan:

xterm (X leiho sistemarako)

gnome-terminal (GNOME ingurunerako)

PuTTY (Windows sistemarako)

- Terminal-emulatzailea UNIX sistema baten komando-interpretatzailea (*shell*-a) atzitzeko erabiltzen da.



Terminal emulatzailea (tty)

Terminal batek, testu karaktere soilak bidali eta jaso egiten ditu, komunikazio sare batetik. Bidalitako karaktereak teklaturan sakatutako teklen balioak dira eta jasotakoak pantailan ikusiko denarekin bat datoz.

Terminal-emulatzaileek emulatzen dituzten terminal fisikoen irudiak:



VT100



IBM 3279



Komando-interpretatzailea (shell)

Shell (edo KI, komando-interpretatzaile) bat komando lerrotik lantzen diren aginduak intepretatu eta egikaritzen ditu. Terminal bat irekitzerakoan, erabiltzailearen kontuan lehenetsita dagoen KI-a exekutatuko da.

Shell motak

Ezagunenak:

sh: garraiagarritasuna ziurtatzen du Unix sistema guztien artean

bash: Linuxek esleitzen duena, besterik esaten ez den bitartean

csh eta **tcsh**: (C-shell) sh hobetuta. Skriptak egiteko C bezalako lengoaia erabiltzen du

*Windows-ek ere badu bere KI-a : **command.com** izenekoa*

Komando-interpretatzailea (shell)

- Erabiltzaile bakoitzak lehenetsitako shell bat dauka esleituta (administratzaileak erabiltzailearen kontua sortzerakoan esleitzen diona)
- Komando-interpretatzailea ez da kernelak eskaintzen duen funtzionalitatea, baizik eta aplikazio arrunt bat
- Nola funtzionatzen du KI batek?

Komando-interpretatzailea (shell)

- 1) erabiltzaileak komando-lerrotik sartutako agindua irakurtzen du
- 2) agindua hauetatik bat izango da (edo ez da agindu zuzena izango):
 - a) barne-komando bat (adibidez: help)
 - b) alias edo goitizen bat (adibidez alias laguntza=help)
 - c) exekutagarri bat (adibidez, /bin/lis)

a, b edo c motakoa bada, KI-k agindua egikarituko du. Bestela errore bat emango du.

Komando-interpretatzailea (shell)

Maiz erabilitako komandoak

man, cd, mkdir, ls, head, tail, more, cat, cp, mv, rm, ln, sort, wc, touch, find, grep, df

Skriptak

Skript bat, lanen automatizazioa ahalbidetzen duen komando fitxategi bat da.

Nola sortu skript bat:

1) Editatu (adibidez vi edo gedit editoreaz)

```
$ vi egoera.sh
```

```
# prozesadore, disko eta memoria baliabideen egoera bistaratu
```

```
uptime
```

```
df
```

```
free
```

2) Egikaritzeko baimenak esleitu

```
$ chmod u+rx egoera.sh
```

3) Egikaritu

```
$ ./egoera.sh
```

Skriptak

Erabiltzaile guztientzako eskuragarri utzi nahi badugu egin berri dugun skripta:

```
$ sudo cp egoera.sh /bin/
```

Orain, egikaritzeko:

```
$ egoera.sh
```

(./ aurrizkia jarri gabe, /bin katalogoa PATH aldagaian baitago)

LAN-INGURUNEA ETA ALDAGAIK

Erabiltzaileak, konektatzen den momentutik, bere oinarrizko ingurunea dauka.

Hasierako ingurune horretan aurredefinitutako aldagai garrantzitsuenak:

- **USER**: erabiltzailearen izena saio berri bat irekitzerakoan.
- **HOME**: erabiltzailearen erro-katalogoa. Saioen hasierako kokapen puntuari dagokion bide absolutua izango da, /home katalogoaren barruan dagoena, oro har.

LAN-INGURUNEA ETA ALDAGAIK

- **PWD**: uneko katalogoa.
- **PATH**: komandoak eta fitxategi exekutagarriak bilatzeko katalogo-zerrenda. Zerrendaren osagaiak “:” karaktereaz bereizten dira.
- **PS1**: gonbitea edo promptaren osaera. Komandoak tekletzeko sistema prest dagoela adierazteko agertu ohi den karaktereari edo karaktere-segidari deitzen zaio gonbitea. Karaktere berezi batzuen bitartez hainbat aldagairen balioa koka daiteke gonbitean

LAN-INGURUNEA ETA ALDAGAIK

Ingurune aldagai baten balioa bistaratzeko:

```
kepa@otoio:~$ echo $PATH
```

```
/home/kepa/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
```

```
kepa@otoio:~$ echo $PS1
```

```
\\u@\\h \\W:\\$
```

Ingurune aldagai baten balioa aldatzeko:

```
PS1="\\u@\\h \\W:\\$ "
```

```
PATH=$PATH:/opt/games
```

LAN-INGURUNEA ETA ALDAGAIK

Ingurune-aldagai globalak ikusi (shell guztiek izango dituzten aldagaiak)

```
$ env
PATH=/home/kepa/perl5/bin:.....
TERM=xterm
SHELL=/bin/bash
USER=kepa
PWD=/home/kepa
EDITOR=/usr/bin/gedit
LANG=en_US.UTF-8
```

LAN-INGURUNEA ETA ALDAGAIK

echo komandoa lagungarria izan daiteke aldagaien balioa ikusteko edo testu soila inprimatzeko)

Adibideak:

```
echo $PATH
```

```
echo "Gaurko data: `date`"
```

```
echo -e "1.lerroa \n 2. lerroa"
```



Eskape karaktereak
Interpretatu nahi
ditugu

Oharra: \n eskape karaktere bat da (Lerro jauzi bat idazten du)
\t beste eskape karaktere baten adibidea da (Tabulatzailea, kasu honetan)

LAN-INGURUNEA ETA ALDAGAIK

BASH interpretatzailean:

- **aldagaiak** ez dira erazagutu behar
- aldagai baten datu mota dinamikoa da

Aldagai baten balioa zenbaki gisa tratatu bada, BASHek eragiketa matematikoak onartuko ditu aldagai horien gainean.

Adibidez:

```
a=1
a=`expr $a + 1`
echo $a
2
a="b"
echo `expr $a + 1`
expr: argumentu ez zenbakizkoa
```

LAN-INGURUNEA ETA ALDAGAIK

echo komandoak karaktere berezi hauek onartzen ditu:

Kakotxa (') artean idazten duguna *echo* komandoak ez du interpretatuko

Adibidez:

```
echo '$PATH'  
$PATH
```

Alderantzizko kakotxa (`) artean dagoena komando gisa hartuko du (exekutatu ondoren lortzen den emaitza da pantailaratuko dena)

Adibidez:

```
echo "Konektatuta dauden erabiltzaileak: `who`"
```

Konektatuta dauden erabiltzaileak:

```
juanan  
kepa
```

'

“

`

LAN-INGURUNEA ETA ALDAGAIK

Kontramarra (\) : ondorengo karakterearen esanahia (izatekotan) desgaitzen du

Adibidez:

```
echo "$HOME"  
    /home/kepa  
echo "\$HOME"  
    $HOME
```

Skripten programazioa: Kontrol-egiturak

test funtzioa balio eta aldagaien arteko konparaketak egiteko erabiltzen da. Horrez gain, fitxategien gaineko eragiketak egiteko ere erabili daiteke.

Adibideak:

Bi kate (string) konparatzeko:

```
test "kaixo" = "kaixo"  
echo $?  
0 <-- (0 eta TRUE baliokideak dira)
```

Oharra: \$? aldagaiak egikaritu den azkenengo komandoaren emaitza jasotzen du.

```
test "kaixo" = "agur"  
echo $?  
1 <-- 0 ez den zenbaki bat jasozerako, kate ezbedinak dira
```

Skripten programazioa: Kontrol-egiturak

Bi string aldagaien arteko konparaketa egiteko:

```
kate1 = "kaixo"  
kate2 = "kaixo"  
test $kate1 = $kate2  
echo $?
```

Bi zenbaki aldagaien arteko konparaketa egiteko:

```
zenb1=1  
zenb2=1  
test $zenb1 -eq $zenb2  
echo $?  
0
```

Skripten programazioa: Kontrol-egiturak

Orokorrean, kateen arteko konparaketak egiteko, horrelako eragiketak ditugu eskuragarri [test](#) aginduan:

```
test $kat1 = $kat2
```

```
test $kat1 != $kat2
```

```
test -z $kat # TRUE kat katearen luzera 0 bada (zero)
```

```
test -n $kat # TRUE kat katearen luzera 0 ez bada (non-zero)
```

<pre>kat="aaa" test -z \$kat echo \$? 1 (FALSE)</pre>	<pre>kat="" test -z \$kat echo \$? 0 (TRUE)</pre>	<pre>kat="aaaa" test -n \$kat echo \$? 0 (TRUE)</pre>
---	---	---

Skripten programazioa: Kontrol-egiturak

Zenbakien arteko konparaketak ere egin daitezke. Zehazki:

zenb1=1

zenb2=2

```
test $zenb1 -eq $zenb2
      # eq = equals
echo $?
1 # ez dira berdinak
```

```
test $zenb1 -ne $zenb2
      # ne = not equivalent
echo $?
0 # bai, (1 != 2)
```

```
test $zenb1 -lt $zenb2
      # lt = less than
echo $?
0 # bai, (1 < 2)
```

```
test $zenb1 -le $zenb2
      # le = less than or equal
echo $?
0 # bai, (1 <= 2 )
```

```
test $zenb1 -gt $zenb2
      # gt = greater than
echo $?
1 # ez da betetzen
```

```
test $zenb1 -ge $zenb2
      # ge = greater than or equal
echo $?
1 # ez da baldintza betetzen
```

Skripten programazioa: Kontrol-egiturak

Fitxagien gaineko eragiketak **test** funtzioa erabiliz:

test funtzioak TRUE (hau da, 0) itzuliko du:

test -e \$fitxategi # fitxategia existitzen bada

test -f \$fitxategi # fitxategia existitzen bada eta EZ bada karpeta edo dispositiboa

test -s \$fitxategi # fitxategiaren tamaina != 0 bada

test -d \$fitxategi # fitxategia berez karpeta bat bada

test -b \$fitxategi # fitxategia bloke-dispositibo bat bada (fd0, hda, ...)

test -G \$fitxategi # fitxategiaren taldea eta uneko erabiltzailearen taldea berdinak badira

test \$fitx1 -nt \$fitx2 # fitx1 fitx2 baino berriagoa bada (newer than)

test \$fitx1 -ot \$fitx2 # fitx1 fitx2 baino zaharragoa bada (older than)

Skripten programazioa: Kontrol-egiturak

Eragiketa **boolearrak** test funtzioan:

`! expresioa` # expresioa ezeztu egiten du (NOT)

`expr1 -a expr2` # AND eragiketa boolearra

`expr1 -o expr2` # OR eragiketa boolearra

Adibideak: (if kontrol-egitura aurkeztuz)

```
y=1
x=1
if test $x -eq 1 -a $y -eq 1
then
    echo "berdinak (eta=1) dira"
fi
```

```
if test -e $fitx
then
    echo "$fitx fitxategia badago"
else
    echo "$fitx fitxategia ez da existitzen"
fi
```

```
if [ ! -f "$fitx" ]
then
    echo " $fitx ez dago."
fi
```

`if ! test -f "$fitx"; then echo " $fitx ez dago"; else echo "badago"; fi`



Skripten programazioa: Kontrol-egiturak

Konparaketak egiteko beste modu bat []

```
if [ $zenb1 -eq $zenb2 ]  
then  
    echo $zenb1  
fi
```

← Zuriune!

```
if [ $z1 -eq $z2 ] # zenbakien arteko konparaketa  
    then echo "$z1=$z2"  
fi
```

```
if [ $k1 == $k2 ] # kate edo string-en arteko konparaketa  
    then echo "$k1=$k2"  
fi
```

Adi: ;

Lerro bakar batean:

```
if [ $kate1 == "kaixo" ]; then echo "kaixo"; fi
```

Skripten programazioa

[read](#) komandoak erabiltzaileak teklatutik sartzen duena jasotzen du.

Adibidea:

```
echo "karpeta baten izena sartu:"  
read dir  
if test -d $dir  
then  
  echo "$dir karpetaren edukia honakoa da:"  
  echo "`ls $dir`"  
fi
```

Skripten programazioa

[expr](#) komandoaz eragiketa matematikoak egin ditzakegu:

```
expr 2 + 200  
202
```

```
expr 4 \* 2  
8
```

```
expr 4 / 2  
2
```

```
expr 5 % 2 # hondarra  
1
```

String-en gainean eragiketak egiteko ere [expr](#) erabili daiteke:

```
$ string="ikusietaikasi"  
$ position=3  
$ length=4  
$ z=`expr substr $string $position $length`  
$ echo $z  
usie  
$ expr length $z  
4
```

Skripten programazioa

expr erabiltzen duen skript baten adibide osoa:

```
$ vi batura.sh
  echo "Sar itzazu bi zenbaki oso:"
  read z1 z2
  echo "Bien arteko batura `expr $z1 + $z2` da"
```

```
$ chmod a+x batura.sh
```

```
$ ./batura.sh
Sar itzazu bi zenbaki oso:
2 3
Bien arteko batura 5 da
```

Skripten programazioa

El shell también tiene un modo de depuración real.

- Si hay un error en tu script "scriptconerror" entonces puedes depurarlo con:

```
bash -x scriptconerror
```

Esto ejecutará el script y mostrará todas la sentencias que se ejecutan con las variables y comodines ya expandidos.

Skripten programazioa

Ingurunea eta parametroak:

BASH interpretatzaileak aldagai batzuk erreserbatuta ditu parametro berezi gisa erabiltzeko. Zehazki:

\$0, \$1, \$2, \$3... \$9 aldagai bereziak dira eta skript batean honako esanahia dute:

\$0 skriptaren izena gordetzen du

\$1 skriptari parametro gisa pasatzen zaion lehenengo katea

\$2 skriptari parametro gisa pasatzen zaion bigarren katea

...

\$# skriptari zenbat parametroekin deitu zaion esango digu

\$* parametro guztiak zerrendatzen ditu

\$? azkenengo eragiketak sortu duen egoera kodea (0 --> TRUE , !=0 --> FALSE)

\$ cat param.sh

echo "Parametro kopurua: \$#"

echo "Lehenengo parametroa: \$1"

echo "Zerogarren parametroa: \$0"

\$./param.sh kaixo agur

Parametro kopurua: 2

Lehenengo parametroa: kaixo

Zerogarren parametroa: ./param.sh

Skripten programazioa

Zer egin 9 parametro baino gehiago tratatu nahi izanez gero?

shift agindua erabili parametroak ezkerreruntz mugitzeko.

```
echo "Skriptaren izena : $0"  
echo "Lehenengo parametroaren balioa: $1"  
echo "Bigarren parametroaren balioa : $2"  
echo "Sartutako parametro kopurua  : $#"  
echo "Parametro guztien zerrenda   : $*"  
shift  
echo "Shift egin eta gero, lehenengo parametroaren balioa: $1"
```

```
./proba.sh 1 2 3 4 5 6 7 8 9 10 11
```

```
Skriptaren izena : ./proba.sh  
Lehenengo parametroaren balioa: 1  
Bigarren parametroaren balioa : 2  
Sartutako parametro kopurua  : 11  
Parametro guztien zerrenda   : 1 2 3 4 5 6 7 8 9 10 11  
Shift egin eta gero, lehenengo parametroaren balioa: 2
```


Skripten programazioa: Kontrol-egiturak

FOR begizta honako patroia jarraitzen du:

```
for ALDAGAIA in ZERREND  
do  
    AGINDUAK  
done
```

Bere eginkizuna, "do" eta "done" etiketen artean dagoena N aldiz exekutatzea da, non N, ZERRENDaren elementu kopurua izango den.

Adibidez:

```
for kontagailu in 1 2 3  
do  
    echo "Begiztaren $kontagailu buelta"  
done
```

Oharra: kontagailu aldagaiak, hasieran 1 balioa izango du, gero 2, eta bukatzeko 3. Hiru buelta eman ondoren, begizta amaituko da.

Skripten programazioa: Kontrol-egiturak

Beste adibide bat:

```
katea=`seq 1 3`  
for a in $katea  
do  
    echo kaixo  
done
```

`seq X Y` aginduak, X-tik Y-ra doan segida sortuko du (non X eta Y zenbaki osoak diren)
`seq X SALTO Y` , gauza bera egingo du, baina segidaren elementuen arteko distantzia SALTO izango da.

Adibidez:

```
seq 1 3  
1 2 3
```

```
seq 9 -1 5  
9 8 7 6 5
```

Skripten programazioa: Kontrol-egiturak

IF kontrol-egitura jada erabili badugu ere, hona hemen adibide oso bat

```
read -p "Sartu fitxategi baten izena:" $file
if [ -z $file ];then
    echo "Fitxategi baten izena behar dut!"
    exit 1
elif [ ! -w $file ];then
    echo "Idazteko baimenik ez?"
    exit 1
else
    echo "kaixo $USER" >> $file
fi
....
exit 0
```

Skripten programazioa

- Komando segida:

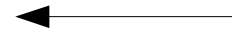
\$ **ls lana ; ls atazak** > fitx.txt



Adi! Ez da gauza bera...

- Komando taldea:

\$ **(ls lana;ls atazak)** > fitx.txt



Skripten programazioa: Kontrol-egiturak

case kontrol-egiturak aldagai baten balioaren arabera komando batzuk egikarituko dugu:

ADIBIDEA

```
read -p "Zein da gustukoen duzun fruta?" fruta
case "$fruta" in
    sagarra)      echo "Mmmh... oso gosoak!"
                  ;;
    banana)       echo "Hm, Kanariaseko platanoa?"
                  ;;
    laranja|mandarina) echo "Ez ditut gustokoak!"
                  echo "Utikan!"
                  exit 1
                  ;;
    *)            echo "Ez du fruta hori ezagutzen"
                  ;;
esac
```

Skripten programazioa: Kontrol-egiturak : While begiztak

\$ **help while** **edo** **man bash**

while COMMANDS; do COMMANDS; done



Baldintza betetzen den
bitartean



Komandoak exekutatu

```
n=1
batura=0
while test $n -le 9
do
    read -p "Zenbaki osoa sartu" zenb
    batura=`expr $batura + $zenb`
    n=`expr $n + 1`
done
echo "Batura=$batura"
```

Parametro lokala (ez du komando
lerrotik sartzen den parametroarekin
zerikusirik)

skripta.sh

```
function agurtu()  
{  
    echo "Kaixo $1"  
}  
read -p "Sartu izen bat:" n  
agurtu $n
```

```
./skripta.sh  
Sartu izen bat: Kepa  
Kaixo Kepa
```

“*. fitxategi*” erabiliz, fitxategia duen guztia uneko skriptan txerta daiteke

proba.sh

./skripta.sh

echo “Agurtu dugu erabiltzailea”

Gauza bera egiteko beste aukera bat: **source** ./skripta.sh

./proba.sh

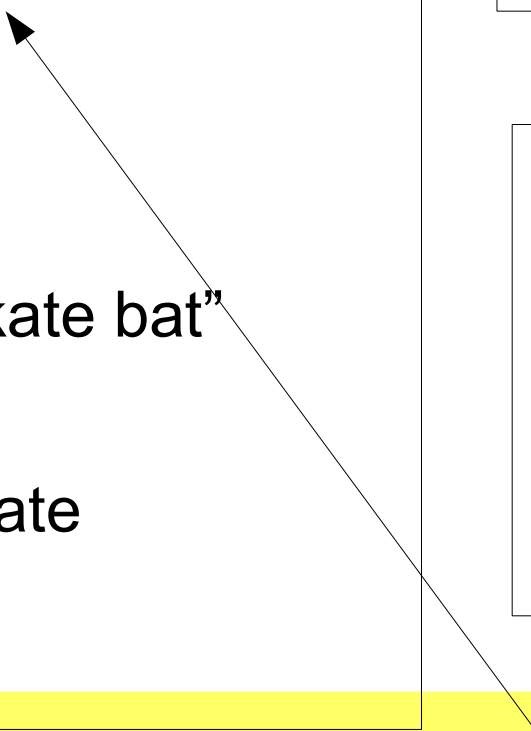
Sartu izen bat: Kepa

Kaixo Kepa

Agurtu dugu erabiltzailea

Funtzioak (Return)

```
$vi proba2.sh
function berdinabc ()
{
  if test $1 = "abc"
  then return 0
  else return 1
  fi
}
echo "Sartu kate bat"
read kate
berdinabc $kate
echo $?
```



```
./proba2.sh
Sartu kate bat
abc
0
```

```
$ ./prueba2.sh
Sartu kate bat
pepe
1
```

Funtzioaren emaitza itzultzeko erabiltzen da "return". Adi! Zenbakekin soilik...