Uvod v programiranje v BASH - HOW-TO

Mike G mikkey at dynamo.com.ar Čet Jul 27 09:36:18 ART 2000

Ta spis vam bo pomagal pri vaših začetkih pisanja osnovnih in nekoliko zahtevnejših lupinskih skriptov. Njegov namen ni biti vsemogočen vodnik (glej naslov). Sam NISEM nikakršen izvedenec v programiranju v lupini in sem ta spis napisal zato, ker se bom pri tem veliko naučil, poleg tega pa bo morda koristilo tudi drugim ljudem. Vsak odziv je dobrodošel, še posebno v obliki popravkov :)

Kazalo

1 Uvod

1.1 Kje dobiti najnovejšo različico?

http://www.linuxdoc.org/HOWTO/Bash-Prog-Intro-HOWTO.html

1.2 Potrebno znanje

Dobro poznavanje ukazne vrstice GNU/Linux ter osnovnih pojmov programiranja je dobrodošlo. Čeprav tole ni uvod v programiranje samo, pojasnjuje (oziroma si vsaj prizadeva pojasniti) mnogo osnovnih pojmov.

1.3 Uporaba tega spisa

Ta spis vam bo prišel prav v sledečih primerih:

- Imate nekaj predstav o programiranju in bi radi začeli pisati lupinske skripte.
- Imate nejasno predstavo o programiranju in potrebujete nekaj napotkov.
- Želite videti nekaj lupinskih skriptov ter komentarjev, da boste lahko začeli pisati svoje lastne.
- Selite se iz DOS/Windows (oziroma ste se pravkar preselili) in bi želeli narediti "paketne"procese.
- Ste popoln računalniški navdušenec in preberete vsak how-to, ki vam pride na pot.

2 Zelo preprosti skripti

Ta HOW-TO vam bo dal nekaj namigov o lupinskih skriptih, ki bodo mo čno oprti na primere.

V tem delu boste našli nekaj kratkih skriptov, ki vam bodo pomagali razumeti razli čne tehnike.

2.1 Tradicionalen skript 'Hello World'

```
#!/bin/bash
echo Hello World
```

Ta skript ima samo dve vrstici. Prva pove sistemu, kateri program naj uporabi pri zaganjanju datoteke.

Druga vrstica vsebuje edin ukaz, ki ga skript izvede, in ta izpiše 'Hello World' na terminal.

Če dobite nekaj kot *./hello.sh: Command not found.*, je verjetno napa čna prva vrstica, '#!/bin/bash' - da bi ugotovili pravilno pot do bash izvedite 'whereis bash' ali poglejte v poglavje 'Iskanje bash'.

2.2 Zelo preprost skript za varnostno kopijo

```
#!/bin/bash
tar -cZf /var/moja-varnostna-kopija.tgz /home/jaz/
```

V tem skriptu namesto izpisovanja sporo čila na terminal naredimo arhiv uporabnikovega doma čega imenika. Ne uporabljajte tega skripta - obstaja tudi mnogo boljši, ki bo predstavljen kasneje.

3 Vse o preusmerjanju

3.1 Teorija in hitri napotki

Obstajajo trije opisniki datoteke - stdin (standardni vhod), stdout (standardni izhod) ter stderr (standardni izhod za napake).

V osnovi lahko:

1. preusmerite stdout v datoteko
2. preusmerite stderr v datoteko
3. preusmerite stdout v stderr
4. preusmerite stderr v stdout
5. preusmerite stderr in stdout v datoteko
6. preusmerite stderr in stdout v stdout
7. preusmerite stderr in stdout v stderr

1 ’predstavlja’ stdout in 2 stderr.

Majhno pojasnilo za predstavo o teh stvareh: z ukazom less si lahko ogledate tako stdout (ki bo ostal v medpomnilniku) kot stderr, ki se bo izpisal na zaslon, vendar bo izginil, ko se boste poskušali premikati po medpomnilniku.

3.2 Primer: stdout v datoteko

To bo preusmerilo izhodni tok programa v datoteko.

```
ls -l > ls-l.txt
```

V tem primeru bo ustvarjena datoteka z imenom ’ls-l.txt’, ki bo vsebovala tisto, kar bi se sicer izpisalo na zaslon, ko bi pognali ukaz ’ls -l’.

3.3 Primer: stderr v datoteko

Takole lahko standardni izhod za napake preusmerimo v datoteko.

```
grep da * 2> grep-napake.txt
```

Ustvarjena bo datoteka z imenom ’grep-napake.txt’, v njej pa bo vse, kar bo ukaz ’grep da *’ izpisal na stderr.

3.4 Primer stdout v stderr

Tukaj bomo izhodni tok stdout preusmerili v isti opisnik datoteke kot stderr.

```
grep da * 1>&2
```

Del izpisa, ki bi sicer šel na stdout, bo v tem primeru preusmerjen na stderr.

3.5 Primer: stderr v stdout

To bo povzročilo, da bo izhodni tok stderr preusmerjen v isti opisnik datoteke kot stdout.

```
grep * 2>&1
```

Stderr del izhodnega toka bo tako preusmerjen na standardni izhodni tok - če boste ta ukaz preko cevovoda povezali s programom less, boste opazili, da bodo vrstice, ki navadno ’izginejo’ (ker so izpisane na stderr), tokrat ostale vidne (ker smo jih preusmerili na stdout).

3.6 Primer: stderr in stdout v datoteko

Celoten izhodni tok programa bomo preusmerili v datoteko. To včasih pride prav pri izvajanju opravil v cron-u, ko želite ukaz "utišati".

```
rm -f $(find / -name core) &> /dev/null
```

Ta ukaz (še vedno smo pri vnosu v cron) bo izbrisal vse datoteke z imenom 'core', ki se bodo nahajale v kateremkoli imeniku. Naj vas opozorim, da morate biti precej gotovi glede tega, kaj bo ukaz storil, vkolikor boste njegov izhod zavrgli.

4 Cevovodi

To poglavje preprosto in praktično razloži kako uporabljati cevovode in zakaj bi to sploh hoteli.

4.1 Kaj so cevovodi in zakaj bi jih hoteli uporabljati

Cevovodi vam omogočajo (zelo preprosto) povezati izhodni tok enega programa z vhodnim tokom drugega.

4.2 Primer: preprost cevovod s programom sed

To je zelo preprost način uporabe cevovodov.

```
ls -l | sed -e "s/[aeio]/u/g"
```

V tem primeru se zgodi sledeče: najprej se izvši ukaz 'ls -l', njegov izhodni tok pa je - namesto, da bi se izpisal na terminal - posredovan programu sed, ki nato izpiše, kar mu je zaukazano.

4.3 Primer: alternativa ukazu 'ls -l *.txt'

To je verjetno bolj neroden način izvajanja 'ls -l *.txt', vendar je tu zaradi prikaza delovanja cevovodov in ne zaradi odločanja o primernosti uporabe ukaza samega.

```
ls -l | grep "\.txt$"
```

Tukaj je izhodni tok ukaza 'ls -l' posredovan programu grep, ki nato izpiše vrstice, ki vsebujejo regularni izraz "\.txt$".

5 Spremenljivke

Spremenljivke lahko uporabljate prav tako, kot v vseh drugih programskih jezikih. Podatkovnih tipov tukaj ni - spremenljivka lahko vsebuje število, znak ali niz znakov.

Spremenljivke vam ni treba deklarirati, ustvari se takoj, ko ji pripišete vrednost.

5.1 Primer: Hello World! z uporabo spremenljivk

```
#!/bin/bash
NIZ="Hello World!"
echo $NIZ
```

Vrstica 2 ustvari spremenljivko z imenom NIZ in ji priredi niz "Hello World!". VREDNOST spremenljivke nato dobimo tako, da na začetek postavimo znak '$'. Če tega znaka ne boste uporabili, bo izhod programa drugačen - verjetno ne takšen, kot bi si želeli (kar poskusite!).

5.2 Primer: Zelo preprost skript za varnostno kopijo (nekoliko boljši)

```
#!/bin/bash
DATOTEKA=/var/moja-varnostna-kopija-$(date +%Y%m%d).tgz
tar -cZf $DATOTEKA /home/jaz/
```

Ta skript prinaša še eno novost. Za začetek morate razumeti ustvarjanje spremenljivke in prirejanje vrednosti v vrstici 2. Gotovo ste opazili izraz '$(date +%Y%m%d)'; če boste skript tudi pognali, boste ugotovili, da izvrši ukaz med oklepaji ter zajame njegov izhod.

Ime izhodne datoteke tega skripta bo vsak dan drugačno, ker smo za ustvarjanje imena uporabili ukaz date s predpisano obliko izhoda (+%Y%m%d). To lahko še nadalje spremenite z drugačnim predpisom oblike.

Še nekaj primerov:

echo ls

echo $(ls)

5.3 Lokalne spremenljivke

Lokalne spremenljivke lahko ustvarimo s ključno besedo *local*.

```
#!/bin/bash
HELLO=Hello
function hello {
        local HELLO=World
        echo $HELLO
}
echo $HELLO
hello
echo $HELLO
```

Ta primer nazorno prikazuje uporabo lokalne spremenljivke.

6 Pogojni stavki

Pogojni stavki vam omogočajo odločitev o izvajanju niza ukazov glede na ovrednotenje določenega izraza.

6.1 Teorija

Pogojni stavki imajo mnogo oblik. Najbolj osnovna je: **if** *izraz* **then** *stavek* pri čemer se 'stavek' izvrši le, če je 'izraz' ovrednoten kot resničen. '2<1' je na primer izraz, ki je ovrednoten kot neresničen, '2>1' pa kot resničen.

Pogjni stavki imajo tudi drugačne oblike, kot na primer: **if** *izraz* **then** *stavek1* **else** *stavek2*. 'stavek1' je v tem primeru izvršen le, če je 'izraz' resničen, sicer pa se izvrši 'stavek2'

Še ena oblika pogojnih stavkov: **if** *izraz1* **then** *stavek1* **else if** *izraz2* **then** *stavek2* **else** *stavek3*. Tukaj je dodan le del "ELSE IF 'izraz2' THEN 'stavek2'", ki izvrši 'stavek2', če je 'izraz2' ovrednoten kot resničen. Vse ostalo verjetno razumete (glejte prejšnje oblike).

Nekaj besed o sintaksi:

Osnova za konstrukcijo 'if' je v bash takšna:

if [izraz];

then

ukazi, če je 'izraz' resničen

fi

6.2 Primer: Osnovni pogojni stavek if .. then

```
#!/bin/bash
if [ "foo" = "foo" ]; then
   echo izraz je ovrednoten kot resničen
fi
```

Ukazi, ki se izvršijo, če je izraz v oglatih oklepajih ovrednoten kot resničen, so navedeni med 'then' in 'fi' - 'fi' označuje konec pogojenih ukazov.

6.3 Primer: Osnovni pogojni stavek if .. then ... else

```
#!/bin/bash
if [ "foo" = "foo" ]; then
   echo izraz je ovrednoten kot resničen
else
   echo izraz je ovrednoten kot neresničen
fi
```

6.4 Primer: pogojni stavki s spremenljivkami

```
#!/bin/bash
T1="foo"
T2="bar"
if [ "$T1" = "$T2" ]; then
    echo izraz je ovrednoten kot resničen
else
    echo izraz je ovrednoten kot neresničen
fi
```

7 Zanke for, while in until

To poglavje razlaga zanke for, while in until.

Zanka **for** se nekoliko razlikuje od takšne zanke v drugih programskih jezikih. Omogoča vam zanko, ki se ponovi za vsako 'besedo' v določenem nizu.

While ponavlja ukaze, dokler je nadzorni izraz ovrednoten kot resničen; ustavi se, ko postane izraz neresničen oziroma ko naleti na ukaz za prekinitev zanke.

Zanka **until** deluje skoraj enako kot while, z razliko, da se ukazi izvajajo, dokler je nadzorni izraz ovrednoten kot neresničen.

7.1 Primer zanke for

```
#!/bin/bash
for i in $( ls ); do
    echo beseda: $i
done
```

V drugi vrstici deklariramo spremenljivko i, kateri bomo pripisali različne vrednosti iz $(ls).

Tretja vrstica bi bila po potrebi lahko tudi daljša oziroma bi se pred 'done' (4) lahko zvrstilo več ukazov.

'done' (4) pove, da je ukazov, ki so uporabljali $i konec in da lahko $i pripišemo novo vrednost.

Ta skript sicer ne počne ničesar koristnega, lahko pa bi mu na primer naročili, naj izpiše le določene datoteke (glej prejšnji primer).

7.2 Zanka for kot v programskem jeziku C

Ta zanka je bolj podobna zanki for v C/perl.

```
#!/bin/bash
for i in ‘seq 1 10‘;
do
        echo $i
```

```
done
```

7.3 Primer zanke while

```
#!/bin/bash
STEVEC=0
while [ $STEVEC -lt 10 ]; do
    echo Stevec kaze $STEVEC
    let STEVEC=STEVEC+1
done
```

Ta skript ’oponaša’ dobro znano strukturo ’for’ (C, Pascal, perl, itd.)

7.4 Primer zanke until

```
#!/bin/bash
STEVEC=20
until [ $STEVEC -lt 10 ]; do
    echo STEVEC $STEVEC
    let STEVEC-=1
done
```

8 Funkcije

Kot v skoraj vsej programskih jezikih lahko tudi tu združite dele programa v funkcije - tako bolj smiselno organizirate program ali pa vadite umetnost rekurzije.

Deklaracija funkcije je na moč preprosta: function moja_funkcija { moji_ukazi }.

Funkcijo kličete tako, kot bi bila drug program; samo napišete njeno ime.

8.1 Primer funkcij

```
#!/bin/bash
function izhod {
    exit
}
function hello {
    echo Hello!
}
hello
izhod
echo foo
```

Vrstice 2-4 opisujejo funkcijo ’izhod’, vrstice 5-7 pa funkcijo ’hello’. Če niste popolnoma prepričani, kaj naredi ta skript, kar poskusite!

Kot vidite funkcij ni potrebno deklarirati v kakršnemkoli posebnem vrstnem redu.

Ko boste skript pognali, bo ta najprej klical funkcijo ’hello’, nato funkcijo ’izhod’. Vrstice 10 skript ne bo dosegel nikoli.

8.2 Primer funkcije s parametri

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

Ta skript je skoraj popolnoma enak prejšnjemu. Glavna razlika je funkcija ’e’, ki izpiše svoj prvi parameter. Parametri, ki jih podamo funkciji so obravnavani enako kot parametri, podani skriptu.

9 Uporabniški vmesniki

9.1 Uporaba select za ustvarjanje preprostih menijev

```
#!/bin/bash
IZBIRE="Pozdrav Izhod"
select opt in $IZBIRE; do
    if [ "$opt" = "Izhod" ]; then
     echo koncano
     exit
    elif [ "$opt" = "Pozdrav" ]; then
     echo Hello World
    else
     clear
     echo nedovoljena izbira
    fi
done
```

Če boste pognali ta skript, boste ugotovili, da so to programerjeve sanje za ustvarjanje tekstnih menijev. Konstrukcija je na moč podobna ’for’, le da namesto izvršitve seznama ukazov povpraša uporabnika za vsako ’besedo’ v $IZBIRE.

9.2 Uporaba parametrov z ukazne vrstice

```
#!/bin/bash
if [ -z "$1" ]; then
    echo uporaba: $0 imenik
    exit
fi
IZVORNA_MAPA=$1
CILJNA_MAPA="/var/varnostne_kopije/"
DATOTEKA=home-$(date +%Y%m%d).tgz
tar -cZf $CILJNA_MAPA$DATOTEKA $IZVORNA_MAPA
```

Kaj naredi ta skript bi vam moralo biti jasno. Izraz v prvem pogojnem stavku preveri, ali je program dobil parameter ($1). V primeru, da ga ni, se skript konča z izpisom navodila za uporabo. Preostanek skripta verjetno razumete.

10 Razno

10.1 Branje uporabnikovega vnosa z read

Gotovo bodo priložnosti, ko boste želeli, da uporabnik kaj vpiše. Tole je eden od možnih na činov:

```
#!/bin/bash
echo Vnesite svoje ime
read IME
echo "Zdravo, $IME!"
```

Z read lahko dobite tudi več vrednosti hkrati:

```
#!/bin/bash
echo Vnesite svoje ime in priimek
read IME PRIIMEK
echo "Zdravo, $PRIIMEK $IME!"
```

10.2 Računanje

V ukazni vrstici poskusite tole:

echo 1 + 1

Če ste pričakovali, da boste dobili ’2’, boste nekoliko razočarani. Kaj storiti, če želite, da vam BASH pomaga izračunati nekaj računov? Rešitev je takšna:

echo $((1+1))

Ta ukaz bo dal bolj ’smiseln’ izpis. Isto lahko dosežete tudi takole:

echo $[1+1]

V primeru, da vaši računi vsebujejo ulomke ali težjo matematiko, lahko za računanje uporabite program bc.

Če na primer v ukazni vrstici poženete "echo $[3/4]", boste dobili rezultat 0, ker bash pri izračunih uporablja le cela števila. Za pravilen rezultat bo treba pognati "echo 3/4|bc -l", kar bo vrnilo pravilen rezultat - 0,75.

10.3 Iskanje bash

Mike (glejte Zahvale) v sporočilu piše:

Primeri vedno uporabljajo #!/bin/bash .. morda bi lahko navedel navodilo kako najti bash, če ga ni na tem mestu.

Še najbolje je uporabiti 'locate bash', vendar vsi sistemi nimajo programa locate.

'find ./ -name bash' v korenskem imeniku je navadno prav tako učinkovit.

Mesta, ki jih preverite:

ls -l /bin/bash

ls -l /sbin/bash

ls -l /usr/local/bin/bash

ls -l /usr/bin/bash

ls -l /usr/sbin/bash

ls -l /usr/local/sbin/bash

(več mest se trenutno ne morem domisliti... sicer pa sem bash po različnih sistemih našel v enem od navedenih mest.

Lahko poskusite tudi 'which bash'.

10.4 Vrnjena vrednost programa

Bash shrani vrnjeno vrednost programa v posebno spremenljivko z imenom $?.

Sledeči primer prikazuje kako ugotoviti vrnjeno vrednost programa; predpostavljam, da imenik *dada* ne obstaja. (Tudi tole je predlagal mike.)

```
#!/bin/bash
cd /dada &> /dev/null
echo vv: $?
cd $(pwd) &> /dev/null
echo vv: $?
```

10.5 Zajemanje izhoda ukaza

Ta kratek skript izpiše vse tabele iz vseh podatkovnih baz (če imate seveda nameščen MySQL). Popraviti morate ukaz mysql, da bo vseboval veljavno uporabniško ime ter geslo.

```
#!/bin/bash
DBS=‘mysql -uroot  -e"show databases"‘
for b in $DBS ;
```

```
do
        mysql -uroot -e"show tables from $b"
done
```

10.6 Več izvornih datotek

Več izvornih datotek lahko uporabite z ukazom source.

__TO-DO__

11 Tabele

11.1 Primerjalni operatorji za nize

(1) niz1 = niz2

(2) niz1 != niz2

(3) niz1 < niz2

(4) niz1 > niz2

(5) -n niz1

(6) -z niz1

(1) niz1 je enak niz2

(2) niz1 ni enak niz2

(3) __TO-DO__

(4) __TO-DO__

(5) niz1 ni prazen (vsebuje enega ali več znakov)

(6) niz1 je prazen

11.2 Primerjave nizov

Primerjanje dveh nizov

```
#!/bin/bash
NIZ1='niz'
NIZ2='Niz'
if [ $NIZ1=$NIZ2 ];
then
        echo "NIZ1('$NIZ1') je enak NIZ2('$NIZ2')"
fi
if [ $NIZ1=$NIZ1 ];
then
        echo "NIZ1('$NIZ1') je enak NIZ1('$NIZ1')"
```

```
fi
```

Andreas Beck je v svojem sporočilu predlagal uporabo *if [$1 = $2]*.

To ni pretirano dobra ideja, ker boste v primeru, da je $NIZ1 ali $NIZ2 prazen, dobili sporočilo o napaki. Bolje je uporabiti x$1=x$2 ali "$1"="$2"

11.3 Aritmetični operatorji

+

-

*

/

% (ostanek)

11.4 Primerjalni aritmetični operatorji

-lt (<)

-gt (>)

-le (<=)

-ge (>=)

-eq (==)

-ne (!=)

Če znate programirati v C-ju, preprosto izberite operator, ki ustreza izbranemu operatorju v oklepajih.

11.5 Priročni ukazi

To poglavje je ponovno napisal Kees (glejte Zahvale).

Nekateri izmed navedenih ukazov so že sami po sebi skoraj pravi programski jeziki, zato bodo o njih povedane le osnovne stvari. Če boste želeli bolj poglobljen opis, si lahko ogledate njihove priročnike (man pages).

sed (stream editor - urejevalnik toka)

Sed je ne-interaktiven urejevalnik. Datoteke ne urejate s premikanjem kurzorja po zaslonu, temveč sed-u podate skript navodil za urejanje ter ime datoteke. Sed bi lahko opisali tudi kot filter. Poglejmo nekaj primerov:

```
$sed 's/za_zamenjavo/zamenjava/g' /tmp/nekaj
```

Sed zamenja niz 'za_zamenjavo' z nizom 'zamenjava', pri čemer bere iz datoteke /tmp/nekaj. Rezultat bo poslan na standardni izhod (navadno konzola), lahko pa seveda na konec ukaza dodate '>zajeto', kar bo preusmerilo izhod v datoteko 'zajeto'.

```
$sed 12, 18d /tmp/nekaj
```

Sed bo izpisal vse vrstice razen 12. in 18. Originalne datoteke ta ukaz ne spremeni.

awk (spreminjanje podatkovnih datotek, iskanje ter procesiranje besedila)

Obstaja mnogo izvedb programskega jezika AWK (najbolj znana interpreterja sta GNU-jev gawk in 'new awk' mawk). Princip je preprost: AWK išče določen vzorec ter izvrši niz ukazov, ko ga najde.

Ustvaril sem testno datoteko 'nekaj', ki vsebuje naslednje vrstice:

"test123

test

tteesstt"

```
$awk '/test/ {print}' /tmp/nekaj
```

test123

test

Vzorec, ki ga AWK išče je 'test', ukaz, ki ga izvrši, ko v datoteki /tmp/nekaj najde vrstico s tem vzorcem, pa 'print'.

```
$awk '/test/ {i=i+1} END {print i}' /tmp/nekaj
```

3

V primeru, da iščete mnogo vzorcev, je pametno zamenjati tekst med narekovaji z '-f datoteka.awk' in napisati vzorce ter ukaze v datoteko 'datoteka.awk'.

grep (izpiše vrstice, ki vsebujejo iskani vzorec)

Ukaz grep smo že nekajkrat srečali v prejšnjih poglavjih, ko je bilo potrebno izpisati vrstice z iskanim vzorcem. Vendar grep zmore še več.

```
$grep "iščemo tole" /var/log/messages -c
```

12

Niz "iščemo toleše je v datoteki /var/log/messages pojavil 12-krat.

[priznam, ta primer ni popolnoma resničen - malce sem priredil /var/log/messages :-)]

wc (prešteje vrstice, besede ter znake)

Ta primer ne izpiše točno tistega, kar bi pričakovali. Uporabljena testna datoteka vsebuje naslednje besedilo: *"uvod v bash testna datoteka"*

```
$wc --words --lines --bytes /tmp/nekaj
```

1 5 28 /tmp/nekaj

Wc se za vrstni red podanih parametrov ne zmeni, temveč izpiše statistike vedno enako: <vrstice> <besede> <znaki> <datoteka>.

sort (razvrsti vrstice besedila)

To pot testna datoteka vsebuje naslednje besedilo:

"b

c

a"

```
$sort /tmp/nekaj
```

Izpis izgleda takole:

a

b

c

Ukazi ne bi smeli biti tako enostavni :-)

bc (programski jezik za računanje)

Bc lahko prebere račune iz datoteke, podane v ukazni vrstici, ali pa preko uporabniškega vmesnika. Ta primer prikazuje nekaj ukazov.

Navadno bc zaženem z parametrom -q, ki prepreči izpis pozdravnega sporočila.

```
$bc -q
```

1 == 5

0

0.05 == 0.05

1

5 != 5

0

2 ^ 8

256

sqrt(9)

3

while (i != 9) {

i = i + 1;

print i

}

123456789

quit

tput (inicializacija terminala ali poizvedba podatkovne baze terminfo)

Manjša demonstracija zmožnosti programa tput:

```
$tput cup 10 4
```

Pozivnik se pojavi na (y10,x4)

```
$tput reset
```

Počisti zaslon, pozivnik pa se pojavi na (y1,x1). (y0,x0) je zgornji levi kot zaslona.

```
$tput cols
```

80

Izpiše število znakov terminala v smeri osi x.

Močno vam priporočam, da se dobro seznanite z vsaj temi programi. Seveda pa obstaja še množica drugih majhnih programov, s katerimi v ukazni vrstici lahko izvajate prave čarovnije.

[nekaj primerov v tem poglavju je vzetih iz priročnikov in pogosto zastavljanih vprašanj]

12 Še več skriptov

12.1 Izvajanje ukaza na vseh datotekah v imeniku.

12.2 Primer: Preprost skript za varnostno kopijo (še nekoliko boljši)

```
#!/bin/bash
IZVORNA_MAPA="/home/"
CILJNA_MAPA="/var/varnostne_kopije/"
DATOTEKA=home-$(date +%Y%m%d).tgz
tar -cZf $CILJNA_MAPA$DATOTEKA $IZVORNA_MAPA
```

12.3 Preimenovalnik datotek

```
#!/bin/sh
# renna: preimenuje več datotek glede na določena pravila
# napisal felix hudson Jan - 2000
```

```
# najprej preverimo za različne 'načine', ki jih program ima.
# če prvi argument ($1) ustreza pogoju, izvedemo določen del
# programa ter nato končamo izvajanje skripta

# preverimo, ali gre za primer predpone
if [ $1 = p ]; then

# sedaj se znebimo spremenljivke za način ($1) ter predpone ($2)
  predpona=$2 ; shift ; shift

# hiter test, ki preveri, ali so bila podana imena datotek
# vkolikor niso bila, je bolje, da ne storimo ničesar, kot
# da poskusimo preimenovati datoteke, ki ne obstajajo!!

  if [$1 = ]; then
     echo "datoteke niso bile podane"
     exit 0
  fi

# ta zanka se ponovi za vsako datoteko, ki je bila podana
# skriptu ter preimenuje eno naenkrat
  for datoteka in $*
    do
    mv ${datoteka} $predpona$datoteka
  done

# tukaj končamo izvajanje
  exit 0
fi

# preverimo, ali gre za preimenovanje končnice
# ta del je skoraj enak prejšnjemu, zato nima zaznamkov
if [ $1 = k ]; then
  koncnica=$2 ; shift ; shift

   if [$1 = ]; then
    echo "datoteke niso bile podane"
   exit 0
   fi

 for datoteka in $*
  do
   mv ${datoteka} $datoteka$koncnica
 done

 exit 0
fi

# preverimo, ali gre za zamenjavo vzorca
if [ $1 = z ]; then
```

```
  shift

# tale del sem vključil zato, da v primeru, ko uporabnik ne
# navede parametrov, ne poškodujemo datotek

  if [ $# -lt 3 ] ; then
    echo "uporaba: renna z [izraz] [zamenjava] datoteke... "
    exit 0
  fi

# odstranimo ostale parametre
  STARO=$1 ; NOVO=$2 ; shift ; shift

# ta zanka se ponovi za vsako datoteko, ki je bila podana
# skriptu ter jo preimenuje z uporabo programa 'sed', ki v
# tekstu iz standardnega vhoda poišče izraz in ga zamenja z
# drugim. Tukaj mu na standardni vhod podamo ime datoteke.

  for datoteka in $*
  do
    NOVO_IME=‘echo ${datoteka} | sed s/${STARO}/${NOVO}/g‘
    mv ${datoteka} $NOVO_IME
  done
exit 0
fi

# če smo prišli do sem, to pomeni, da programu ni bila podan
# noben parameter, zato uporabniku povemo, kako se ga uporablja

echo "uporaba;"
echo " renna p [predpona] datoteke.."
echo " renna k [končnica] datoteke.."
echo " renna z [izraz] [zamenjava] datoteke.."
exit 0

# končano!
```

12.4 Preimenovalnik datotek (preprost)

```
#!/bin/bash
# preimenuj.sh
# enostaven preimenovalnik datotek

kriterij=$1
izraz=$2
zamenjava=$3
```

```
for i in $( ls *$izraz* );
do
    datoteka=$i
    novo_ime=$(echo $i | sed -e "s/$iraz/$zamenjava/")
    mv $datoteka $novo_ime
done
```

13 Ko gre kaj narobe (razhroščevanje)

13.1 Načini klicanja BASH

V prvo vrstico skripta napišite

```
#!/bin/bash -x
```

To bo pri izvajanju izpisalo nekaj koristnih informacij.

14 O tem spisu

Sporočite mi vaše predloge/popravke oziroma karkoli bi radi videli v tem spisu. Poskusil ga bom posodobiti kakor hitro bo mogoče.

14.1 (brez) jamčenja

Ta spis ne jamči česarkoli o čemerkoli. In tako dalje.

14.2 Prevodi

Italijanski: William Ghelfi (wizzy at tiscalinet.it) *je tukaj*

Francoski: Laurent Martelli *je neznano kje*

Korejski: Minseok Park *http://kldp.org*

Korejski: Chun Hye Jin *unknown*

Španski: neznan *http://www.insflug.org*

Domnevam, da obstaja še več prevodov, vendar o njih nimam informacij. Če veste za kakšnega, mi to, prosim, sporočite.

14.3 Zahvale

- Ljudem, ki so ta spis prevedli v druge jezike (prejšnje poglavje)
- Nathanu Hurstu za mnogo popravkov.

- Jonu Abbottu za pripombe o aritmetičnih izrazih
- Felixu Hudsonu za skript *renna*
- Keesu van den Broeku za mnoge popravke ter ponovno pisanje poglavja o uporabnih ukazih
- Mike (pink) je imel nekaj predlogov o iskanju bash in preverjanju datotek
- Fieshu za predlog pri poglavju o zankah
- Lion je predlagal, da omenim pogosto napako (./hello.sh: Command not found.)
- Andreasu Becku za nekaj popravkov in komentarjev.

14.4 Zgodovina

Vključeni novi prevodi in nekaj manjših popravkov.

Dodal poglavje o uporabnih ukazih, ki ga je ponovno napisal Kess.

Upošteval še nekaj popravkov in predlogov.

Dodani primeri pri primerjavi nizov.

v0.8 opustil verzije, domnevam, da je datum dovolj.

v0.7 Popravki ter napisanih nekaj starih TO-DO.

v0.6 Manjši popravki.

v0.5 Dodal poglavje o preusmeritvi.

v0.4 je izginila s svojega mesta zaradi mojega nekdanjega šefa in ta spis je našel mesto tam, kjer mora biti: www.linuxdoc.org.

prej: ne spominjam se, poleg tega pa nisem uporabljal ne rcs ne cvs :(

14.5 Še več virov

Uvod v bash (pod BE) *http://org.laol.net/lamug/beforever/bashtut.htm*

Bourne Shell Programming http://207.213.123.70/book/