Robert Meolic
meolic@uni-mb.si

Skriptni jezik lupine Bash

interno gradivo za predmet VSO, 2006/07

1. LITERATURA

Pri sestavljanju gradiva o skriptnem jeziku lupine Bash sem uporabljal internet in naslednji dve knjigi:

- EricFoster-Johnson, John C. Welch, Micah Anderson: Beginning Shell Scripting, Wiley Publishing Inc., 2005
- Ken O. Burtch: Linux Shell Scripting with Bash, Sams Publishing, 2004

2. UVOD

Naš prvi program v Bashu bo imel samo dve vrstici, ki ju vpišemo v datoteko `hello.sh`.

```
#!/bin/sh
echo "Hello World"
```

S prvo vrstico operacijskemu sistemu povemo, da mora datoteko izvajati interpreter sh. Če je na voljo bash, je običajno nastavljeno tako, da se namesto sh požene bash. Lahko pa namesto `#!/bin/sh` napišemo `#!/bin/bash` in tako izrecno zahtevamo interpreter bash. Druga vrstica na ekran izpiše pozdrav. Da program lahko izvedemo, moramo najprej datoteko narediti izvršljivo.

```
$ chmod 700 hello.sh
$ ./hello.sh
```

Z zadnjim stavkom program poženemo. Pri tem navedemo celotno pot do programa - v tem primeru se program nahaja v trenutnem imeniku, kar zapišemo z eno piko.

Kot drug primer vzamimo program, ki prekopira vse datoteke v trenutnem imeniku v imenik `trash`, nato pa ta imenik skupaj z njegovo vsebino zbriše.

```
#!/bin/bash
mkdir trash
cp * trash
rm -fr trash
```

3. SPREMENLJIVKE IN ARGUMENTI

Vse **spremenljivke** v bashu so nizi. Spremenljivk ni potrebno deklarirati, spremenljivka se ustvari, ko ji dodelimo vrednost. Paziti moramo le, da niti levo niti desno od enačaja ni nobenega presledka! Paziti moramo tudi na to, da bash loči med malimi in velikimi črkami. Splošno razširjen stil pisanja skript je tak, da imena spremenljivk pišemo z velikimi črkami.

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

Druga vrstica ustvari spremenljivko `STR`, ki je niz z vrednostjo `Hello World!`. Ko spremenljivko uporabimo, pred njenim imenom dodamo znak `$`. Včasih lahko pride pri imenih spremenljivk do dvoumnosti, takrat damo ime spremenljivke v zavite oklepaje.

```
#!/bin/bash
STR=INZENIR
STRING=VAJE
echo $STRING
echo ${STR}ING
```

Znake, ki imajo poseben pomen, vnesemo tako, da pred njimi zapišemo znak \

Bash vse spremenljivke obravnava kot nize znakov. Zato zna tudi precej operacij nad nizi. Nekatere med njimi so naslednje:

- `${#string}` - vrne dolžino niza
- `${string:position}` vrne podniz $string od mesta $position
- `${string:position:length}` vrne podniz dolžine $length znakov v nizu $string od mesta $position

Primer:

```
#!/bin/bash
ST=0123456789
echo ${#ST}
echo ${ST:6}
echo ${ST:6:2}
```

Za programerje nevajene skriptnih jezikov so nenavadne naslednje možnosti za delo s spremenljivkami:

- `${var-default}` - če spremenljivka var nima vrednosti, uporabi default,
- `${var=default}` - če spremenljivka var še nima vrednosti, postavi njeno vrednost na default,
- `${var+value}` - če spremenljivka var ima vrednost, uporabi value, sicer pa prazen niz,
- `${parameter?msg}` - če spremenljivka var ima vrednost, jo uporabi, sicer izpiši msg.

V zvezi s spremenljivkami je tudi ukaz `export`. Ukaz `export` spremenljivko izvozi tako, da postane dostopna potomcem procesa. Če proces potomec spremeni vrednost spremenljivke, se sprememba ne odrazi pri procesu očetu.

Če so v nizu samo števke, lahko s spremenljivko računamo, kot da bi imeli število. Pri tem uporabljamo ukaz `let`, oglate oklepaje ali pa dvojne okrogle oklepaje.

```
#!/bin/bash
X=2
let Y=$X+2*4
Z=$[$X+2*4]
W=$(($X+2*4))
echo "X = $X"
echo "Y = $Y"
echo "Z = $Z"
echo "W = $W"
```

Podprte so aritmetične operacije seštevanje (+), odštevanje (−), množenje (*), celoštevilčno deljenje (/) in ostanek pri deljenju (%).

Če želimo spremenljivki prirediti niz, ki vsebuje presledke ali druge posebne znake, moramo niz pisati med enojnimi ali dvojnimi narekovaji. Razlika med enojnimi in dvojnimi oklepaji je v tem, da se pri dvojnih narekovajih spremenljivke v nizu nadomestijo s svojimi vrednostmi, pri enojnih narekovajih pa ne!

```
#!/bin/bash
VAR="test string"
NEWVAR1='Value of var is $var'
NEWVAR2="Value of var is $var"
echo $NEWVAR1
echo $NEWVAR2
```

Poleg enojnih in dvojnih narekovajev lahko uporabimo tudi vzvratne narekovaje `` `...` ``, ki so drugačni od enojnih narekovajev `'...'`. Z njim obdamo ukaze. Ukaz napisan v vzvratnih narekovajih se izvede, v nadaljevanju pa se uporablja dobljen rezultat.

```
#!/bin/bash
LIST=`ls`
echo $LIST
```

Vzvratni narekovaje pridejo prav, če želimo v ukazni vrstici imeti izpisan trenutni imenik. To dosežemo tako, da ustrezno spremenimo spremenljivko `PS1`:

```
PS1='`pwd`>'
```

Enak učinek dosežemo tudi z uporabo okroglih oklepajev tako, da zapišemo `$(command)`. Način z vzvratnimi oklepaji je bolj pogosto uporabljen, ker ga zna izvesti tudi interpreter sh.

Za interakcijo z uporabnikom je na voljo ukaz `read`.

```
#!/bin/bash
echo -n "Enter name of file to delete: "
read FILE
echo "Type 'y' to remove it, 'n' to change your mind ... "
rm -i $FILE
echo "That was YOUR decision!"
```

Tretja vrstica ustvari spremenljivko `FILE`, njena vrednost je niz znakov, ki ga vnese uporabnik.

Ukaz read pozna nekaj parametrov, med katerimi sta najbolj zanimiva `-s`, s katerim izklopimo izpisovanje vnesenih znakov `-n`, s katerim povemo, da se skripta nadaljuje po določenem številu vnesenih znakov. Brez parametra `-n` se vnos zaključi takrat, ko pritisnemo enter. Naslednji ukaz počaka, da uporabnik stisne eno tipko in potem nadaljuje izvajanje:

```
read -s -n 1 choice
```

Skriptom lahko ob zagonu podamo argumente, ki se obravnavajo pozicijsko:

- `$0` je ime skripte,
- `$1` je prvi argument,
- `$2` je drugi argument,
- `${10}` je deseti argument itd.

Poseben pomen povezan z argumenti imajo tudi spremenljivke

- `$#` (število argumentov)
- `$*` (vsi argumenti skupaj kot en string)
- `$@` (vsi argumenti skupaj, vendar kot zaporedje stringov)

4. KRMILNI STAVKI

Osnovna oblika **pogojnega stavka** je naslednja:

```
if [ pogoj ]
then
 statements
elif [ pogoj ]
then
 statements
else
 statements
fi
```

Stavka `elif` (else if) in `else` nista obvezna. Oglati oklepaji okoli pogoja so obvezni. Paziti moramo, da je med oglatim oklepajem in pogojem, ter med pogojem in oglatim zaklepajem vsaj en presledek. Pogoj, ki ga uporabimo v stavku `if`, ima lahko eno od naslednjih oblik:

- primerjava nizov,
- numerična primerjava,
- datotečni operatorji,
- logični operatorji.

Operatorji pri primerjavi nizov so:

- s1 = s2 (ali sta niza enaka?)
- s1 != s2 (ali sta niza različna?)
- -n s1 (ali je dolžina niza večja od 0?)
- -z s1 (ali je dolžina niza enaka 0?)

Primer primerjave nizov je naslednji:

```
if [ "$FILENAME" != "$NEWNAME" ] then echo "Imeni sta enaki"
```

Operatorji pri primerjavi števil so:

- n1 -eq n2 (enako)
- n1 -ne n2 (različno)
- n1 -gt n2 (večje)
- n1 -ge n2 (večje ali enako)
- n1 -lt n2 (manjše)
- n1 -le n2 (manjše ali enako)

Datotečni operatorji so:

- -d filename (preveri ali je direktorij)
- -f filename (preveri ali je datoteka)
- -e filename (preveri ali datoteka obstaja)
- -r filename (preveri ali je dovoljeno branje)
- -s filename (preveri ali je dolžina datoteke večja kot 0)
- -w filename (preveri ali je dovoljeno pisanje)
- -x filename (preveri ali je dovoljeno izvajanje)

Logični operatorji so:

- ! (logična negacija, NOT)
- -a (logična konjunkcija, AND)
- -o (logična disjunkcija, OR)
- && (logična konjunkcija, AND)
- || (logična disjunkcija, OR)

Namesto večkratnega stavka `if` lahko uporabimo stavek `case`. Blok stavkov se končuje z dvojnim podpičjem. Splošna oblika stavka `case` je naslednja:

```
case $var in
val1)
 statements;;
val2)
 statements;;
*)
 statements;;
esac
```

Blok označen kot `*)` se izvrši, če se ne izvrši noben drug blok.

Zanke tvorimo s stavki `for`, `while` in `until`.

Poznamo dve obliki stavka `for`. Pri prvi obliki se stavek `for` obnaša enako kot v programskem jeziku C. Splošna oblika stavka `for` je v tem primeru naslednja:

```
for (( EXPR1 ; EXPR2 ; EXPR3 ))
do
 statements
done
```

Najprej se ovrednoti izraz `EXPR1`. Nato se ovrednoti stavek `EXPR2`. Če je `EXPR2` resničen, se izvršijo podani stavki in potem se ovrednoti še `EXPR3`. Vsi koraki razen prvega se nato ponavljajo, dokler je stavek `EXPR2` resničen.

Druga oblika zanke `for` je podobna kot v drugih skripnih jezikih in izgleda takole:

```
for var in list
do
 statements
done
```

Seznam list je string, v katerem so posamezne vrednosti ločene s presledkom. Telo zanke se izvrši za vse vrednosti var s seznama. Če seznam list zaključimo s podpičjem, potem je besedica `do` lahko zapisana v isti vrstici.
Primer zanke `for` je tukaj:

```
#!/bin/sh
SUM=0
for NUM in 1 2 3 4 5; do
  let SUM = $SUM + $NUM"
done
echo $SUM
```

Če seznam list izpustimo, deluje zanka `for` nad vrednostmi argumentov, torej dobi spremenljivka `var` po vrsti vrednosti `$1`, `$2`, `$3`,…

Stavek `while` je namenjen izvajanju bloka ukazov, dokler je določen pogoj resničen. Zanka se neha izvajati takoj, ko pogoj postane neresničen. Splošna oblika je naslednja:

```
while [ pogoj ]
do
 statements
done
```

Stavek `until` je zelo podoben stavku `while`. Zanka se izvaja, dokler pogoj ne postane resničen. Splošna oblika stavka `until` je naslednja:

```
until [ pogoj ]
do
 statements
done
```

Tudi pri stavkih `while` in `until` moramo paziti, da je med oglatim oklepajem in pogojem, ter med pogojem in oglatim zaklepajem vsaj en presledek.

Pri zankah sta koristna tudi ukaza `continue` in `break`. Ukaz `continue` povzroči skok na naslednjo iteracijo zanke, ostali ukazi v trenutni iteraciji se preskočijo. Ukaz `break` prekine izvajanje zanke in nadaljuje za zanko.

V Bashu lahko uporabljamo tudi polja. Indekse elementov podajamo v oglatih oklepajih. Prvi element v polju ima indeks 0. Polje lahko definiramo tudi tako, da naštejemo njegove elemente v okroglih oklepajih. Največje število elementov je 1024. Do posameznih elementov pridemo z uporabo `${polje[i]}`, do vseh elementov pa z uporabo `${polje[*]}`. Tukaj je primer:

```
#!/bin/sh
PET[0]=dog
PET[1]=cat
PET[2]=fish
echo ${PET[*]}
PET=(cat fish dog)
echo ${PET[*]}
```

Polja lahko kombiniramo z zanko `for` tako, da uporabimo naslednje stavke:

```
for x in ${polje[*]}
do
 statements
done
```

Skripte v Bashu lahko vsebujejo funkcije. Z uporabo funkcij program razbijemo v manjše dele. Uporaba funkcij prispeva tudi k lažji berljivosti programa. Funkcija mora biti definirana, preden jo kličemo. Tukaj je primer.

```
#!/bin/sh

# funkcija hello
hello()
{
  echo "Hello world!"
}

# funkcija konec
konec()
{
  echo $1
  echo $2
  echo $3
}

# glavni program
echo "Glavni program"
hello
konec ena dva tri
echo "Nasvidenje."
```

Pri pisanju skript včasih potrebujemo naključno število. V Bashu to ni noben problem, saj spremenljivka $RANDOM v vsakem trenutku vsebuje naključno število od vključno 0 do vključno 32767.

Pri programiranju pogosto vnesemo napake in pisanje skript ni pri tem nobena izjema. Interpreter Bash ima dva parametra namenjena razhroščevanju in iskanju napak:

- `-v` prikaže vsako vrstico, tako kot je napisana, tik preden se vrstica izvede
- `-x` prikaže vsako vrstico, z ovrednotenimi spremenljivki, tik preden se vrstica izvede

Parametra navedemo v prvi vrstici, lahko ju uporabimo sočasno. Oglejmo si primer:

```
#!/bin/sh -vx
STR=Hello World!”
echo $STR
```

Dobimo naslednji izpis:

```
#!/bin/sh -vx
STR="Hello World!"
+ STR='Hello World!'
echo $STR
+ echo Hello 'World!'
Hello World!
```

Na koncu naštejmo še nekaj drugih ukazov, ki jih pozna lupina Bash. Podrobnejši opis vsakega ukaza dobimo, če uporabimo ukaz man.

- /usr/bin/yes
- /bin/true
- /bin/false
- /bin/doexec
- /usr/bin/xargs

Tukaj pa sta še dve praktični skript za Bash.

```
# PROGRAM: PREPROST MENU
#
#!/bin/sh

while [ true ]
do
  echo
  echo "Menu"
  echo "===="
  echo "D: izpise datum"
  echo "W: izpise podatke o trenutno prijavljenih uporabnikih"
  echo "P: izpise pot do trenutnega imenika"
  echo "Q: konec programa"
  echo
  read -s -n 1 choice
  case $choice in
    D | d) echo "Danasnji datum"; date ;;
    W | w) echo "Uporabniki"; who ;;
    P | p) echo "Trenutni imenik"; pwd ;;
    Q | q) break ;;
    *) echo "'$choice' ni pravilna izbira" ;;
  esac
done
```

```
# PROGRAM: SPREMENI IMENA DATOTEK TAKO, DA VSEBUJEJO LE MALE CRKE
#
#!/bin/sh

for FILE in *
do
  FILENAME=`basename "$FILE"`
  NEWNAME=`echo "$FILENAME" | tr A-Z a-z`
  if [ "$FILENAME" != "$NEWNAME" ]
  then
    echo "$FILENAME --> $NEWNAME"
    mv "$FILENAME" "$NEWNAME"
  fi
done
```

5. OBDELAVA TEKSTOVNIH DATOTEK

Najpomembnejši element obdelave tekstovnih datotek je iskanje določenega niza. Osnovni ukaz za iskanje nizov v lupini Bash je `grep`. Omogoča bolj kompleksna iskanja kot ukaza `FIND` in `FINDSTR` v lupini Windows command. Tukaj je nekaj preprostih primerov, ki kažejo na podobnost z ukaza `grep` z ukazoma `FIND` in `FINDSTRING`.

Poišči in izpiši vse vrstice, ki vsebujejo niz „skripta“ v datoteki `Besedilo.txt`, ob tem izpiši tudi številke vrstic:

```
grep -n "skripta" Besedilo.txt
```

Poišči in izpiši vse vrstice, ki vsebujejo niz „skripta“ v datoteki `Besedilo.txt`, pri čemer ne loči med malimi in velikimi črkami:

```
grep -i "skripta" Besedilo.txt
```

Poišči in izpiši vse vrstice, ki NE vsebujejo znaka „.“ v datoteki `Besedilo.txt`:

```
grep -v "\." Besedilo.txt
```

Preštej in izpiši število vrstic, ki vsebujejo niz „skripta“ v vseh datotekah s končnico txt v trenutnem imeniku:

```
grep -c "skripta" *.txt
```

Poišči in izpiši vse vrstice, ki vsebujejo niz „ime“ kot samostojno besedo v datoteki `Besedilo.txt` (levo in desno od besede so lahko presledki ali pa posebni znaki kot npr. vejica, enačaj itd.):

```
grep "\<ime\>" Besedilo.txt
```

Podobno kot pri lupini Windows shell, lahko tudi v lupini Bash vrstice uredimo po abecedi z ukazom sort. Računalnik najprej čaka, da vnesemo besedilo. Vnos besedila v lupini Bash končamo tako, da pritisnemo kombinacijo tipk CTRL+D. Nato se izpišejo vnesene vrstice urejene po abecedi.

Prikazani primeri so želeli pokazati na podobnost obdelave tekstovnih datotek v lupinah Windows command in Bash. Vendar pa je lupina Bash mnogo močnejša, saj imamo na voljo še številne druge ukaze, ki nam zelo olajšajo obdelavo tekstovnih datotek. Za začetek omenimo ukaza `head` in `tail`. Ukaz `head` izpiše prvih 10 vrstic tekstovne datoteke, ukaz `tail` pa zadnjih 10 vrstic tekstovne datoteke. Če želimo, lahko nastavimo drugačno število vrstic, ki je večje ali manjše od 10. Tukaj sta dva primera:

```
head Imena.txt
tail -n 5 Imena.txt
```

Ukaz `tail` ima tudi zanimivo kretnico `-f`, ki jo uporabimo, če se datoteka spreminja (npr. kakšen dnevnik z napakami). V tem primeru se izpis sproti posodablja tako, da je vedno prikazanih zadnjih 10 vrstic.

Naslednji enostaven a uporaben ukaz je `wc`. Z njim dobimo statistiko o tekstovni datoteki. Preštejemo lahko število znakov, besed, vrstic in število zlogov ter ugotovimo dolžino najdaljše vrstice. Parametri ukaza `wc` so naslednji:

- `-m` (število znakov)
- `-w` (število besed)
- `-l` (število vrstic)
- `-c` (število zlogov)
- `-L` (dolžina najdaljše vrstice)

Z ukazom `cut` izpišemo samo določen del vsake vrstice. Kot kriterij lahko podamo zaporedno številko znaka ali pa polja, pri čemer podamo tudi, s katerim znakom so posamezna polja ločena med seboj. Ločilni znak je vedno samo eden in če se v datoteki pojavita dva zaporedoma (npr. dva presledka zaporedoma) se šteje, da je vmes prazno polje.

Izpiše samo prvi znak vsake vrstice v datoteki `Imena.txt`:
```
cut -c 1 Imena.txt
```

Izpiše prvih 8 znakov vsake vrstice v datoteki `Imena.txt`:
```
cut -c 1-8 Imena.txt
```

Izpiše drugo, tretjo, četrto in sedmo polje vsake vrstice v datoteki `Imena.txt`, posamezna polja so ločena z vejicami:
```
cut -f 2-4,7 -d "," Imena.txt
```

Za konec pa si poglejmo še zelo uporaben ukaz `tr`. Z njim spremenimo oz. zbrišemo določene znake v datoteki. Ker po privzetem ukaz `tr` deluje nad besedilom, ki ga natipkamo, moramo za obdelavo datotek uporabiti preusmeritve (podrobno so opisane v naslednjem poglavju).

Zamenja vse velike črke v datoteki `Imena.txt` z malimi (šumnike moramo podati posebej :-)
```
tr A-ZČŠ a-zčš < Imena.txt
```

Vse večkratne presledke v datoteki `Imena.txt` nadomesti z enim samim presledkom:
```
tr -s " " < Imena.txt
```

6. PREUSMERITVE

Preusmeritve so preprost in zelo koristen mehanizem za kombiniranje ukazov. V splošnem obstajata dva načina delovanja ukazov:

- ukaz bere podatke s tipkovnice oz. piše rezultate na ekran,
- ukaz bere podatke iz datoteke oz. piše rezultate v datoteko.

Bolj zanimiv je prvi primer. Če ukaz bere podatke s tipkovnice lahko vzamemo poljubno tekstovno datoteko in mu povemo, da naj se obnaša tako, kot da mu jo bomo natipkali. To naredimo tako, da uporabimo operator „<“. Če ukaz piše na ekran, lahko njegov izhod preusmerimo v datoteko. To dosežemo tako, da uporabimo operator „>“ ali pa „>>“. Razlika me njima je v tem, da operator „>“ tvori novo datoteko s podanim imenom, morebitno obstoječo z nekim imenom pa zbriše. Operator „>>“ pa tvori novo datoteko le v primeru, da datoteka z navedenim imenom še ne obstaja, drugače pa doda izpis na konec obstoječe datoteke. Poseben primer nastopi, če pride med izvajanjem ukaza do kakšne napake. V tem primeru se obvestilo o napaki izpiše na ekran ne glede na to, ali ukaz piše na ekran ali v datoteko. Če želimo tudi obvestila o napaki preusmeriti v datoteko uporabimo operator „2>“ ali pa „2>>“.

Primeri:

```
sort < Imena.txt
sort < Imena.txt > Urejeno.txt
dir Vaje > Seznam.txt 2> Napaka.txt
cat Imena.txt | sort
```

Za razliko od lupine Windows command lahko v lupini Bash izpis in napake preusmerimo v isto datoteko, vendar pa rezultat morda ne bo uporaben. Lupina Bash ne pozna preusmeritve na odložišče.

Preusmeritve lahko v lupini bash izkoristimo za branje in obdelavo tekstovne datoteke vrstico po vrstico. Najpreprosteje to naredimo z uporabo cevi po naslednjem vzorcu (bodite pozorni, da je znak | zadnji znak v prvi vrstici):

```
cat Studenti.txt |
while read line
do
  ... obdelaj vrstico $line ...
done
```

Nekoliko bolj kompliciran pa izgleda rešitev, pri kateri datoteko odpremo za branje in potem iz nje beremo vrstico po vrstico. Primer je naslednji:

```
exec 5< Studenti.txt
while read -u 5 line
do
  ... obdelaj vrstico $line ...
done
```

V prikazanem primeru smo za branje odprli datoteko Studenti.txt, kot kazalec datoteke (angleško "file descriptor") pa smo uporabili številko 5. Kazalec datoteke je lahko katerakoli številka, vendar pa so številke od 0 do 4 rezervirane in jih uporabnik ne sme poljubno uporabljati. Kretnica `-u` pri ukazu `read` pove, da naj se branje namesto s tipkovnice (ki ima kazalec datoteke enak 0) izvrši iz datoteke, ki ima kazalec datoteke enak 5.