Operativni sistemi I

Vežbe 6

OSNOVE SHELL PROGRAMIRANJA

Promenljive

Na Linux sistemima postoje dva tipa promenljivih:

- **sistemske promenljive**, koje kreira i održava sam operativni sistem. Ne preporučuje se promena njihovog sadržaja (zašto?). Ovaj tip promenljivih definiše se strogo velikim slovima,
- **korisnički definisane promenljive** (User defined variables - UDV), koje kreiraju i održavaju korisnici. Ovaj tip promenljivih se obično definiše malim slovima;

U shell programiranju promenljive se ne deklarišu za specifični tip podataka - dovoljno je dodeliti vrednost promenljivoj i ona će biti alocirana prema toj vrednosti. U *Bourne Again Shell*-u, promenljive mogu sadržavati brojeve, karaktere ili nizove karaktera.

Važnije sistemske promenljive

Sistemske promenljive mogu se videti pozivom komande set:

```
$ set
BASH=/bin/bash
HOME=/home/jsmith
PATH=/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/games
PS1=[\u@\h \W]\$
PWD=/tmp/junk
SHELL=/bin/bash
USERNAME=jsmith
...
```

Neke od njih su:

- BASH lokacija komandnog interpretera
- HOME home directorijum korisnika
- PATH putanja u kojoj se traže izvršne datoteke
- PS1 podešavanje prompta
- PWD tekući direktorijum
- SHELL ime komandnog interpretera
- USERNAME ime korisnika koji je u ovom režimu trenutno prijavljen na sistem.

Pojedinačno, sadržaj promenljive može se videti pozivom:

```
$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/games
```

Definisanje korisničkih promenljivih

Svaka promenljiva je univerzalna i nema nikakvu deklaraciju tipa (integer, float, string) i definiše se na sledeći način:

```
variablename=value
```

Primer.

```
$ br=10
```

Prilikom definisanja, odnosno dodele vrednosti, potrebno je primeniti sledeće konvencije o imenima promenljivih:

- **Ime promenljive mora početi alfanumeričkim karakterom ili donjom crtom '_'** (underscore character) praćenim jednim ili više alfanumeričkih karaktera.

  Primer.

  Korektne promenljive su: HOME, SYSTEM_VERSION, br, _ime;

- **Prazne karaktere ne treba stavljati ni sa jedne strane znaka jednakosti** prilikom dodele vrednosti promenljivim.

  Primer.

  ```
  $ br =10 # neispravno - prazni karakteri
  $ br= 10 # neispravno - prazni karakteri
  $ br = 10 # neispravno - prazni karakteri
  $ br=10 # ispravno
  ```

- **case sensitive**

  ```
  $ bR=20
  $ Br=30
  $ echo $bR
  20
  ```

- Može se definisati **promenljiva nulte dužine (NULL variable)**, odnosno promenljiva koja nema vrednost u trenutku definisanja. Vrednosti ovih promenljivih se ne mogu prikazati komandom echo, sve dok sim se ne dodeli vrednost;

  ```
  $ br=
  $ ime=""
  ```

- **Imena promenljivih ne smeju sadržati specijalne znake** (poput ? i *).

Primer 1.

```
$ echo $ime # prikazuje vrednost promenljive ime
johnny
$ echo ime # prikazuje string ime
ime
```

Primer 2.

```
$ x=10
$ xn=abc
$ echo $x $abc
10 abc
```

Primer 3.

Definisati dve promenljive, x i y, sa vrednostima 20 i 5, respektivno, i promenljivu z kao njihov

količnik. Rezultat prikazati na ekranu, u jednom redu.

```
$ x=20
$ y=5
$ z=`expr $x / $y`
$ echo x/y=$z
x/y=4
```

Komanda expr

```
$ expr 6 + 3   # expr posmatra 6 + 3 kao matematički izraz
9
```

Komanda expr određuje rezultat neke matematičke operacije. Sintaksa komande expr je:

```
expr op1 operacija op2
```

gde su op1 i op2 celi brojevi, a operator +, -, *, /, &. Rezultat operacije je ceo broj. Argumenti op1, op2 i operator se moraju razdvojiti praznim karakterom.

```
$ expr 6 + 3   # ispravno
9
```

Specijalne promenljive

Ove promenljive su rezervisane za specifične funkcije. Na primer, karakter $ reprezentuje proces ID (PID) tekućeg *shell*-a.

Ako se ukuca

```
$ echo $?
```

u *shell* promptu, dobiće se izlazni status poslednje komande.

Special variables that you can use in your bash scripts.

Promenljiva	**Funkcija**
?	Izlazni status prethodne komande
$	PID tekućeg shell procesa
-	Opcije sa kojima je pozvan aktuelni *shell*
!	PID poslednje komande koja radi u pozadini
0	ime fajla tekućeg skripta
1-9	argumenti komandne linije dati tekućem skriptu **$1** je prvi argument **$9** deveti
_	poslednji argument dat prethodno pozvanoj komandi

Čitanje podataka sa ulaza - read

Komanda read se koristi za čitanje ulaznih podataka sa tastature i memorisanje unete vrednosti u promenljivu. Sintaksa komande je:

```
$ read varible1, varible2,...varibleN
```

Primer.

```
#
# ss2.sh: upotreba komande read
#
echo "Unesite podatak:"
read var1
echo "Uneli ste: $var1"
```

Pokretanje:

```
# bash ss2.sh
Unesite podatak: 123
Uneli ste: 123
```

Komande, argumenti i izlazni status

Izlazni status komandi

Nakon izvršenja Linux komande vraćaju vrednost na osnovu koje se može odrediti da li je komanda izvršena uspešno ili ne. Ako je povratna vrednost 0, komanda je izvršena uspešno. Ako je povratna vrednost različita od 0 (veća od 0), komanda se nije uspešno završila, a taj broj predstavlja neku vrstu dijagnostičkog statusa koja se naziva izlazni status.

```
$ rm plumph
rm: cannot remove `plumph': No such file or directory
$ echo $?
1 # izlazni status 1 -> komanda izvršena s greškom
$ date
$ echo $?
0 # izlazni status 0 -> komanda izvršena bez greške
```

Argumenti

Komanda može biti zadata bez parametara (npr. **date, clear, who**), kao i sa jednim ili više parametara (**ls -l, ls -l /etc, mount -t ntfs /dev/hda1 /mnt/winc**).

Promenljiva **$#** memoriše broj argumenata specifirane komandne linije, a **$*** ili **$@** upućuju na sve argumente koji se prosleđuju shell programu.

Komandni argumenti se na isti način mogu zadati i shell script-u.

Na primer:

```
$ ss3.sh arg1 arg2 arg3
```

Argumente ovako pozvanog script-a se može pamte sledeće promenljive:

```
$0 je ime programa - ss3.sh
$1 je prvi komandni argument - arg1
$2 je drugi komandni argument - arg2
```

```
$3 je treći komandni argument - arg3
$# je broj komandnih argumenta - 3
$* su svi komandni argumenti. $* se proširuje u `$1,$2...$9` - arg1 arg2 arg3.
```

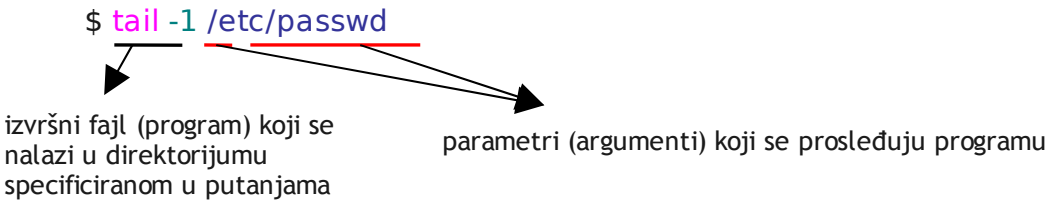
Primer.

```
$ df
$ less /etc/passwd
$ ls -l /etc
$ mount -r /dev/hda2 /mnt/winc
```

ime programa **$0**	broj argumenata **$#**	**argumenti**		
		$1	**$2**	**$3**
df	0			
less	1	/etc/passwd		
ls	2	-l	/etc	
mount	3	-r	/dev/hda2	/mnt/winc

Primer.

```
#
# ss3.sh: korišcenje argumenata komandne linije
#
echo "Ukupan broj argumenata komandne linije: $#"
echo "$0 je ime programa, a $1 je prvi argument."
echo "Svi argumenti su redom: $*"
Pokretanje:
$ bash ss3.sh arg1 arg2 arg3
```



Kontrola toka

Rad sa promenljivama je koristan, ali mora se proširiti odgovarajućom kontrolom toka u vidu petlji i uslova kako bi se dobio zaista koristan skript koji nešto radi. Uobičajene su dve vrste kontrole toka:

- kondicione i
- iterativne.

Kondiciona kontrola toka pomoću if-then izraza

U opštem slučaju, if-then konstrukcija izgleda ovako:

```
if [uslov]
then
    naredbe
fi
```

OBRATITI PAŽNJU NA RAZMAKE I NOVE REDOVE! If-then blok se završava naredbom **fi** (obrnuto od **if**).

Primer.

```
#!/bin/bash
echo 'Guess the secret color'
read COLOR
if [ $COLOR = 'purple' ]
then
    echo 'You are correct.'
fi
```

Više uslova može se dodati korišćenjem klauzule **else**:

Primer.

```
#!/bin/bash
echo 'Guess the secret color'
read COLOR
if [ $COLOR = 'purple' ]
then
echo 'You are correct.'
else
echo 'Your guess was incorrect.'
fi
```

Takođe, prisutna je i klauzula **elif**:

```
#!/bin/bash
echo 'Guess the secret color'
read COLOR
if [ $COLOR = 'purple' ]
then
    echo 'You are correct.'
elif [ $COLOR = 'blue' ]
then  echo 'You’re close.'
else
    echo 'Your guess was incorrect.'
fi
```

Instrukcija **elif** može biti proizvoljno mnogo, što je posebno zgodno kod npr. ispitivanja poslatih argumenata komandne linije. Evo još nekih primera korišćenja **if** klauzule u ugnežđenom obliku i složenih uslova dobijenih logičkim operatorima:

Multiple Conditions:

```
if [ condition1 ]
then
     if [ condition2 ]
     then
           some action
     fi
fi
```

Ili isto to, ali jednostavnije, korišćenjem logičkih operatora:

```
if [ condition1 && condition2 ]
then
     some action
fi
```

Isto kao u C-u, postoji i logički operator "||":

```
if [ condition1 || condition2 ]
then
     some action
fi
```

što je ekvivalentno sledećem:

```
if [ condition1 ]
then
     some action
elif [ condition2 ]
then
     the same action
fi
```

Komanda test

Komanda test se koristi za evaluaciju uslova. Naime, primećuje se da svi navedeni primeri uslova uključuju **uglaste zagrade** oko uslova koji se izračunava. **Uglaste zagrade su praktično ekvivalentne komandi test**. Na primer, uslov iz prethodnog skripta bi mogao da glasi i ovako:

```
if ( test $COLOR = 'purple' )
```

Ovaj koncept je važan iz razloga što se veliki broj opcija komande test može iskoristiti za testiranje različitih uslova. Na primer, provera da li neki fajl postoji na fajl sistemu:

```
if ( test -e filename )
```

što je isto što i

```
if [ -e filename ]
```

U slučaju da fajl postoji , vraća se true, tj. izlazni status 0, a ako ne postoji vraća se false, tj. 1. Sledeća tabela daje pregled najkorišćenijih opcija **test** komande:

Opcija	**Uslov**
-d	Specificirani fajl postoji i u pitanju je direktorijum
-e	Specificirani fajl postoji

-f	Fajl postoji i u pitanju je regularni fajl (a ne direktorijum ili drugi specijalni fajl)
-G	Fajl postoji i vlasništvo je efektivnog GID-a
-nt	Fajl je noviji od drugog navedenog fajla (sintaksa je *file1 -nt file2*).
-ot	Fajl je stariji od drugog navedenog fajla (sintaksa je *file1 -nt file2*).
-O	Korisnik koji izvršav akomandu je vlasnik fajla
-r	Korisnik koji izvršava komndu ima dozvolu čitanja nad fajlom
-s	Fajl postoji i nije prazan
-w	Korisnik koji izvršava komandu ima dozvolu pisanja
-x	Korisnik koji izvršava komandu ima dozvolu izvršavanja

Sa svim ovim opcijama, komanda će vratiti ili **true** ili **false.**

Operatori poređenja

Operatori poređenja rade isto kao i u bazičnoj aritmetici. Stringovi se takođe mogu porediti, standardno u abecednom redosledu.

=	jednako
!=	različito
>	veće od
<	manje od

Case izraz

Opšti format case komande je:

```
case expression in
pattern1)
     action1
     ;;
pattern2)
     action2
     ;;
pattern3)
     action3
     ;;
esac
```

Treba primetiti da se svaka sekcija zaključuje duplim znakom ";". Case izrazi su odlični za razvrstavanje argumenata komandne linije. Sledeći skrip je uzet samo kao primer, a služi za kontrolu NFS (*Network File System*) servisa.

```
# See how we were called.
case "$1" in
start)
     # Start daemons.
     action $"Starting NFS services: " /usr/sbin/exportfs -r
     echo -n $"Starting NFS quotas: "
```

```
        daemon rpc.rquotad
        echo
        echo -n $"Starting NFS mountd: "
        daemon rpc.mountd $RPCMOUNTDOPTS
        echo
        echo -n $"Starting NFS daemon: "
        daemon rpc.nfsd $RPCNFSDCOUNT
        echo
        touch /var/lock/subsys/nfs
        ;;
    stop)
        # Stop daemons.
        echo -n $"Shutting down NFS mountd: "
        killproc rpc.mountd
        echo
        echo -n $"Shutting down NFS daemon: "
        killproc nfsd
        echo
        action $"Shutting down NFS services: " /usr/sbin/exportfs -au
        echo -n $"Shutting down NFS quotas: "
        killproc rpc.rquotad
        echo
        rm -f /var/lock/subsys/nfs
        ;;
    status)
        status rpc.mountd
        status nfsd
        status rpc.rquotad
        ;;
    restart)
        echo -n $"Restarting NFS services: "
        echo -n $"rpc.mountd "
        killproc rpc.mountd
        daemon rpc.mountd $RPCMOUNTDOPTS
        /usr/sbin/exportfs -r
        touch /var/lock/subsys/nfs
        echo
        ;;
    reload)
        /usr/sbin/exportfs -r
        touch /var/lock/subsys/nfs
        ;;
    probe)
        if [ ! -f /var/lock/subsys/nfs ] ; then
        echo start; exit 0
        fi
        /sbin/pidof rpc.mountd >/dev/null 2>&1; MOUNTD="$?"
        /sbin/pidof nfsd >/dev/null 2>&1; NFSD="$?"
        if [ $MOUNTD = 1 -o $NFSD = 1 ] ; then
        echo restart; exit 0
        fi
        if [ /etc/exports -nt /var/lock/subsys/nfs ] ; then
        echo reload; exit 0
        fi
        ;;
    *)
        echo $"Usage: $0 {start|stop|status|restart|reload}"
        exit 1
    esac
```

Iterativna kontrola - while petlja

Ponaša se isto kao u jeziku C:

```
#!/bin/bash
echo 'Guess the secret color: red, blue, yellow, purple, or orange \n'
read COLOR
while [ $COLOR != 'purple' ]
do
    echo 'Incorrect. Guess again. \n'
    read COLOR
done
echo 'Correct.'
```

Iterativna kontrola - until izraz

Razlikuje se od **while** petlje samo po uslovu. Prethodni primer bi mogao da se napiše i ovako:

```
#!/bin/bash
echo 'Guess the secret color: red, blue, yellow, purple, or orange \n'
read COLOR
until [ $COLOR = 'purple' ]
do
echo 'Incorrect. Guess again. \n'
read COLOR
done
echo 'Correct.'
```

Iterativna kontrola - for petlja

Sintaksa for petlje ne liči mnogo na sintaksu C jezika:

```
for name [in words ...];
do
    commands;
done
```

Promenljiva **name** dobija vrednost tekućeg člana liste **word**. Ako se "**in words"** izostavi u naredbi select, ili ako se specificira `in "$@"', tada će name uzimati vrednost pozicionih parametara. **Izlazni status for petlje jednak je izlaznom statusu zadnje izvršene komande u grupi commands**. Ako je lista words prazna nijedna komanda se neće izvršiti i tada će izlazni status biti 0.

```
#
# ss11.sh: upotreba for petlje
#
if [ $# -eq 0 ]
then
    echo "Greška - numericki argument nije naveden"
    echo "Sintaksa : $0 broj"
    echo "Program prikazuje tablicu množenja za dati broj"
    exit 1
fi
n=$1
for i in 1 2 3 4 5 6 7 8 9 10
do
    echo "$n * $i = `expr $i \* $n`"
done
```

Sledeći primer ilustruje upotrebu alternativne for petlje:

```
for i in `seq 1 10`
do
    echo $i
done
```

For petlja najpre kreira promenljivu **i**, a zatim joj redom dodeljuje vrednosti iz liste (u ovom slučaju numeričke vrednosti od 1 do 10). Shell izvršava echo naredbu za svaku vrednost promenljive **i**.

Naredba select

Select naredba služi za prikaz menija čije su stavke definisane u **words** i prihvatanje izbora od korisnika.

```
select name [in words ...];
do
    commands;
done
```

Lista reči se proširuje generišući listu stavki (item). Skup proširenih reči prikazuje se na standardnom izlazu za greške, pri čemu svakoj prethodi redni broj. Ako se `in words' izostavi u naredbi select, ili ako se specificira `in "$@"', tada se prikazuju pozicioni parametri. U slučaju `in "$@"' PS3 prompt se prikazuje i linije se čitaju sa standardnog ulaza. Ako se linija sastoji od broja koji odgovara jednoj od prikazanih reči tada se vrednost promenljive **name** postavlja u tu reč. Ukoliko je linija prazna reč i prompt se prikazuju ponovo. Ako se pročita EOF select komanda završava rad. Svaka druga pročitana vrednost uzrokuje da promenljiva **name** bude postavljena na nulu. Pročitana linija se čuva u promenljivoj **REPLY**.

Komande se izvršavaju posle svake selekcije sve dok se ne izvrši **break** komanda, čime se komanda **select** završava.

Primer ilustruje upotrebu naredbe select: program dozvoljava korisniku da sa tekućeg direktorijuma izabere datoteku čije će ime i indeks nakon toga biti prikazani.

```
select fname in *;
do
    echo Datoteka: $fname \($REPLY\)
    break;
done
```

Sledeći primer ilustruje kreiranje prostog menija:

```
opcije="Pozdrav Kraj"
select op in $opcije;
do
    if [ "$op" = "Kraj" ];
    then
          echo OK.
          exit
    elif [ "$op" = "Pozdrav" ];
    then
          echo Linux Rulez !
    else
          clear
          echo Opcija ne postoji.
    fi
done
```

STDIN, STDOUT, i STDERR

Svaki put kada se otvori shell, Unix otvara tri fajla koja program koristi:

- STDIN (standard in) - uglavnom tastatura terminala
- STDOUT (standard out) - uglavnom monitor terminala
- STERR (standard error) - obično je i ovo monitor terminala

Bažno je zapamtiti da se podrazumevano ulaz uzima sa tastature i štampa na ekran. Evo ponovljenog skupa operatora za redirekciju:

Operator	Akcija
>	Redirektuje STDOUT u fajl
<	Redirektuje STDIN iz fajla
>>	Dodaje STDOUT fajlu
|	Uzima izlaz iz jednog programa i šalje kao ulaz drugom
<< graničnik	Pridružuje tekući ulazni tok STDIN-u dok se ne dostigne odgovarajući graničnik

Primeri.

```
$ ls > fileList
```

STDOUT se upisuje u fajl **fileList**.

```
$ ls /home/student >> fileList
```

STDOUT se dodaje u fajl **fileList**.

```
$ ls | wc
```

Pipeline koji broji reči u izlazu komande **ls**.

```
Cat <<KRAJ
The cat
Sat on the
Mat.
KRAJ
```

Gordnji primer koristi poslednji navedeni operator redirekcije ulaza sa graničnikom. Sve dok se ne unese reč KRAJ, vrši se redirekcija iz STDIN. Komanda **cat** zatim štampa uneti sadržaj.

```
$ ls >& fileList
```

Korišćenje >& znači da se se i STDOUT i STDERR usmeravaju u navedeni fajl.

Standardni ulaz (STDIN), standardni izlaz (STDOUT) i standardni izlaz za greške (STDERR) su deskriptori datoteke kojima su dodeljeni brojevi po sledećim pravilima:

- 0 predstavlja STDIN,
- 1 predstavlja STDOUT i
- 2 predstavlja STDERR.

Još primera redirekcije:

Sledeći primer demonstrira kreiranje datoteke greperr.txt i upis poruka o greškama koje proizvodi

komanda grep u datoteku:

```
$ grep kyuss * 2> greperr.txt
```

Redirekcija STDERR u STDOUT je demonstrirana sledećim primerom. Rezultat izvršenja komande **grep** smešta se u fajl i može se naknadno videti, a poruke o greškama koje komanda grep proizvodi prikazuju se na standardnom izlazu, a to je u podrazumevanom stanju ekran;

```
$ grep kyuss * > greperr.txt 2>&1
```

Ova vrsta redirekcije je korisna za programe koji rade u pozadini, tako da se od njih očekuje da poruke ne upisuju na ekran, već u neku datoteku. Dodatno, ukoliko korisnik ne želi da vidi "feedback"komande, izlaz i poruke o greškama mogu se preusmeriti na uređaj /dev/null, kao u sledećem primerom:

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