6

SHELL PROGRAMIRANJE

Shell program (shell script) je datoteka koju čini niz Linux komandi koje se mogu izvršavati sekvencijalno ili povezati programskim strukturama tipičnim za više programske jezike, kao što su uslovni skokovi i petlje. Za razliku od programa napisanih u višim programskim jezicima (C i Pascal), shell program se ne prevodi, tj. ne zahteva se postojanje posebnog prevodioca i linkera. Komandni interpreter, koji je sastavni deo sistema, interpretira i izvršava program. Shell script funkcioniše kao MS-DOS batch datoteka, ali u odnosu na nju ima znatno više mogućnosti. Sam razvojni proces je nekonformniji u odnosu na klasični programske jezike, zato što nema programa za kontrolu toka programa (debugera) niti izvršenja korak po korak. Ulazni parametri se shell programu mogu zadati u komandnoj liniji ili interaktivno, tokom izvršenja programa, a izlaz se može formirati i prikazati na ekranu. Shell programiranje je veoma korisna tehnika kojom korisnik može kreirati specifične komande u cilju automatizacije svakodnevnih poslova.

Shell programi se pišu za određeni komandni interpreter. Korisnik može biti siguran da će shell program napisan za jedan komandni interpreter na njemu raditi ispravno - za druge komandne interpretere ne može se dati garancija o ispravnom izvršenju programa. Npr. ukoliko je program napisan za bash, niko ne može sa sigurnošću garantovati da će se on ispravno izvršiti na csh. Shell programiranje obuhvaćeno ovim poglavljem odnosi se na bash (Bourne Again Shell).

Shell programiranje obuhvata nekoliko bitnh elemenata:

- tipove podataka (koristi se univerzalni tip promenljive koji se ne deklariše, već se promenljiva automatski prilagođava tipu podataka),
- naredbe (koriste se sve UNIX komande, bilo interne bilo eksterne),
- programske strukture (uslovni skokovi, petlje i funkcije, odnosno podprogrami).

Osnovi shell programiranja

Jednostavan shell script. Pokretanje shell programa. Elementi shell programa. Sistemske i korisničke promenljive.

U nastavku teksta dat je primer jednostavnog shell programa koji štampa tekst "Goodbye, Cruel World !" na ekranu. Za pisanje shell programa može se koristiti bilo koji Linux tekst editor, poput vi ili joe, ili komanda cat. Komanda cat se može iskoristiti za pisanje jednostavnih programa - najpre se komanda zada sa imenom datoteke kao argumentom za redirekciju izlaza, zatim se otkuca sadržaj programa i na kraju se pritisne kombinacija tastera <Ctrl-D>.

```
$ cat >ss1
#
# ss1: jednostavan shell program
#
clear
echo "Goodbye, Cruel World !"
<CTRL-D>
```

Program je na ovaj način upisan u datoteku first. Dalje je potrebno dati korisnicima x (execute) dozvolu nad datotekom i pokrenuti program:

```
$ chmod +x ss1
$ ./ss1
```

Efekat našeg prvog shell programa je sledeći. Najpre će se obrisati ekran (komanda clear), a zatim će se na ekranu prikazati poruka Goodbye, Cruel World !

Komentari

U shell programu, sve linije koje počinju karakterom # smatraju se komentarima i kao takve komandni interpreter ih ignoriše. Komentari služe da objasne korišćenje programa, da prikažu autora, ili da pruže neka objašnjenja specifičnih delova programa.

Pokretanje shell programa

Pokretanje shell programa uključuje dva koraka: dodelu dozvola za izvršavanje i samo pokretanje programa iz komandne linije. Shell programi su tekstualne datoteke čija pristupna prava nakon kreiranja ne uključuju dozvolu za izvršavanje, tako da se ona mora eksplicitno dodeliti određenoj grupi korisnika.

```
$ chmod +x ss1      # dodela dozvole za izvršavanje
$ ./ss1             # pokretanje programa
```

Tačka (.) ukazuje na tekući direktorijum i mora da se navede ukoliko se tekući direktorijum ne nalazi u sistemskoj putanji. Ukoliko se program ne nalazi u tekućem direktorijumu potrebno je navesti i putanju do programa (apsolutnu ili relativnu).

```
$ /home/jsmith/development/x12/scripts/beta/ss1
```

Da bi se izbeglo pisanje relativno dugih imena putanja korisnicima treba dati dozvolu za izvršavanje shell programa, a zatim ga postaviti u neki direktorijum koji je uključen u sistemsku putanju PATH, kao što je /usr/bin. Na ovaj način se program može pokrenuti iz bilo kog direktorijuma bez navođenja putanje.

```
$ ss1
```

Kada korisnik zada komandu, komandni interpreter najpre proverava da li je to interna komanda, i ukoliko jeste, odmah je izvršava. U protivnom, komandi interpreter traži komandu u direktorijumima koji su specificirani sistemskom putanjom, odnosno promenljivom PATH i ukoliko je nađe izvršava je. U protivnom se na ekranu prikazuje karakteristična poruka "bash:xxxx:command not found".

Prilikom pokretanja programa može se specificirati komandni interpreter u kome će se program izvršavati. Potrebno je u prvu liniju programa upisati sledeće:

```
#!/bin/bash
```

Ukoliko se komandni interpreter ne specificira na ovaj način, program se izvršava u tekućem interpreteru.

Shell program se može pokrenuti i na drugi način, bez eksplicitne dodele x dozvole - dovoljno je pozvati komandni interpreter da izvrši shell program:

```
$ bash ss1
```

ili

```
$ /bin/sh ss1
```

Ukoliko se shell program ne nalazi u tekućem direktorijumu, potrebno je specificirati putanju do programa.

```
$ bash /home/jsmith/ss1
```

Za razvoj i korišćenje shell programa preporučuje se sledeća procedura: shell program treba razviti na svom direktorijumu, zatim ga istestirati pomoću komande bash shell-program, i na kraju iskopirati u neki direktorijum koji je podrazumevano uključen u sistemsku putanju. Program se može kopirati u bin poddirektorijum home direktorijuma autora. Ukoliko veći broj korisnika želi da koristi program datoteku treba kopirati u direktorijume /bin ili /usr/bin kojima mogu pristupati svi korisnici. Dodatno, korisnicima treba dati dozvolu execute da bi mogli da pokreću program pomoću imena datoteke.

Pronalaženje komandnog interpretera

Gotovo po pravilu, shell script programi počinju linijom #!/bin/bash, što ukazuje da će program izvršavati komandni interpreter bash iz direktorijuma /bin. Ukoliko se bash ne nalazi na tom nestu potrebno je locirati njegovu poziciju, što se može učiniti komandama find, which i whereis.

```
$ whereis bash
bash: /bin/bash /etc/bash.bashrc /usr/share/man/man1/bash.1.gz
```

Bourne Again Shell se najčešće nalazi u jednom od sledećih direktorijuma: /bin, /sbin, /usr/local/bin, /usr/bin, /usr/sbin i /usr/local/sbin/bash.

Promenljive u Linux operativnom sistemu

Jedan od bitnih elemenata shell programiranja je upotreba promenljivih. Promenljive omogućavaju čuvanje podataka u operativnoj memoriji računara. Sledi ukratko objašnjenje pojma promenljiva: RAM memorija se deli na manje fragmente - memorijske lokacije koje imaju memorijsku adresu, odnosno jedinstveni broj koji se koristi za obraćanje toj memorijskoj lokaciji. Programer dodeljuje simboličko ime memorijskoj lokaciji i na taj način kreira memorijsku promenljivu, ili kraće, promenljivu. Promenljiva je imenovana memorijska lokacija koja može sadržati različite vrednosti, ali samo jednu vrednost u jednom trenutku. Na Linux sistemima postoje dva tipa promenljivih:

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

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 Shellu, promenljive mogu sadržavati brojeve, karaktere ili nizove karaktera.

Važnije sistemske promenljive

Sistemske promenljive mogu se videti pomoću komande:

```
$ 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
...
```

U nastavku teksta dat je opis značajnijih sistemskih promenljivih.

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 pomoću komande echo:

```
$ 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. Na taj način se vrednost value dodeljuje promenljivoj sa imenom variable. Vrednost se mora nalaziti sa desne strane znaka jednakosti. Na primer:

```
$ br=10             # ispravno
$ 10=br             # neispravno - vrednost na levoj strani
```

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 sa jednim ili više alfanumeričkih karaktera. Na primer, korektne promenljive su: HOME, SYSTEM_VERSION, br, _ime;
- Prazne karaktere ne treba stavljati ni sa jedne strane znaka jednakosti prilikom dodele vrednosti promenljivim. Na primer, sledeće deklaracije neće biti korektne, odnosno napraviće grešku zbog postojanja praznih karaktera:

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

- Promenljive su osetljive na velika i mala slova (case sensitive), kao i imena datoteka u Linux sistemu. Na primer, br, bR, Br i BR su četiri različite promenljive;

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

- Može se definisati promenljiva nulte dužine (NULL variable), odnosno promenljiva koja nema vrednost u trenutku definisanja. Vrednosti ovih promenljivih se logično ne mogu prikazati pomoću komande echo, sve dok se promenljivoj ne dodeli neka vrednost;

```
$ br=
```

```
$ ime=""
```

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

Prikazivanje i korišćenje vrednosti UDV promenljivih

Da bi prikazali ili pristupili vrednosti promenljive potrebno je ukazati na njenu vrednost, što se postiže uporebom znaka $ sa imenom promenljive.

Na primer, da bi prikazali vrednost promenljive ime upotrebićete komandu:

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

U nastavku teksta dati su primeri koji služe za bolje razumevanje promenljivih.

Primer 1. Definišite promenljive x i xn koje će imati vrednosti 10 i abc respektivno i prikažite njihove vrednosti na ekranu u istom redu.

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

Primer 2. Koristeći naredbu expr, prikazati zbir dva broja na ekranu

```
$ echo 6+3          # echo posmatra 6+3 kao niz karaktera
6+3
$ 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 +, -, *, /, &. Relzultat operacije je ceo broj (20/3=6, 20%3=2). Argumenti op1, op2 i operator se moraju razdvojiti praznim karakterom.

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

Primer 3. Definisati dve promenljive, x i y, i promenljivu z kao njihov količnik. Vrednosti promenljivih x i y su 20 i 5 respektivno. Rezultat prikazati na ekranu, u jednom redu.

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

Komande specifične za shell programiranje

Sastavni deo shell programa su komande. To mogu biti standardne Linux komande (poput cp ili mv) ili komande specifične za shell programiranje. Neke od komandi specifičnih za shell programiranje su gotovo kompletni programski jezici (na primer awk). U nastavku teksta opisani su samo njihovi osnovni elementi, dok je za više informacija potrebno konsultovati on-line uputstva (man pages).

echo

Komanda echo prikazuje tekst ili vrednost promenljive na ekranu. Sintaksa komande echo je:

```
$ echo [opcije] [string, promenljive...]
```

Opcije:

-n	ova opcija ne prebacuje kursor u novi red,nakon izvršenja echo komande	
-e	omogućava interpretaciju sledećih karaktera u kombinaciji sa obrnutom kosom crtom:	
\a	upozorenje	(alert bell)
\b	povratak unazad	(backspace)
\c	ne prelaziti u novi red	(suppress trailing new line)
\n	novi red	(new line)
\r	povratak na početak reda	(carriage return)
\t	horizontalni tabulator	(horizontal tab)
\\	obrnuta kosa crta	(backslash)

Na primer:

```
$ echo -e "Linux\n\t\tRulez !\n"
Linux
          Rulez !
```

Navodnici

Linux prepoznaje tri tipa navodnika.

- Dvostruki navodnici - "Double Quotes". Sve što se nalazi u ovim navodnicima gubi originalno značenje (osim \ i $).
- Jednostruki navodnici - 'Single quotes'. Sve što je zatvoreno jednostrukim navodnicima ostaje nepromenjeno.
- Obrnuti navodnici `Back quote`. Izraz zatvoren obrnutim navodnicima tretira se kao komanda koju treba izvršavati.

Na primer, ukoliko korisnik želi da na ekranu prikaže tekući datum navešće izraz date sa obrnutim navodnicima:

```
$ echo "Današnji datum : date"        # tretira date kao string
Today is : date
$ echo "Današnji datum : `date`"      # tretira date kao komandu
Today is : Fri Apr  2 16:30:35 CEST 2004
```

Komanda expr i shell aritmetika

Naredba expr koristi se da obavi jednostavne aritmetičke operacije. Sintaksa komande je:

```
$ expr op1 operator op2
```

gde su op1 i op2 celi brojevi, a operator:

+	Sabiranje
-	Oduzimanje
*	Množenje (za množenje se koristi *, a ne *, pošto je * džoker)
/	Deljenje
%	Ostatak po modulu

Primer komande expr su:

```
$ expr 1 + 3
4
$ expr 20 % 3
2
$ echo 6+3=`expr 6 + 3`      # ispravno (obrnuti navodnici)
6+3=9
$ echo 6+3="expr 6 + 3"      # neispravno (obični navodnici)
expr 6 + 3=9
```

Na osnovu poslednjeg primera ukazaćemo na sledeća sintaksna pravila:

Naredba expr se kao parametar komande echo zadaje u obrnutim navodnicima, a ne u dvostrukim ili jednostrukim. Samo u tom slučaju echo posmatra expr kao komandu, a ne kao običan string.

read

Komanda read se koristi za čitanje ulaznih podataka sa tastature i memorisanje unete vrednosti u promenljivu. Dodatno, to je jednostavan i dobar način da se u razvojnoj fazi ubace prekidne tačke u program i na taj način kontroliše njegovo izvršavanje.

Sintaksa komande read je:

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

Upotreba ove komande ilustrovana je programom ss2 koji čita ulazni podatak sa tastature u promenljivu var1, a zatim ga ispisuje na ekran.

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

Program se pokreće na standardan način, komandom bash ss2.

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

sed (stream editor)

Editor sed je neinteraktivni editor - ime izvorišne datoteke i instrukcije za rad sa tekstom se navode kao parametri u komandnoj liniji. Funkcionisanje editora opisano je pomoću sledeća dva primera:

```
$ cat data
123456789
Roadrunner
9876-12345
Wile E. Coyote, genius
1234
$ sed 's/12345/abc/g' /tmp/data
abc6789
Roadrunner
9876-abc
Wile E. Coyote, genius
1234
$ sed 2, 4d /tmp/data
123456789
1234
```

U prvom slučaju sed editor zamenjuje niz karaktera '12345' datoteke /tmp/data nizom karaktera 'abc'. U drugom slučaju sed editor prikazuje sve linije datoteke /tmp/data osim linija 2 do 4. Rezultat se prikazuje na standardnom izlazu (stdout, odnosno ekran), a ulazna datoteka se ne modifikuje. Ukoliko je potrebno promene snimiti u datoteku, koristi se redirekcija izlaza:

```
$ sed 's/12345/abc/g' /tmp/data > /tmp/data1n
$ sed 5, 10d /tmp/data  > /tmp/data2n
```

awk

Osnovna funkcija komande awk je pronalaženje uzorka teksta u datoteci i izvršavanje neke akcije, najčešće obrade pronađenog teksta. Generalno, awk je tekst procesor realizovan putem jednostavnog programskog jezika, a najpoznatiji interpreteri su GNU awk (gawk) i mawk. Funkcionisanje awk prikazano je sledećim primerima:

```
$ cat /tmp/data
```

```
123abc
Wile E.
aabcc
Coyote
$ awk '/abc/ {print}' /tmp/data
123abc
aabcc
```

U ovom slučaj awk traži uzorak ‘abc’ u datoteci /tmp/data, a akcija koja se pri tom obavlja nad nađenim uzorcima je prikazivanje teksta na ekranu (print). Drugi primer je štampanje rednog broja prve linije teksta iza koje se traženi uzorak ne pojavljuje:

```
$ awk '/abc/ {i=i+1} END {print i}' /tmp/data
3
```

Ukoliko se u datoteci traži više uzoraka i ukoliko se vrši više obrada, potrebno je prvo napraviti datoteku u kojoj su opisane akcije (na primer actionfile.awk). Prilikom zadavanja komande awk potrebno je zameniti tekst između navodnika, kojim su opisani uzorak i akcija, imenom datoteke: '-f actionfile.awk'.

grep

Osnovna funkcija grep komande je da pronađe sve linije sa traženim uzorkom. Dodatno, grep može umesto prikazivanja da obavi prebrojavanje linija u kojima je uzorak pronađen.

```
# grep bora /etc/group
bora-admin:x:1003:
bora:x:1047:
# grep bora /etc/group -c
2
```

U ovom slučaju, niz "bora" je pronađen 2 puta u datoteci /etc/group.

wc (word count)

Komanda wc prebrojava broj linija, reči i bajtova u datoteci. Pretpostavimo da datoteka /tmp/data sadrži sledeći tekst:

```
$ less /tmp/text_file
First line of the text file
Second line of the text file
$ wc /tmp/text_file
2      12      58 text_file
```

Kao što se vidi iz primera komanda wc prikazuje rezultat u standardnom poretku: broj linija, broj reči, broj bajtova.

sort

Komanda sort uređuje linije tekstualne datoteke po abecedi ili nekom drugom kriterijumu (kao što su meseci u godini). U ovom slučaju datoteka /tmp/data sadrži sledeći tekst:

```
$ cat /tmp/data
bash
crypt
awk
$ sort /tmp/data
awk
bash
crypt
```

bc (basic calculator)

Program bc služi za izračunavanje vrednosti izraza - interaktivno ili sa komandne linije. Dodatno, bc ima ugrađene neke elemente programskog jezika (petlje i uslovi). Sledeći primer demonstrira upotrebu programa (parametar -q se navodi ukoliko je potrebno izbeći pozdravnu poruku):

```
$ bc -q
2 ^ 8
256
sqrt(9)
3
quit
$ cat bctest
print "Unesite pocetnu vrednost: "; i=read ();
print "Unesite krajnju vrednost: "; j=read ();
while (i<=j) {print i^2; print "\n"; i=i+1}
quit
$ bc -q bctest
Unesi pocetnu vrednost: 5
Unesi krajnju vrednost: 8
25
36
49
64
```

tput

tput inicijalizuje terminal ili postavlja upit u bazu podataka u kojoj su opisani terminali.

```
$ tput cup 10 4     # postavlja prompt u poziciju (y=10,x=4)
$ tput reset        # briše ekran i postavlja prompt u (y=1,x=1)
$ tput cols         # ova komanda prikazuje broj kolona
80
```

Komande, argumenti i izlazni status

Pretpostavimo da korisnik zadaje sledeću komandu:

```
$ tail -1 /etc/passwd
jsmith:x:1051:1051:John Smith,,,:/home/jsmith:/bin/bash
```

Postavlja se sledeće pitanje: šta se dešava kada korisnik zada prethodnu komandu? Prva reč (tail) je ime interne komande koju treba izvršiti ili programa koji treba pokrenuti. Sve ostalo iza imena predstavlja ulazne parametre komande, odnosno programa. To znači da se pokreće program tail iz direktorijuma koji je specificiran sistemskom putanjom, a da se parametri -1 i /etc/password prosleđuju tom programu.

Komanda može biti zadana 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.

Zašto se zahtevaju komandni argumenti ?

Na primeru komande rm može se jednostavno objasniti zašto je ponekad neophodno navesti komandne argumente.Naime, komanda rm se koristi za brisanje datoteka, tako da je neophodno da korisnik navede šta želi da obriše.

```
$ rm filename1 filename2 dir1
```

Ovde je rm ime komande, a argumenti su imena datoteka i direktorijuma koje korisnik želi da obriše. Takođe, navođenjem dodatnih parametara može se osim imena datoteka specificirati i način izvršenja komande (interaktivno ili forsirano, rekurzivno).

Shell program i komandni argumenti

Komandni argumenti se na isti način mogu zadati i shell programu. Na primer, za shell program ss3, koji je pozvan pomoću sledeće linije, prvi komandni argument je arg1, drugi arg2, a treći arg3:

```
$ ss3 arg1 arg2 arg3
```

U shell programu se argumentima komandne linije pristupa pomoću sledećih promenljivih:

- vrednost promenljive $0 je ime programa (u ovom slučaju ss3)
- vrednost promenljive $1 je prvi komandni argument (u ovom slučaju arg1)
- vrednost promenljive $2 je drugi komandni argument (u ovom slučaju arg2)
- vrednost promenljive $3 je treći komandni argument (u ovom slučaju arg3)
- vrednost promenljive $# je broj komandnih argumenta (u ovom slučaju dva)
- vrednost promenljive $* su svi komandni argumenti. $* se proširuje u `$0,$1,$2...$9` (što je u ovom slučaju arg1, arg2, arg3).

U nastavku teksta su na osnovu nekoliko komandi analizirane vrednosti specijalnih promenljivih: ime programa ($0), broj argumenata ($#), i aktuelne vrednosti argumenata ($1,$2 i tako dalje).

```
$ df
$ less /etc/passwd
$ ls -l /etc
```

```
$ mount -r /dev/hda2 /mnt/winc
```

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

Upotreba argumenata u samom programu objašnjena je sledećim primerom koji na ekranu ispisuje ime programa, broj argumenata i sve argumente redom. Program se pokreće na standardan način (na primer: bash ss3 arg1 arg2 arg3).

```
#
# ss3: korišćenje argumenata komandne linije
#
echo "Ukupan broj argumenata komandne linije: $#"
echo "$0 je ime programa, a $1 je prvi argument."
echo "Svi argumenti su redom: $*"
```

Izlazni status komande

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. Da bi se odredila ova vrednost koristi se sistemska promenljiva **$?**, čija je upotreba demonstrirana sledećim primerima:

```
$ 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
```

Grupisanje komandi

Bash obezbeđuje dva načina grupisanja komandi, tj. izvršavanja više komandi u okviru jedne celine. Kada se komande grupišu redirekcije mogu da se primene na celu grupu komandi. Na primer, izlaz svih komandi u listu može da se redirektuje u jedan niz.

(list)

Postavljanje liste komandi između malih zagrada ima za posledicu kreiranje posebnog shell potprograma u kome se izvršavaju sve komande iz liste. Pošto se lista komandi izvršava u shell podprogramu dodeljene promenljive prestaju sa važenjem nakon završetka potprograma. Male zagrade su operatori, njih komandni interpreter prepoznaje kao specijalne karaktere, čak i ukoliko nisu razdvojeni od list komandi praznim karakterima.

{ list; }

Postavljanje liste komandi između vitičastih zagrada ima za posledicu izvršavanje komandi iz liste u tekućem shell kontekstu. Za razliku od prethodnog slučaja poseban shell podprogram (proces) se ne kreira. Na kraju liste zahteva se znak ; ili prazan red. Velike zagrade su rezervisane reči, tako da one praznim karakterima moraju biti odvojene od liste komande.

Izlazni status za obe ove konstrukcije je izlazni status liste.

Parametri komandnog interpretera (Shell Parameters)

Postoje dve klase parametara komandnog interpretera: pozicioni parametri koji predstavljaju argumente komandne linije i specijalni parametri, koji imaju specijalno značenje za shell.

Parametar je celina koja memoriše vrednosti. To može biti ime, broj, ili jedan od specijalnih karaktera koji su opisani u nastavku teksta. Za svrhu komandnog interpretera, promenljiva je parametar koji je označen imenom. Promenljiva ima vrednost i atribute (0 ili više atributa). Atributi se dodeljuju korišćenjem deklarativnih ugrađenih komandi. Parametar je postavljen ako mu je dodeljena vrednost. Niz veličine 0 je validna vrednost. Kad se promenljiva postavi, to se može poništiti korišćenjem ugrađenih komandi za poništenje (unset builtin command).

Promenljiva se postavlja naredbom u sledećoj formi: name=[value]. Ako parametar value nije dat promenljivoj se dodeljuje niz veličine 0 (null string). U vrednosti promenljive ubrajaju se tilda proširenje, parametarsko proširenje i proširenje varijabli, komandna zamena, aritmetičko proširenje, i upotreba navodnika. Ako promenljiva ima svoj celobrojni atribut postavljen, tada se vrednost value pokorava aritemtičkom proširenju, čak i ako se $((...)) proširenje ne koristi. Razdvajanje reči se ne izvršava, sa izuzetkom "$@" kao što će biti objašnjeno kasnije. Proširenje imena datoteke se ne izvršava. Naredbe za dodeljivanje mogu takođe da se pojave kao argument za deklarisanje, postavljanje tipa, eksportovanje, nepromenljivost (readonly) i lokalne ugrađene komande.

Pozicioni parametri (positional parameters)

Pozicioni parametar se označava pomoću jedne ili više cifara, različitih od jednocifrenog broja 0. Pozicioni parametri se dodeljuju iz argumenata komandnog interpretera kada se pozove, a mogu se ponovo dodeliti koristeći unutrašnje komande. Pozicioni parametar N referencira se kao ${N}, ili kao $N, u slučaju kada se N sastoji od jedne cifre. Pozicioni parametri ne mogu se postavljati naredbama za postavljanje name=[value]. U tu svrhu

koriste se interne komande set i shift. Pozicioni parametri se privremeno zamenjuju kada se shell funkcija izvršava.

Specijalni parametri (special parameters)

Komandni interpreter tretira neke parametre specijalno. Ovi se parametri mogu samo referencirati, njihovo dodeljivanje nije dozvoljeno.

*	Proširuje se u pozicione parametre, počev od prvog ($1). Kada se proširivanje dogodi unutar duplih navodnika proširuje se jedna reč sa vrednošću svakog parametra, razdvojenog prvim karakterom specijalne promenljive koja se zove IFS. To znači, "$*" je ekvivalentno sa "$1c$2c...", pri čemi je c prvi karakter vrednosti IFS promenljive. Ako IFS nije postavljena parametri se razdvajaju praznim karakterom (space). Ako je IFS niz veličine 0 (null), parametri se udružuju bez međusobnog separatora.
@	Proširuje se u pozicione parametre, počevši od jedan. Kada se proširivanje dogodi unutar duplih navodnika, svaki parametar se proširuje u posebnu reč. To znači, "$@" je ekvivalentno sa "$1" “$2” ...".Kada nema nijednog pozicionog parametra, "$@" i $@ se ne mogu proširiti, te se uklanjaju.
#	Proširuje broj pozicionih parametara u decimalni.
?	Proširuje se u izlazni status poslednje izvršene foreground komande.
-	Proširuje se u tekuće opcione zastavice (flegove) koje su specificirane nakon pozivanja, a koje su postavile unutašnje komande ili komandni interpreter (kao na primer –i opcija).
$	Proširuje se u broj procesa (process ID) komandnog interpretera. U slučaju podprograma, parametar se proširuje u ID komandnog interpretera koji je pozvao potprogram, a ne u ID podprograma.
!	Proširuje se u broj procesa poslednje izvršene background komande.
0	Proširuje se u ime komandnog interpretera ili programa za taj komandni interpreter. To se postavlja prilikom inicijalizacije komandnog interpretera. Ako se Bash pozove sa script datotekom, $0 se postavlja na ime te script datoteke. Ako se Bash startuje sa –c opcijom, $0 se postavlja na prvi argument nakon što niz počne da se izvršava, ako je taj argument prisutan. U suprotnom, postavlja se na ime datoteke koja je pozvala Bash, kao sa argumentom 0.

Redirekcija i pipe mehanizam

Redirekcija ulaza i izlaza

Standardni ulaz (stdin), standardni izlaz (stdout) i standardni izlaz za greške (stderr) su deskriptori datoteke (file descriptor) kojima su dodeljeni brojevi po sledećim pravilima: 0 predstavlja stdin, 1 predstavlja stdout i 2 predstavlja stderr. U osnovi, u okviru shell programa mogu da se obave sledeće redirekcije, odnosno preusmerenja:

- redirekcija stdout u datoteku. Sledeći primer demonstrira kreiranje datoteke myfiles.txt i upis rezultata izvršenja komande ls-l u datoteku;

```
$ ls -l > myfiles.txt
```

- redirekcija stderr u datoteku. Sledeći primer demonstrira kreiranje datoteke greperr.txt i upis poruka o greškama koje proizvodi komanda grep u datoteku;

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

- redirekcija stdout u stderr;
- redirekcija stderr u stdout, koja je demonstrirana sledećim primerom. Rezultat izvršenja komande grep smešta se u bafer 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 * 2>&1 greperr.txt
```

- redirekcija stderr i stdout u stdout. Rezultat izvršenja komande i poruke o greškama koje komanda proizvodi prikazuju se na standardnom izlazu, a to je u podrazumevanom stanju ekran;
- redirekcija stderr i stdout u stderr;
- redirekcija stderr i stdout u datoteku. Ova vrsta redirekcije je korisna za programe koji rade u pozadini (background), 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 što je prikazano sledećim primerom:

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

- redirekcija datoteke u stdin.

Pomoću komande less mogu se videti i stdout i stderr - na primer, stdout će ostati u baferu dok se stderr prikazuje na ekranu. Bafer se može naknadno pregledati, nakon čega se briše.

Pipe mehanizam

Pipe mehanizam omogućava korišćenje standardnog izlaza jedne komande kao standardnog ulaza druge komande. Primer pipe mehanizma je komanda:

```
$ ls -l | wc -l
36
```

Komanda ls -l će izlistati tekući direktorijum (jedna datoteka u jednom redu), a pipe će standardni izlaz ove komande preusmeriti na standardni ulaz komande wc -l, koja broji redove teksta. Na ovaj način korisnik dobija informaciju o broju datoteka u tekućem direktorijumu.

Shell proširenja (Shell Expansions)

Proširenje preko zagrada, tilda proširenje, proširenje parametara i promenljivih, zamena komandi, aritmetičko proširenje, razdvajanje reči, proširenje imena datoteke.

Proširenje se izvršava na komandnoj liniji. Bash prepoznaje sedam proširenja i izvršava ih sledećim redom:

- proširenje preko zagrada (brace expansion)
- tilda proširenje (tilde expansion)
- proširenje parametara i promenljivih (parameter and variable expansion)
- aritmetičko proširenje (arithmetic expansion)
- zamena komandi (command substitution), koja se obavlja sleva nadesno
- razdvajanje reči (word splitting)
- proširenje imena datoteke (filename expansion).

Samo proširenje u zagradama, razdvajanje reči i proširenje imena datoteka mogu promeniti broj reči u proširenju, ostala proširenja proširuju jednu reč u jednu reč.

Proširenje preko zagrada (Brace Expansion)

Proširenje preko zagrada je mehanizam kojim se mogu proširiti proizvoljni nizovi. Ovaj mehanizam je sličan proširenju imena datoteke, ali generisana imena datoteka ne moraju da postoje. Uzorci koji se preko zagrada proširuju uzimaju formu opcionog uvodnog dela, koju prati serija zapetom razdvojenih nizova između para

zagrada, iza kojih ide opcioni dotatak. Uvodni deo je prefiks svakog niza koji se nalazi unutar zagrada, a dodatak se dodaje s desne strane na svaki rezultujući niz.

Proširenja preko zagrada mogu da se gnezde, odnosno da se umeću jedno u drugo. Rezultati svakog proširenog niza nisu sortirani, samo se poštuje poredak sleva nadesno, odnosno prefiks, zatim niz iz zagrade, i na kraju dodatak-sufiks. Jednostavan primer je proširenje komande echo:

```
$ echo a{d,c,b}e
ade ace abe
```

Proširenje preko zagrade se izvršava pre bilo kog drugog proširenja. Bilo koji karakter koji ima specijalno značenje za ostala proširenja čuva se u rezultatu, odnosno ne dira se. To je strogo tekstualno proširenje. Bash ne primenjuje interpretaciju u kontekstu proširenja ili teksta između zagrada. Da bi izbegavao konflikte sa parametarskim proširenjima niz "${" se ne smatra pogodnim za proširenje preko zagrada. Korektno formirano proširenje preko zagrada mora sadržavati otvorenu i zatvorenu zagradu koje su van navodnika, i barem jednu zapetu. Svako nekorektno proširenje se ne izvršava.

Ova konstrukcija se tipično koristi kao skraćenica kada se isti zajednički prefiks generiše više puta, kao što je prikazano u sledećim primerima. Tako se:

```
$ mkdir /home/jsmith/{data,video,mp3}
```

proširuje u:

```
$ mkdir /home/jsmith/data
$ mkdir /home/jsmith/video
$ mkdir /home/jsmith/mp3
```

Komplikovaniji slučaj je korišćenje ugnježđenih proširenja.

```
$ chown root /home/{jsmith/{ss1,ss2},nmacek/{data,ss3}}
```

proširuje se u:

```
$ chown root /home/jsmith/ss1
$ chown root /home/jsmith/ss2
$ chown root /home/nmacek/data
$ chown root /home/nmacek/ss3
```

Tilda proširenje (Tilde Expansion)

Ako reč počinje tilda karakterom koji nije pod navodnicima (~), svi karakeri do prve kose crte koja je takođe van navodnika (/ slash) tretiraju se kao tilda prefiks. Ukoliko nema kose crte onda su svi karakteri tilda prefiks.

Ukoliko nema karaktera pod navodnicima unutar tilda prefiksa, tilda prefiks se tretira kao potencijalno ime korisnika za login proceduru (login-name). Tilda prefiks se zamenjuje po sledećim pravilima: ako je login-name nulte dužine, tilda se zamenjuje vrednošću HOME shell promenljive, a ako je HOME promenljiva nepostavljena, tilda se zamenjuje home direktorijumom korisnika koji izvršava taj komandi interpreter.

U drugom slučaju tilda prefiks se zamenjuje home direktorijumom specificiranog korisnika (login-name). Ako je vrednost tilda prefiksa "~+", tada tilda prefiks uzima vrednost shell promenljive PWD koja predstavlja tekući radni direktorijum. Ako je vrednost tilda prefiksa "~-", tada tilda prefiks uzima vrednost shell promenljive OLDPWD koja predstavlja prethodni tekući radni direktorijum (pod uslovom da je OLDPWD setovana).

Ako je login-name pogrešan, tilda proširenje se ne izvršava, reč s leve stane ostaje nepromenjena. Svaka dodela promenljivoj se proverava za tilda prefikse van navodnika iza kojih neposredno ide : ili =. U ovim slučajevima tilda proširenje se takođe izvršava. Prema tome, nekom mogu koristiti imena datoteka sa tildom u dodeljivanju sistemskih promenljivih kao što je PATH, MAILPATH i CDPATH, a komandni interpreter će im dodeliti proširene vrednosti.

Sledeći primeri pokazuju kako Bash tretira tilda prefikse bez navodnika. Prvi primer demonstrira upotrebu tilda proširenja za pozicioniranje na home direktorijum:

~ vrednost promenljive $HOME (/home/jsmith)

~/data $HOME/data (/home/jsmith/data)

~jim home directorijum korisnika jim (/home/jim).

Primeri se oslanjaju na pretpostavku da je na sistem prijavljen korisnik jsmith, čiji je home direktorijum /home/jsmith i da on pokreće komande koje sadrže ova proširenja.

```
$ whoami
jsmith
$ pwd
/etc
$ cd ~/data              # poddirektorijum data home direktorijuma
$ pwd
/home/jsmith/data
$ cd ~jim                # home direktorijum korisnika jim
$ pwd
/home/jim
```

Sledeći primer demonstrira upotrebu tilda proširenja za promenljivu $OLDPWD:

~+/misc $PWD/misc

~-/temp $OLDPWD/temp

```
$ pwd
/etc
$ cd /bin
$ cd ~-/network
$ pwd
/etc/network
```

Parametarsko proširenje (Shell Parameter Expansion)

Znak $ uvodi parametarsko proširenje, komandnu zamenu ili aritmetičko proširenje. Ime parametra ili simbola koji se proširuje može biti zatvoreno u zagradama koje su opcione ali štite promenljivu koja se proširuje od karaktera koji je neposredno slede i koji bi se mogli pogrešno interpretirati kao deo imena. Kada se koriste zagrade, zadnja zagrada je prvi znak koji nije u sastavu obrnute kose crte ili pod navodnicima i nije u okviru aritmetičkih proširenja, komandnih zamena ili parametarskih proširenja.

Osnovna forma parametarskog proširenja je ${par}. Vrednost parametra (par) se zamenjuje. Zagrade se zahtevaju kada je reč o pozicionom parametru, sa više od jedne cifre, ili kada je parametar praćen karakterom koji se ne interpretira kao deo njegovog imena.

Ako je prvi karakter parametra znak uzvika "!", uvodi se nivo promenljive indirekcije. Bash koristi vrednost promenljive formirane od ostatka parametra kao ime promenljive. Ova promenljiva se zatim proširuje i ta vrednost se koristi u ostatku zamene, umesto vrednosti samog parametra. Ovo je poznato kao indirektno proširenje. Izuzetak ovog proširenja je slučaj ${!prefix*}.

${par:-word} Ako parametar ne postoji ili je null, zamenjuje se proširenjem word. U drugom slučaju zamenuje se vrednošću parametra.

${par:=word}	Ako parametar ne postoji ili je null, proširenje word se dodeljuje parametru. Vrednost parametra se tada zamenjuje. Pozicioni i specijalni parametri ne moraju se postavljati na ovaj način.
${par:?word}	Ako parametar ne postoji ili je null, proširenje word se upusuje na standardni izlaz za greške (standard error) i komandi interpreter prekida rad (exit). Vrednost parametra se tada zamenjuje.
${par:+word}	Ako parametar ne postoji ili je null, ništa se ne zamenjuje, a u drugom slučaju proširenje word se zamenjuje.
${par:offset:lenght}	Proširuje do dužine length karaktera parametra, počevši od karaktera specificiranog pomoću polja offset. Ako se polje length izostavi, proširuje se podniz počevši od karaktera specificiranog pomoću polja offset zaključno sa zadnjim karakterom. Polja length i offset su aritmetički izrazi. Ovo se još naziva i podnizno proširenje (Substring Expansion). Polje length mora biti broj veći ili jednak 0. Ako je polje offset broj manji od 0, vrednost se koristi kao pomeraj u odnosu na kraj vrednost i parametra. Ako je parametar "@", rezultat je polje length pozicionih parametara koji počinju u polju offset. Ako je parametar polje imena koje se indeksira pomoću "@" ili "*", rezultat je length polje članova polja koji počinju sa ${par[offset]}. Podnizno indeksiranje je bazirano na nuli, osim u slučaju kada se koriste pozicioni parametri, kada indeksiranje startuje u 1.
${!prefix*}	Proširuje imena promenljivih čija imena počinju prefiksom prefix, razdvojenim prvim karakterom IFS specijalne promenljive.
${#par}	Dužina proširene vrednosti parametra se zamenjuje. Ako je parametar "*" ili "@", zamenjena vrednost je broj pozicionih parametara. Ako je parametar polje imena koja se indeksiraju pomoću "@" ili "*", zamenjena vrednost je broj elemenata u polju.

Komandna zamena (Command Substitution)

Komandna zamena dozvoljava da se izlaz komande zameni samom komandom, odnosno da izlaz jedne komande postaje argumenat druge. Komanda zamena se izvršava kada se komanda zatvori zagradama ili navodnicima, kao u sledećim primerima:

```
$(command)
```

ili

```
`command`
```

Bash izvršava proširenje izvršavanjem komande command i zamenjuje komandnu substituciju sa standardnim izlazom komande. Ugrađene nove linije se ne brišu, ali mogu da se uklone za vreme razdvajanja reči.

Kada se koristi zamena sa starim stilom forme obrnutog navodnika, karakter obrnuta kosa crta zadržava doslovno značenje osim kada je praćen sa "$", "`", ili "\". Prvi obrnuti navodnik, koji nije praćen obrnutom kosom crtom, prekida komandnu zamenu. Kada se koristi $(command) forma, svi karakteri između malih zagrada tretiraju se kao komande, ništa se ne tretira specijalno.

Ako se zamena pojavljuje sa duplim navodnicima, razdvajanje reči i proširenje imena datoteka datoteka se ne izvršava.

Komandu zamenu demonstriraćemo preko kompresije grupe *.bak datoteka. Komanda find pronaći će sve takve datoteke:

```
$ find / -name '*.bak' –print
```

Komprimovanje u jednoj komandi može se izvršiti na dva načina:

```
$ gzip ` find / -name '*.bak' –print `
```

ili

```
$ gzip $( find / -name '*.bak' –print )
```

Dodatno, pomoću komandne zamene mogu se dodeliti vrednosti promenljivim.

```
$ x = `date`
$ echo $x
Thu Apr 15 09:53:44 CEST 2004
$ y = `who am i;pwd`
$ echo $y
nmacek pts/0 Apr 15 09:40 (nicotine.internal.vets.edu.yu)
/home/nmacek
```

Aritmetičko proširenje (Arithmetic Expansion)

Aritmetičko proširenje omogućava izračunavanje aritmetičkog izraza i zamenu rezultata. Format aritmetičkog izraza je:

```
$(( expression ))
```

ili

```
$[ expression ]
```

Izraz se tretira kao da je bio u duplim navodnicima, ali dupli navodnici unutar zagrada se ne tretiraju specijalno. Svi simboli u izrazu podležu parametarskom proširenju, komandnoj zameni i navodničkom uklanjanju. Aritmetičke zamene mogu da se gnezde.

Izračunavanje se izvršava prema pravilima shell aritmetike. Ako je izraz pogrešan bash prikazuje poruku koja prijavljuje otkaz i zamena se ne izvršava.

Evo nekoliko primera:

```
$ echo 1 + 1        # shell interpretira 1 + 1 kao string
1 + 1
$ echo $((1+1))     # $((1+1)) je aritmetičko proširenje
2
$ echo $((7/2))     # bash koristi celobrojnu aritmetiku
3
$ echo 3/4|bc -l    # celobrojna aritmetika
0.75
$ a=1
$ b=2
$ echo $(($a+$b))   # promenljive i aritmetičko proširenje
3
```

Bash koristi celobrojnu aritmetiku - komanda echo $[3/4] na ekranu prikazuje 0. Ukoliko je potrebno izvršiti neku operaciju sa realnim rezultatom ili više matematičkih operacija, može se koristiti program bc - rezultat komande echo 3/4|bc -l je 0.75, što je korektno.

Aritmetičko proširenje se može iskoristiti za određivanje istinitosti izraza. U tom slučaju, proširenje vraća status 0 ili 1 zavisno od izračunavanja uslovnog izraza expression. Izraz se komponuje pomoću operatora <, <=, >, >=, == i ! =. Dodatno, izrazi mogu da se kombinuju koristeći sledeće operatore:

(expression)	vraća vrednost izraza expression. Ovo se može koristiti za nadjačavanje normalnog prioriteta operatora.
! expression	tačno ukoliko je expression netačan (negacija)
exp1 && exp2	tačno samo pod uslovom ako su oba izraza (exp1 i exp2) tačni
exp1 || exp2	tačno ako je bar jedan od izraza (exp1 ili exp2 tačan).

Operatori && i || ne izračunavaju vrednost exp2 ako je vrednost izraza exp1 dovoljna da odredi povratnu vrednost celog uslovnog izraza.

```
$ echo $((1>3||2<4))
1
$ echo $((1>3&&2==2))
0
```

Kada se koriste operatori "==" i "!=" niz desnog operatora smatra se uzorkom, a provera identičnosti odgovara pravilima za pronalaženje uzorka (Pattern Matching). Vrednost 0 se vraća ako niz odgovara uzorku, a vrednost 1 ako ne odgovara. Razdvajanje reči i proširenje imena datoteka se ne izvršavaju unutar ovog proširenja; tilda proširenje, parametarsko proširenje, komandna zamena, procesna zamena i upotreba navodnika se izvršavaju.

```
$ ime=jsmith
$ echo $(($ime==jsmith))
1
$ echo $(($ime!=jsmith))
0
```

Proširenje imena datoteka (Filename Expansion)

Nakon zadavanja komande, Bash razdvaja reči koje predstavljaju parametre i u parametrima koji predstavljaju datoteke traži karaktere "*", "?", i "[". Ako se jedan od tih karaktera pojavi tada se reč smatra uzorkom i zamenjuje se alfabetski sortiranom listom imena datoteka koja odgovara uzorku. Ukoliko je Bash pokrenut sa parametrom -f ova zamena se ne izvršava.

Pronalaženje uzorka (Pattern Matching)

Prilikom pronalaženja uzorka specijalni karakteri imaju sledeće značenje:

*	odgovara bilo kom nizu uključujući i niz nulte dužine. Tako će, na primer, komanda ls * prikazati sve datoteke, ls a* sve datoteke čije ime počinje sa a, a ls *.c sve datoteke koje imaju ekstenziju .c;
?	odgovara bilo kom karakteru. Tako će, na primer, ls ? prikazati sve datoteke čije ime ima tačno jedan karakter, a ls fo? sve datoteke čije ime ima tačno tri karaktera, od kojih su prva dva fo;
[...]	odgovara jednom od karaktera koji je naveden između zagrada. Ukoliko je prvi karakter iza otvorene zagrade "!" ili "^" tada odgovaraju svi karakteri koji nisu navedeni između zagrada. Na primer, ls [abc]* će prikazati sve datoteke čije ime počinje slovima a,b ili c, a ls [^abc]* sve datoteke čije ime ne počinje tim slovima;
[..-..]	par karaktera razdvojen znakom "-" označava zonu, odnosno opseg. Ukoliko je prvi karakter iza otvorene zagrade "!" ili "^" tada odgovaraju svi karakteri koji ne pripadaju opsegu. Na primer, ls /bin/[a-f]* će prikazati sve datoteke direktorijuma /bin čije ime počinje slovima a,b,..f, a ls /bin/[^a-e]* sve datoteke direktorijuma /bin čije ime ne počinje tim slovima;

Konstrukcije u shell programiranju

Uslovne konstrukcije (if, case), petlje (while, until, for, select). Funkcije.

Od konstrukcija koje su karakteristične za više programske jezike, u shell programima se mogu koristiti:

- uslovne konstrukcije (if, case),
- petlje (while, until, for, select),
- funkcije.

Uslovne konstrukcije

Uslovna konstrukcija if

if konstrukcija se može primeniti u tri osnovna oblika:

- if-then-fi
- if-then-else-fi
- if-then-elif-else-fi.

Glavni deo svake if naredbe je provera uslova koja se obavlja pomoću interne komande test i na osnovu čijih rezultata se odlučuje koje će se naredbe izvršavati.

U najkompleksnijem obliku sintaksa if komande je:

```
if test-commands;
then
    consequent-commands;
[elif more-test-commands;
    then
    more-consequents;]
[else alternate-consequents;]
fi
```

If petlja funkcioniše na sledeći način. Najpre se ispituju uslovi izvršavanjem liste komandi test-commands čiji povratni status određuje da li su uslovi ispunjeni. Uslovi su ispunjeni ako je povratni status 0, što znači da se tada izvršava lista komandi consequent-commands i to je kraj if petlje. Ukoliko uslovi nisu ispunjeni test-command će vratiti status različit od 0, a onda će se svako elif testiranje izvršavati redom do prvog ispunjenja uslova, kada će se izvršiti more-consequents naredbe iz odgovarajuće elif strukture. Ukoliko je else struktura prisutna, a svi prethodni testovi otkažu, počevši od glavne if strukture pa preko svih elif struktura, tada će se izvršiti lista naredbi alternate-consequents koju sadrži else struktura. Povratni status cele if strukture je izlazni status zadnje izvršene komande, ili 0 ako nijedan od postavljenih uslova u if i elif strukturama nije ispunjen, a else ne postoji.

if-then-fi

U ovom obliku if komande izvršiće se jedna ili grupa komandi ukoliko je uslov ispunjen, a u protivnom cela if struktura ne radi ništa.

Sintaksa najprostijeg oblika je:

```
if condition
then
    command 1
    ...
    command n
fi
```

U najprostijem svom obliku komanda if testira uslov condition i ako je on ispunjen, izvršava jednu ili grupu komandi (zaključno sa komandom pre komande fi, koja je kraj if strukture).

Provera uslova i test naredba

Provera uslova realizuje se preko izlazne vrednosti koja može biti 0, što znači da je uslov ispunjen, ili različita od 0, što znači da uslov nije ispunjen.

Uslov condition može biti, na primer, komparacija dve vrednosti koju obavlja komanda test ili njen skraćeni oblik pisanja [expression]. Izraz expression je kombinacija vrednosti, relacionih operatora kao što je >,<, == ili matematičkih operatora kao što su +, -, /.

Drugi oblik uslova condition je izlazni status komande. Svaka Linux komanda vraća status koji opisuje da li je uspešno izvršena ili ne. Status zadnje izvršene komande smešta se u promenljivu ?, što je demonstrirano sledećim shell programom:

```
#
# ss4: korišćenje izlaznog statusa komande
#
if rm $1
then
    echo "Datoteka $1 je uspešno obrisana"
fi
```

Program se pokreće pomoću komande bash ss4 filename. Argument filename je prvi argument ($1). U uslovnom delu mi kompariramo da li datoteka postoji: if rm $1 (odnosno if rm filename). Ako komanda rm pronađe datoteku i uspešno je obriše, njen izlazni status je 0, te će se izvršiti naredba ispod then (echo "Datoteka $1 je uspešno obrisana). U protivnom, izlazni status je različit od 0, i komanda se izvršava.

```
$ bash ss4 non_existing
rm: cannot remove `non_existing': No such file or directory
$ bash ss4 existing
Datoteka existing je uspešno obrisana
```

Provera tačnosti izraza komandom test

Komanda test se koristi za proveru tačnosti izraza u uslovnim konstrukcijama. Ukoliko je izraz tačan komanda vraća vrednost 0, odnosno vrednost veću od nule ako je izraz netačan.

Sintaksa komande test je:

```
test  expression
```

ili:

```
[ expression ]
```

U nastavku teksta dat je jednostavan shell program koji određuje da li je numerički argument pozitivan.

```
#
# ss5: Da li je argument pozitivan broj ?
#
if test $1 -gt 0        # alternativno: if [ $1 -gt 0 ]
then
    echo "$1 je pozitivan broj"
```

```
fi
```

Program se pokreće komandom: bash ss5 arg, gde je arg numerička vrednost.

```
$ bash ss5 5
5 je pozitivan broj
$ bash ss5 -4
```

Linija if test $1 -gt 0, utvrđuje tačnost izraza $1>0, odnosno proverava da li je prvi argument komande ($1) veći od 0. U prvom slučaju uslov je ispunjen, komanda vraća vrednost 0, a if konstrukcija pokreće komandu echo koja će prikazati poruku "5 je pozitivan broj". U drugom slučaju argument je -4, izraz 4>0 nije tačan, pa se komanda echo ne izvršava.

Pomoću komande test, odnosno [expr] mogu se upoređivati celi brojevi, upoređivati nizovi karaktera i može se odrediti da li datoteka postoji, da li je regularna, izvršna, itd.

Sledeći matematički operatori se koriste za upoređivanje celih brojeva:

Operator	Značenje	if test	if [ ]
-eq	jednakost (5=6)	if test 5 -eq 6	if [5 -eq 6]
-ne	nejednakost (5 != 6)	if test 5 -ne 6	if [5 -ne 6]
-lt	strogo manje od (5<6)	if test 5 -lt 6	if [5 -lt 6]
-le	manje od ili jednako (5<= 6)	if test 5 -le 6	if [5 -le 6]
-gt	strogo veće od (5>6)	if test 5 -gt 6	If [5 -gt 6]
-ge	veće od ili jednako (5>=6)	if test 5 -ge 6	If [5 -ge 6]

Za upoređivanje nizova koriste se sledeći operatori:

string1 = string2	da li je niz string1 jednak nizu string2
string1 != string2	da li je niz string1 različit od niza string2
string1	da li je string1 definisan i ako jeste da li nije NULL
-n string1	da li string1 nije NULL
-z string1	da li ¸je string1 NULL

Komandom test takođe se mogu izvršiti testovi nad datotekama i direktorijumima:

-s file	da li datoteka ima neki sadržaj
-f file	da li datoteka postoji, da li je obična a ne direktorijum
-d dir	da li direktorijum postoji, i da li nije obična datoteka
-w file	da li je datoteka sa pravom upisa
-r file	da li je datoteka bez prava upisa (read-only)
-x file	da li je datoteka izvršna

Dodatno, u komandi test mogu se koristiti logički testovi za kombinovanje dva ili više uslova istovremeno:

! expression	logička negacija
exp1 -a exp2	logička AND funkcija
exp1 -o exp2	logička OR funkcija

if-then-else-fi

Osnovna osobina ovog oblika if komande je postojanje jednog uslova i dve grupe komandi od kojih će, zavisno od toga da li je uslov ispunjen ili ne, izvršiti samo jedna.

Sintaksa if-then-else-fi strukture je:

```
if  condition
then
    command1-1
    ...
    ...
else
    command2-1

    ...
    ...
fi
```

Ukoliko je uslov condition ispunjen (izraz je tačan, ili je izlazni status komande 0), izvršiće se komanda ili grupa komandi command1, a ako nije izvršiće se command2.

U nastavku teksta dat je jednostavan shell program koji određuje da li je numerički argument pozitivan ili negativan.

```
#
# ss6: Da li je argument pozitivan ili negativan broj ?
#
if [ $# -eq 0 ]
then
     echo "$0 : Morate navesti jedan numerički argument"
     exit 1
fi
if test $1 -gt 0
then
     echo "$1 je pozitivan broj"
else
     echo "$1 je negativan broj"
fi
```

Program se pokreće komandom: bash ss6 arg, gde je arg numerička vrednost.

```
$ bash ss6 5
5 je pozitivan broj
$ bash ss6 -4
-4 je negativan broj
$ bash ss6
ss6: Morate navesti jedan numerički argument
```

```
$ bash ss6 0
0 je negativan broj
```

Program proverava broj ulaznih argumenata - ako je bilo koji argument prosleđen programu tada ($#) nije jednak 0 i dalje se nastavlja testiranje znaka broja. U protivnom, izvršiće se komanda echo "$0 : Morate navesti jedan numerički argument", a zatim exit 1, čime se prekida program sa izlaznim statusom 1, koji govori o otkazu programa. Ako je argument prosleđen programu prelazi se na drugu if strukturu, koja će odrediti da li je broj pozitivan ili negativan, pri čemu se 0 tretira kao negativan broj.

if-then-elif-else-fi

Ovaj oblik if strukture predstavlja proširenje prethodnog slučaja dodatnim uslovima i dodatnim grupama naredbi koji se mogu izvršiti. Osim prvog uslova i else grupe u strukturu se uvodi N-1 novih uslova i isto toliko novih grupa naredbi, od kojih se izvršava samo jedna, zavisno od toga koji je uslov tačan. Ukoliko nijedan uslov nije ispunjen izvršiće se grupa komandi u else bloku.

Sintaksa if-then-elif-else-fi strukture je:

```
if condition1
then
    command1
    ...
elif condition2
then
    command2
    ...
elif conditionN-1
then
    commandN-1
    ...
else
    commandN
    ...
fi
```

U nastavku teksta dat je jednostavan shell program koji određuje da li je argument numerički, i ukoliko jeste, da li je pozitivan, negativan ili nula.

```
#
# ss7: Da li je argument pozitivan, negativan ili nula
#
if [ $1 -gt 0 ]
then
     echo "$1 je pozitivan broj"
elif [ $1 -lt 0 ]
then
     echo "$1 je negativan broj "
elif [ $1 -eq 0 ]
then
     echo "Argument je nula"
else
     echo "$1 nije numerički argument"
```

```
fi
```

Program se pokreće komandom: bash ss7 arg:

```
$ bash ss7 1
1 je pozitivan broj
$ bash ss7 -2
-2 je negativan broj
$ bash ss7 0
Argument je nula
$ bash ss7 a
ss7: [: -gt: unary operator expected
ss7: [: -lt: unary operator expected
ss7: [: -eq: unary operator expected
a nije numerički argument
```

U poslednjem slučaju program obavlja tri poređenja celih brojeva sa nečim što nije broj i zato daje tri poruke o greškama, a tek posle toga štampa informaciju "a nije numerički argument".

Uslovna konstrukcija case

Naredba case je dobra alternativa višeslojnoj if-then-else-fi stukturi. Fleksibilnija je i lakša za pisanje koda. Kompaktna sintaksa case komande je:

```
case word in
[ [(] pattern [| pattern]...) command-list ;;]...
esac
```

Naredba case selektivno izvršava listu komandi koja odgovara prvom uzorku koji je identičan sa promenljivom word. Karakter `|' se koristi da razdavaja višestruke uzorke, a karakter `)' služi da označi kraj jedne liste uzoraka. Lista uzoraka i pridružena lista komandi naziva se član ili klauzula (clause) case naredbe. Svaki član se mora završiti karakterom `;;'. Ulaznu promenljivu word moguće je podvrgnuti tilda proširenju, parametarskom proširenju, komandnoj zameni, aritmetičkom proširenju i upotrebi navodnika pre nego što se zada pretraživanje uzoraka u case naredbi. Takođe, svaki uzorak moguće je podvrgnuti tilda proširenju, parametarskom proširenju, komandnoj zameni i aritmetičkom proširenju. Broj case članova je proizvoljan, a svaki član se završava sa `;;'. Prvi uzorak sa kojim se nalazi podudarnost u case naredbi određuje listu komandi koja će se izvršavati.

U nastavku teksta dat je pregledniji oblik sintakse:

```
case  $variable-name  in
    pattern1)
        command
        ...
        ...
        command;;
    pattern2)
        command
        ...
        ...
        command;;
```

```
    patternN)
        command
        ...
        ...
        command;;
    *)
        command
        ...
        ...
        command;;
esac
```

Promenljiva $variable-name upoređuje se sa svim uzorcima do prvog podudaranja, nakon čega shell izvršava sve naredbe u tom bloku, zaključno sa naredbom iza koje se nalaze dve dvotačke ;;. U slučaju da nema podudaranja izvršava se grupa naredbi iza *) (default).

Povratni status cele case strukture je 0, ukoliko nijedan uzorak ne odgovara ulaznoj promenljivoj word. U suprotnom, povratni status jednak je izlaznom statusu poslednje naredbe iz komandne liste koja će se izvršiti.

Korišćenje case strukture demonstrirano je programom ss8:

```
#
# ss8: korišćenje case strukture
#
if [ -z $1 ]
then
    echo "*** Unesite korisničko ime ***"
    exit 1
fi
echo -n "Korisnik sistema: "
case $1 in
     "jsmith") echo "John Smith, jr.";;
     "nmacek") echo "Nemanja Maček";;
     "bora") echo "Borislav Đorđević";;
     "dragan") echo "Dragan Pleskonjić";;
     *) echo "*** Nepostojeći korisnik ***";;
esac
```

Program najpre proverava da li je zadat ulazni argument, a zatim na osnovu njegove vrednosti prikazuje na ekranu odgovarajuću poruku.

Petlje

while petlja

Računar može izvršavati grupu instrukcija više puta sve dok su ispunjeni izvesni uslovi. Grupa instrukcija koja se ponavlja zove se petlja (loop). Bash komandni interpreter podržava until, while, for i select petlje. Naznačimo da pojava simbola ";" u opisu sintakse komande znači da ";" može biti zamenjena jednim ili više novih redova od kojih svaki sadrži komandu. Interne naredbe komandnog interpretera break i continue mogu se koristiti za kontrolu izvršenja petlji.

Komanda while izvršava grupu komandi u petlji sve dok su ispunjeni odgovarajući uslovi. Kompaktna sintaksa while petlje:

```
while test-commands;
do
   consequent-commands;
done
```

While petlja se izvršava na sledeći način: najpre se provere uslovi, i ako su ispunjeni (izlazni status grupe test-commands je 0), izvršava se grupa komandi consequent-commands. Petlja prestaje sa izvršavanjem ako uslov više nije ispunjen, odnosno kada izlazni status grupe test-commands postane različit od 0. Izlazni status while petlje jednak je izlaznom statusu zadnje izvršene komande u grupi consequent-commands, ili 0, ako nijedna komanda iz grupe nije izvršena (na primer, ako uslovi nisu odmah ispunjeni, pa petlja praktično nema ni jednu iteraciju).

Pregledniji oblik sintakse while petlje je:

```
while [ condition ]
do
    command1
    command2
    ...
    commandN
done
```

Program ss9 demonstrira upotrebu while petlje:

```
#
# ss9: tablica množenja realizovana while petljom
#
if [ $# -eq 0 ]
then
    echo "Greška - numerički argument nije naveden"
    echo "Sintaksa : $0 broj"
    echo "Program prikazuje tablicu množenja za dati broj"
    exit 1
fi
n=$1      # Postavi vrednost argumenta u promenljivu n
i=1       # Inicijalizacija promenljive i (brojača)
while [ $i -le 10 ]     # Uslov petlje - blok se izvršava dok je
i<10
do
    echo "$n * $i = `expr $i \* $n`"  # Prikazuje npr. 6*4=24
    i=`expr $i + 1`     # Inkrementira promenljivu i
done
```

Na osnovu ovog primera mogu se uočiti tri bitne činjenice vezane za petlje:

- promenljiva koja se koristi u uslovu petlje mora da se inicijalizuje pre početka petlje,
- provera uslova se obavlja na početku svake iteracije (condition),
- telo petlje završava se ili mora da sadrži naredbu koja modifikuje vrednost promenljive koja se koristi u uslovu, inače je petlja beskonačna (nema izlaska iz petlje).

until petlja

Until petlja, potpuno suprotno while petlji, izvršava grupu komandi u petlji sve dok se odgovarajući uslovi ne ispune. Sintaksa until komande je:

```
until test-commands;
do
    consequent-commands;
done
```

until petlja se izvršava na sledeći način: najpre se provere uslovi i ako nisu ispunjeni (izlazni status grupe test-commands je različit od 0), izvršava se grupa komandi consequent-commands. Petlja prestaje sa izvršavanjem kad se uslovi ispune, odnosno kada izlazni status grupe test-commands postane 0. Izlazni status until petlje jednak je izlaznom statusu poslednje izvršene komande iz grupe consequent-commands, ili 0, ako nijedna komanda iz grupe nije izvršena (na primer, ako su uslovi odmah ispunjeni, pa petlja praktično nema ni jednu iteraciju).

Program ss10 demonstrira upotrebu while petlje:

```
#
# ss10: upotreba until petlje
#
c=20
until [  $c -lt 10 ]
do
    echo c = $c
    let c-=1
done
```

for petlja

Sintaksa for petlje ne liči mnogo na sintaksu C jezika:

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

Komanda for će razviti listu words, i za svakog člana redom u rezultantnoj listi će izvršiti grupu komandi commands, pri čemu promenljiva name dobija vrednost tekućeg člana liste. 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.

Program ss11 demonstrira upotrebu for petlje:

```
#
# ss11: upotreba for petlje
#
if [ $# -eq 0 ]
then
    echo "Greška - numerički 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
```

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.

Alternativna forma for petlje podseća na sintaksu for petlje programskog jezika C:

```
for (( expr1 ; expr2 ; expr3 )) ;
do
    commands ;
done
```

U ovoj konstrukciji najpre se izračunava aritmetički izraz expr1 saglasno pravilima za aritmetiku komandnog interpretera, čime se obično postavljaju početni uslovi. Petlja funkcioniše na sledeći način: aritmetički izraz expr2 se izračunava u iteracijama sve dok ne postane 0. Svaki put kada je izraz expr2 različit od 0, izvršavaju se komande u for petlji a takođe se izračunava izraz expr3. Ako se bilo koji od tri izraza izostavi, for naredba ga tretira kao da ima vrednost 1. Izlazni status cele for petlje jednak je izlaznom statusu zadnje izvršene komande iz grupe commands, ili 0, ako je bilo koji od tri izraza pogrešno zadat (invalid).

Sledeći primer ilustruje upotrebu alternativne for petlje:

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

Naredba select

Naredba select omogućava jednostavnu konstrukciju menija. Sintaksa naredbe select je slična sintaksi for petlje:

```
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 reči 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 REPLAY.

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
```

Funkcije

Pomoću funkcija komandnog interpretera komande se mogu grupisati u imenovanu celinu. Funkcije se izvršavaju kao i ostale komande - funkcija se poziva po imenu, kao i obična shell komanda, a lista komandi koja je pridružena funkciji se izvršava. Shell funkcije se izvršavaju u tekućem shell kontekstu.

Funkcije se deklarišu pomoću sledeće sintakse:

```
[ function ]
name ()
{
    command-list;
}
```

Ova konstrukcija definiše shell funkciju po imenu name. Rezervisana reč function je opciona, a ukoliko se navede opcione su srednje zagrade. Telo funkcije čini lista komandi između vitičastih zagrada { }, i izvršava se prilikom pozivanja funkcije. Vitičaste zagrade su rezervisane reči i shell ih prepoznaje samo ako su razdvojene praznim karakterima ili

novim redovima od tela funkcije. Takođe, lista komandi command-list mora biti završena sa ; ili sa novim redom.

Prilikom izvršavanja argumenti funkcije postaju pozicioni parametri, specijalni parametar "#", koji opisuje broj pozicionih parametara, se ažurira, a ime funkcije se upisuje u promenljivu FUNCNAME. Pozicioni parametar 0 se ne menja.

Ako se povratak iz ugrađene komande izvrši u funkciji, funkcija se završava i izvršenje se nastavlja sa sledećom komandom iza funkcijskog poziva. Kada se funkcija izvrši, vrednosti pozicionih parametara i specijalni parametar # se vraćaju na vrednosti koje su imali pre izvršenja. Ako je izlazni argument specificiran, funkcija će vratiti taj argument, a u protivnom funkcija vraća status svoje poslednje izvršene komande.

Izlazni status funkcije jednak je izlaznom statusu poslednje izvršene komande u telu funkcije.

Funkcije mogu biti rekurzivne, pri čemu ne postoji limit na broj rekurzivnih poziva.

Lokalne promenljive

Lokalne varijable mogu se deklarisati unutar funkcije pomoću ključne reči local, i vidljive su samo unutar funkcije.

```
x=globalx
function myfunc {
    local x=localx
    echo $x
}
echo $x
myfunc
echo $x
```

Ovaj primer dovoljno jasno ilustruje korišćenje lokalnih promenljivih.

Primeri složenijih shell programa

Backup home direktorijuma. Promena imena grupe datoteka.

U nastavku teksta data su dva složenija primera shell programa.

Backup home direktorijuma

Program hbackup kreira backup home direktorijuma za jednog korisnika, čije se korisničko ime navodi kao parametar, ili za sve korisnike, ukoliko se kao parametar navede allusers. Backup se kreira pomoću programa tar i smešta u direktorijum /var/hbackup, ukoliko korisnik ne navede drugi direktorijum pomoću argumenta -d dest_dir. Ime backup datoteke formira se na osnovu imena korisnika čiji se backup kreira, i tekućeg datuma. Program zahteva bash za korektno izvršenje.

```
#!/bin/bash
# hbackup: backup home direktorijuma
if [ -z $1 ]; then
    echo "$0 [-d dest_dir] allusers (back-up home direktorijuma
svih korisnika)"
    echo "$0 [-d dest_dir] user (back-up home direktorijuma
korisnika user)"
    echo "$0: Ako se ne specificira direktorijum, koristi se
/var/hbackup"
    exit 0
fi

if [ "$1" = "-d" ]; then
    if [ -z $2 ]; then
        echo "$0: niste naveli putanju home direktorijuma"
        exit1
    fi
    hdir = $2
    if [ -z $3 ]; then
        echo "$0: niste naveli ime korisnika ili allusers"
        exit1
    elif [ "$3" = "allusers" ]; then
        hmode = allusers
    else
        hmode = oneuser
        hname = $3
    fi

else
    hdir = "/var/hbackup"
    if [ -z $1 ]; then
        echo "$0: niste naveli ime korisnika ili allusers"
        exit1
    elif [ "$1" = "allusers" ]; then
        hmode = allusers
    else
        hmode = oneuser
        hname = $1
    fi
fi

if  [ "$hmode" = "allusers" ]
    for hname in $( ls /home )
        do
        echo "Korisnik: $hname, datoteka: $hdir/$hname-$(date
+%Y%m%d).tar.gz"
        tar -czf  $hdir/$hname-$(date +%Y%m%d).tar.gz /home/$hname/
        exit 0
    done

else
    tar -czf  $hdir/$hname-$(date +%Y%m%d).tar.gz /home/$hname/
    exit 0
fi
```

Promena imena grupe datoteka

Program renamer vrši promenu imena datoteke ili grupe datoteka. Imena se mogu menjati dodavanjem prefiksa, sufiksa i zamenom niza karaktera u imenima datoteke. Program zahteva bash za korektno izvršenje.

```
#!/bin/bash
# renamer: promena imena grupe datoteka
if [ $1 = p ]; then
    prefix=$2 ; shift ; shift
    if [$1 = ]; then
        echo "$0: niste naveli ime datoteke(a)."
        exit 0
    fi
    for file in $*
        do
        mv ${file} $prefix$file
    done
    exit 0
fi

if [ $1 = s ]; then
    suffix=$2 ; shift ; shift
    if [$1 = ]; then
        echo "$0: niste naveli ime datoteke(a)."
        exit 0
    fi
    for file in $*
        do
        mv ${file} $file$suffix
    done
    exit 0
fi

if [ $1 = r ]; then
    shift
    if [ $# -lt 3 ] ; then
        echo "Sintaksa: $0 r [izraz] [zamena] datoteka(e) "
        exit 0
    fi
    OLD=$1 ; NEW=$2 ; shift ; shift
    for file in $*
        do
        new=`echo ${file} | sed s/${OLD}/${NEW}/g`
        mv ${file} $new
        done
    exit 0
fi

echo "Sintaksa:"
echo "$0 p [prefiks] datoteka(e)"
echo "$0 s [sufiks] datoteka(e)"
echo "$0 r [izraz] [zamena] datoteka(e)"
exit 0
```