Sveučilište J. J. Strossmayera u Osijeku

Fakultet elektrotehnike, računarstva i informacijskih tehnologija

Kneza Trpimira 2b

HR-31000 Osijek

www.ferit.unios.hr

Laboratorijska vježba 3:

Razvoj skripti za ljusku operacijskog sustava Linux

Sadržaj

1. UVOD

Današnje distribucije operacijskog sustava Linux uglavnom sadržavaju grafičko korisničko sučelje, no velik dio korisnika tog operacijskog sustava preferira komandno sučelje, tj. korištenje ljuske. Komandno sučelje ljuske je mnogo brže, moćnije i opsežnije za korištenje ako je korisnik dobro upoznat sa svim pripadnim funkcionalnostima i naredbama. Najpoznatija ljuska operacijskog sustava Linux je **Bash** (***B**ourne **A**gain **SH**ell*) ljuska koju je napisao Steve Bourne, tada član Bell laboratorija. Mnoge druge ljuske su također u upotrebi (*ksh, csh, …*), ali Bash je uobičajena ljuska za Linux sustave.

Operacijski sustav Linux je najkorišteniji i jedan od najvažnijih operacijskih sustava zadnjih nekoliko desetljeća. Programiranje u Linuxu je širok termin koje se proteže kroz nekoliko područja: modifikacija jezgre, razvoj modula jezgre, razvoj korisničkih aplikacija za Linux i razvoj skripti za ljusku. Razvoj skripti za ljusku (engl. *shell scripting*) operacijskog sustava Linux je najmanje nalik klasičnom programiranju u odnosu na ostala područja. Međutim, skripte za ljusku su odličan način za automatizaciju rutinskih zadataka i izvođenje posla u kraćem vremenu. Osim toga, napredno skriptiranje zahtijeva i dublje razumijevanje operacijskog sustava te razdvaja početnike Linuxa od naprednih Linux korisnika.

Umijeće skriptiranja nije teško savladati, jer postoji prilično mali skup operatora i akcija ljuske koji mogu činiti skriptu. U ovoj laboratorijskoj vježbi ćemo objasniti osnovnu strukturu skripte za Bash ljusku, prikazati neke naprednije koncepte i način pozivanja takvih skripti. Za pisanje skripti koristiti ćemo uobičajene uređivače teksta za Linux distribucije, Vi, Joe i Nano.

1.1. Uređivači teksta

Osim što služe za obrađivanje čistog teksta, uređivači teksta često sadrže pakete za obradu konfiguracijskih i dokumentacijskih datoteka te za razvoj izvornog koda u raznim programskim jezicima. Distribucije operacijskog sustava Linux dolaze s nekoliko uobičajenih tekstualnih editora, pri čemu je najpoznatiji **vi**. Danas je poznatiji uređivač teksta **vim** koji je unaprijeđena verzija vi editora.

vi
(*text editor* – tekstualni editor)

Uređivač teksta **vi** (*Visual Editor*) je jedan od najpoznatijih tekstualnih editora zastupljen na svim UNIX/Linux sustavima. Sadrži tri moda rada:

- *Command Mode*
  - Korisnik dolazi u ovaj mod prilikom pokretanja editora
  - Koristi se za pokretanje kursora na ekranu i napredno editiranje
  - Iz njega se prelazi u ostala dva moda
  - Tipka 'Esc' uvijek vraća korisnika u Command Mode iz ostalih modova

Tablica 1. Naredbe za *Command mode* vi editora

Naredba	Značenje
j (ili strelica prema dolje)	Pomicanje prema dolje za jedan redak
k (ili strelica prema dolje)	Pomicanje prema gore za jedan redak
Razmaknica	Pomicanje desno (naprijed) za jedan znak

w	Pomicanje prema naprijed za jednu riječ (uključujući interpunkciju)
b	Pomicanje prema natrag za jednu riječ (uključujući interpunkciju)
$	Pomicanje na kraj retka
0 (nula) ili ^	Pomicanje na početak retka
Tipka „Enter“	Pomicanje prema dolje na početak sljedećeg retka
h (ili strelica prema lijevo)	Pomicanje prema natrag za jedan znak
l (ili strelica prema dolje)	Pomicanje prema naprijed za jedan znak

- *Entry Mode*
  - Omogućuje modifikaciju teksta učitane datoteke
  - Iz Command Mode može se preći u Entry Mode na tri načina:
  - Pritiskom na tipku ‘i’ (eng. *insert*) za umetanje teksta točno na poziciji kursora,
  - Pritiskom na tipku ‘a’ (eng. *append*) za dodavanje teksta na poziciji kursora + jedno mjesto dalje,
  - Pritiskom na tipku ‘o’ (eng. *open*) za kreiranje nove linije između linije kursora i linije ispod nje
  - Za povratak u Command Mode pritisnuti tipku Esc

Tablica 2. Naredbe za *Entry mode* vi editora

Naredba	Značenje
:w	Zapiši u međuspremnik (Pohrani promjene i nastavi raditi u **vi**-ju)
:w novi_naziv_dat	Zapiši u međuspremnik pod nazivom **novi_naziv_dat**
:wq	Zapiši u međuspremnik (pohrani promjene) i zatvori **vi**
ZZ (velikim slovima)	Pohrani promjene i zatvori **vi**. Aternativa za **:wq**
:q!	Zatvori **vi** bez bez pohranjivanja promjena
:wq!	Zapiši u međuspremnik (pohrani promjene) i zatvori **vi** (uskličnik nadjačava „read only“ prava pristupa u slučaju vlasništva nad datotekom)

- Last Line Mode
  - Omogućuje zadavanje naredbi (spremanje datoteke, izlaz i editora, pretragu, mijenjanje postavki editora,..)
  - U Line Mode ulazi se unosom ‘:’ (dvotočka)
  - Kursor se prebacuje na zadnju liniju ekrana gdje se mogu unijeti naredbe
  - /string pretražuje izraz string prema naprijed
  - ? string pretražuje izraz string prema nazad
  - Za povratak u Command Mode pritisnuti tipku Esc

Alternative vi editoru su uređivači teksta Nano i Joe. Otvaranje Nano editora se izvršava pomoću naredbe: **nano** */path/to/filename*. Sustav pomoći za Nano je prikazan na dnu ekrana. Otvaranje Joe editora se izvršava pomoću naredbe: **joe** */path/to/filename*. Za izradu zadataka na laboratorijskoj vježbi, preporuča se korištenje **nano** uređivača teksta koji se pokazao najlakšim za savladavanje iz perspektive studenata.

2. RAZVOJ BASH SKRIPTI

Ljuske operacijskih sustava su obično interaktivne, tj. izvršavaju naredbe odmah pri unošenju. Međutim, ponekad je potrebno izvršiti mnogo naredbi kao dio jedne rutine, pa ovakav interaktivan način nije prikladan. Ljuska također može primiti naredbe iz datoteke i izvršiti ih, čime se izbjegava ponovno unošenje. Te datoteke se nazivaju skripte za ljusku (engl. *shell scripts*) ili programi za ljusku (engl. *shell programs*). Svaka skripta se sprema s nastavkom *.sh*.

Skripte za ljusku operacijskog sustava brz su način prototipiranja složene aplikacije. Izvođenje čak i ograničenih podskupina funkcionalnosti za rad u obliku skripte često je prva faza u razvoju softverskih projekata. Na taj način, struktura aplikacije se može testirati i pobliže utvrditi, prije nego što se pristupi konačnom kodiranju u programskim jezicima. Skripte za ljusku imaju sintaksu kao i svaki drugi programski jezik. Sastoji se od ključnih riječi, naredbi ljuske, funkcija, naredbi za kontrolu toka i sličnih naredbi.

Najčešće primjene skripti za ljusku su:
- Automatizacija posla i izbjegavanje ponavljajućeg rada
- Za izradu rutina za stvaranje sigurnosnih kopija
- Upravljanje sustavom
- Dodavanje novih funkcionalnosti u ljusku

Prednosti skripti za ljusku su:
- Naredbe i sintaksa je jednaka kao pri direktnom unosu u komandno sučelje
- Pisanje skripti je mnogo brže od unošenja naredbi u ljusku
- Lako učenje
- Interaktivno debugiranje

S druge strane, nedostaci skripti za ljusku su:
- Sklonost skupim pogreškama, jedna pogreška može promijeniti naredbu na štetan način
- Spora brzina izvršavanja
- Nedostaci dizajna unutar sintakse jezika ili implementacije
- Nisu prikladne za velike i kompleksne zadatka
- Pružaju minimalne strukture podataka

2.1. Osnovna struktura skripte

2.1.1. Razvojni ciklus skripte

Skripte su ASCII datoteke koje sadrže jednu ili više naredbi OS-a. Pišu se po pravilima ljuske koja će ih izvršavati i mogu sadržavati komentare koji započinju znakom '#'. Razvojni ciklus Bash skripte:
- Kreiranje skripte:
    - određivanje što skripta treba raditi
    - izrada liste naredbi
    - izrada nove datoteke za skriptu
    - identifikacija ljuske koju će skripta koristiti
    - dodavanje naredbi i komentara
    - snimanje skripti
- Izvršavanje skripte

    - omogućavanje izvršavanja skripte (opcija *execute*)
    - pokretanje skripte pozivom njenog imena
- Otklanjanje pogrešaka
    - otklanjanje pogrešaka i izmjena skripte

U prvom redu skripte se definira ljuska koja će izvesti skriptu, sljedećom sintaksom: *#!/path/to/shell*. Ako se ne navede, koristi se trenutna ljuska. Skripte se uvijek izvode u podljusci (kao novi proces). Primjer 1 pokazuje izradu jednostavne skripte.

Primjer 1: Kreiranje skripte 'logiran.sh'

```
student00@linux:~$ touch logiran.sh
student00@linux:~$ ls -l logiran.sh
-rw-r--r--  1 root root 0 Apr  8 18:21 logiran.sh
student00@linux:~$ nano logiran.sh  # Otvaranje uređivača teksta nano za izradu
skripte
student00@linux:~$ cat logiran.sh
#!/bin/sh
#skripta provjerava da li je korisnik trenutno logiran na sustavu
user="$1" # postavlja za varijablu 'user' vrijednost prvog argumenta skripte
if who | grep "$user"
then
echo "$user je trenutno logiran!"
fi
student00@linux:~$
student00@linux:~$ chmod 755 logiran.sh
student00@linux:~$ ls -l logiran.sh
-rwxr-xr-x  1 root root 214 Apr  8 18:29 logiran.sh
student00@linux:~$ ./logiran.sh student
student  pts/1        Apr  1 01:01 (161.53.201.76)
student je trenutno logiran!
```

2.1.2. Specijalni znakovi

Specijalni znakovi imaju određeno meta-značenje, tj. veće značenje od značenja doslovnog znaka. Zajedno s naredbama i ključnim riječima, specijalni znakovi su građevni blokovi svake skripte. Specijalni znakovi su prikazani u Tablici 3. Osim prikazanih znakova, postoji još mnogo drugih, no za potrebe ove laboratorijske vježbe su dovoljni znakovi u Tablici 3.

Tablica 3. Specijalni znakovi u Bash ljusci

Specijalni znak	Naziv	Značenje
#	Komentar	Linije između dva uzastopna # znaka neće biti izvršene
;	Rastavljač naredbi	Razdvaja više naredbi u istoj liniji
;;	Terminator za *case* opciju	Razdvaja više slučajeva u *case* bloku
.	*dot* naredba	Izvršavanje skripte
	Dio imena datoteke	Prefiks skrivene datoteke ili razdvaja naziv od nastavka
	Podudaranje s jednim znakom	Kao dio regularnih izraza, točka odgovara jednom znaku

,	Zarez operator	Povezuje niz aritmetičkih operacija. Svaka se evaluira, a samo zadnja se vraća
****	*Escape*	Navodnici za jedan znak
/	Rastavljač putanje	Razdvaja komponente naziva datoteke
:	*null* naredba	*Do-nothing* operacija, ekvivalent *true* riječi
!	*Negator*	Negira izlaz naredbe
*****	*Asterisk (wild card*)	Odgovara bilo kojem nazivu datoteke u direktoriju
$	Zamjena varijable	Prefiks $ ukazuje na vrijednost unutar varijable
()	Grupa naredbi	Označava grupu naredbi ili inicijalizaciju polja
{}	Blok koda	Kao i kod ostalih programskih jezika, no varijabla definirana unutar bloka ostaje vidljiva izvan bloka
{xxx, yyy, zzz…}	Ekspanzija vitičastih zagrada	Lista riječi odvojenih zarezom
{a..z}	Proširena ekspanzija vitičastih zagrada	Ispis znakova između *a* i *z*
[]	Test	Izraz za test između []
	Element polja	Pristup elementu polja preko indeksa u []
	Raspon znakova	Dio regularnih izraza

2.1.3. Varijable

Varijable su programske reprezentacije podataka. Varijabla u programu nije ništa drugo nego oznaka dodijeljena lokaciji u računalnoj memoriji u kojoj se nalazi odgovarajući podatak. Nad varijablama se mogu provoditi aritmetičke operacije, manipulacije s količinama te parsiranje.

Ime varijable je posrednik za dobivanje vrijednosti koja je u njoj sadržana. Postupak referenciranja ili dohvaćanja te vrijednosti se naziva supstitucijom varijable (engl. *variable substitution*). Supstitucija varijable u skriptama za ljusku se izvodi pomoću $ specijalnog znaka. Primjer 1 prikazuje postupak ispisivanja vrijednosti varijable pomoću znaka $. Za ispis se koristi naredba **echo**. Kao i u ostalim programskim jezicima, operator pridruživanja je = . Bitno je napomenuti da s niti jedne strane operatora pridruživanja ne smije postojati praznina, tj. razmak (engl. *whitespace*).

Primjer 1: Postavljanje i ispis varijable

```
student@linux:~$ variable1=23
student@linux:~$ echo variable1
variable1
student@linux:~$ echo $variable1
23
```

2.1.4. Tipovi podataka

Za razliku od drugih programskih jezika, Bash ljuska ne razlikuje varijable prema tipu podataka. U osnovi, sve Bash varijable su nizovi znakova (eng. *strings*), a ovisno o kontekstu Bash dozvoljava provedbu aritmetičkih operacija i usporedbi. Utvrđujući faktor je informacija sastoji li se varijabla samo od znamenaka. Primjer 2 prikazuje provedbu aritmetičkih operacija nad *integer* i *string* varijablama. Nad varijablom koja se sastoji samo od znamenki se normalno provode aritmetičke operacije kao i za *integer* varijablu u drugim programskim jezicima. Ukoliko varijabla sadrži ne-numeričke znakove, njena cjelobrojna vrijednost pri prebacivanju u *integer* je 0. Isto vrijedi i za *null* i nedeklarirane varijable kako je prikazano u Primjerima 3

i 4. Naredba **let** je objašnjena kasnije, a u ovom kontekstu je dovoljno znati da služi za evaluaciju aritmetičkih izraza.

Primjer 2: Razlika između string i integer varijabli

```
a=2234   #Integer
let "a += 1"
echo "a = $a"# a = 2335

b=${a/23/BB} #Zamjena "BB" za "23" - $b je string
echo "b = $b"# b = BB35
declare -i b  # Deklariranje varijable b kao integer ne pomaže
echo "b=$b"# b = BB35
let "b += 1"# BB35 + 1
echo "b=$b"# b = 1, Bash postavlja integer vrijednost stringa na 0
```

Primjer 3: Operacije nad *null* varijablama

```
e=''  # ili e="" ili e=
echo "e = $e" $ e =
let "e + =1" # null varijabla se transformira u integer s vrijednosti 0
echo "e = $e" # e = 1
```

Primjer 4: Operacije nad nedeklariranim varijablama

```
echo "f = $f"# f =
let "f += 1"  # Nedeklarirane varijable se transformiraju u integer s vrijednosti 0
echo "f = $f"# f = 1
```

2.1.5. Specijalni tipovi varijabli

Kao i u drugim programskim jezicima, varijable mogu biti lokalne (engl. *local variables*) i varijable okružja (engl. *environmental variables*) . Lokalne varijable su vidljive samo unutar bloka koda ili funkcije. Varijable okružja utječu na ponašanje ljuske i korisničko sučelje. Općenito, svaki proces ima okružje, tj. skupinu varijabli koje može referencirati. U tom smislu, ljuska je također proces. Pri pokretanju, kreiraju se varijable okruženja ljuske. Ukoliko ljuska sadrži takve varijable, one se moraju "izvesti" (engl. *export*) u skriptu. To se postiže naredbom **export**. Skripta može "exportati" varijable samo u procese djecu, tj. samo u one naredbe ili procese koje ta skripta inicira. Skripta pokrenuta iz komandnog sučelja ne može "exportati" varijable natrag u komandno sučelje.

2.1.6. Korištenje navodnika

Korištenje navodnika (engl. *quoting*) znači ispravno navođenje *string* varijable u navodnicima. Time se štite posebni znakovi u *stringu* od reinterpretacije ili proširenja od strane ljuske. Pri referenciranju varijable, općenito je korisno napisati njeno ime unutar dvostrukih navodnika. Time se sprječava reinterpretacija specijalnih znakova unutar riječi, tj. specijalni znak $ unutar navodnika na taj način omogućava referenciranje vrijednosti varijable. Dvostruki navodnici su obavezni za sprječavanje rastavljanja riječi u rečenici (engl. *word splitting*). Argument unutar dvostrukih navodnika se promatra kao jedna riječ, iako sadrži prazne znakove. Primjer 5 pokazuje efekte korištenja navodnika sa *string* varijablom.

Primjer 5: Korištenje navodnika sa string varijablom

```
List="one two three"
for a in $List  # Razdvaja varijablu $List na dijelove po razmaku
do
  echo "$a"
done
# one
# two
# three

for a in "$List"  # Razmaci su sačuvani u jednoj varijabli
do
  echo "$a"
done
# one two three
```

Jednostruki navodnici (') rade na sličan način kao i dvostruki (""), ali ne omogućuju referenciranje varijabli. Unutar jednostrukih navodnika, svaki specijalni znak (osim ') se interpretira doslovno. Navođenje unutar jednostrukih navodnika se još naziva i strožim navođenjem.

Navođenje samih navodnika je moguće koristeći \ (*escape*) operator. Taj operator mora prethoditi specijalnom znaku kojeg ljuska treba reinterpretirati doslovno. Primjerice, echo "\"Operacijski sustavi\"" će ispisati "Operacijski sustavi". Još neki od poznatijih specijalnih znakova koji sadrže \ operator su: \n, \r, \t, \v, \b, \a, \0xx. Njihova značenja su jednaka kao i u drugim programskim jezicima.

2.1.7. Izlaz skripte

Naredba **exit** završava skriptu, kao i u C programu. Također može vratiti vrijednost, koja je potom dostupna roditeljskom procesu skripte. Svaka naredba ljuske vraća izlazni status (engl. *exit status*). Uspješna naredba vraća 0, a neuspješna broj različit od nule koji se može interpretirati kao kod pogreške (engl. *error code*). Također, funkcije unutar skripte, pa i same skripte, vraćaju izlazni status. Zadnja naredba svake skripte i funkcije je **exit** naredba. Naredba može primiti parametar koji predstavlja kod izlaznog statusa. Ukoliko se naredba pozove bez parametara, skripta vraća izlazni status zadnje naredbe u skripti (koja prethodi **exit** naredbi).

Kombinacija specijalnih znakova $? čita izlazni status zadnje izvršene naredbe. To je način na koji Bash skripta dohvaća izlazne vrijednosti (engl. *return values*), nalik čitanju povratne vrijednosti funkcije u konvencionalnim programskim jezicima.

2.1.8. Kontrola toka skripte

Svaki suvisli program, pa i skripta, provjerava stanje određenih varijabli te na temelju toga preusmjerava svoj rad. U Bash ljusci, kontrola toka se vrši pomoću **test** naredbe, [] i () zagrada te **if/then** konstrukta:

- **if/then** konstrukt testira je li izlazni status liste naredbi jednak 0 te u tom slučaju izvršava daljnje naredbe
- Naredba **[[..]]** je ekvivalent **test** naredbe, a prima argumente za usporedbu i vraća izlazni status provedene usporedbe (0 za istinu, 1 za laž)
- Naredbe **((...))** i **let ...** konstruiraju izlazni status ovisno o tome vraćaju li aritmetičke operacije koje one evaluiraju vrijednost različitu od 0. Ova naredba služi za izvođenje aritmetičkih usporedbi.

Primjeri 6 i 7 pokazuju korištenje navedenih naredbi za kontrolu toka skripte.

Primjer 6: Primjer test naredbi

```
(( 0 && 1 )) # Logički I
echo $?  #1

let "num = (( 200 || 11 ))"# Logički ILI
echo $?  #0
```

Primjer 7: Kontrola toka skripte

```
if [ uvjet1 ]   #Uvjet u if-u mora biti odvojen razmacima od zagrada [ i ]
then
  naredba1
  naredba2
elif [ uvjet2 ] # Kao i else if
then
  naredba3
  naredba4
else
  default-naredba
fi
```

Osim naredbi za kontrolu toka, postoje i operatori specijalni za testiranje datoteka, koji su prikazani u Tablici 4. Primjer 8 pokazuje upotrebu nekih od operatora u tablici.

Tablica 4. Specijalni operatori za testiranje datoteka

Operator	Značenje
-e	postoji datoteka
-f	datoteka je regularna datoteka
-s	datoteka nije prazna
-d	datoteka je direktorij
-p	datoteka je cjevovod
-h	datoteka je simbolički link
-S	datoteka je *socket*
-r	datoteka ima dozvolu za čitanje
-w	datoteka ima dozvolu za pisanje
-x	datoteka ima dozvolu za izvršavanje
-g	datoteka ima postavljenu *group-id* zastavicu
-u	datoteka ima postavljenu *user-id* zastavicu
-O	Vi ste vlasnik datoteke
-G	datoteka ima isti *group-id* kao i Vi
f1 -nt f2	datoteka f1 je novija od datoteke f2
f1 -ot f2	datoteka f1 je starija od datoteke f2
f1 -ef f2	datoteke f1 i f2 imaju teške linkove na istu datoteku

Primjer 8: Upotreba specijalnih operatora za testiranje datoteka

```
dat="/home/stud/student50/a1.txt"
if [ -d "$dat" ]    # Paziti na razmak poslije [ i prije ]
then
  echo "$dat je direktorij"
elif [ -f "$dat" ]     # Paziti na razmak poslije [ i prije ]
then
  echo "$dat je regularna datoteka"
else
  echo "$dat nije direktorij ni regularna datoteka"
fi
```

Binarni operatori za usporedbu uspoređuju dvije varijable ili količine. Usporedba *integer* i *string* varijabli se izvršava pomoću različitih operatora.

Tablica 5. Operatori za usporedbu cijelih brojeva

Operator	Značenje	Primjer
-eq	je jednak	if ["$a" -eq "$b"]
-ne	različit od	if ["$a" -ne "$b"]
-gt	veći od	if ["$a" -gt $b"]
-ge	veći ili jednak	if ["$a" -ge $b"]
-lt	manji od	if ["$a" -lt $b"]
-le	manji ili jednak	if ["$a" -le $b"]
<	manji od	(("$a" < "$b"))
<=	manji ili jednak	(("$a" <= "$b"))
>	veći od	(("$a" > "$b"))
>=	veći ili jednak	(("$a" >= "$b"))

Tablica 6. Operatori za usporedbu *string* varijabli

Operator	Značenje	Primjer
==	je jednak	if ["$a" == "$b"]
=~	odgovara regularnom izrazu	if [["$word" =~ [0-9]]]
!=	različit od	if ["$a" != "$b"]
<	manji od	if [["$a" < "$b"]]
>	veći od	if [["$a" > "$b"]]
-z	*null string* (duljine 0)	if [-z "$String"]
-n	nije *null string*	if [-n "$String"]

Višestruka usporedba se provodi korištenjem logičkih operatora **&&** i **||** kao i u drugim programskim jezicima.

2.1.9. Operatori

Kao i u većini programskih jezika, operatori Bash ljuske se dijele na: operatore pridruživanja, aritmetičke, relacijske, logičke i operatore s bitovima. Uz navedene, operator , (zarez) služi za povezivanje više aritmetičkih operacija u jednu. Relacijski i logički operatori su prikazani u

prethodnom poglavlju. Simboli i princip korištenja ostalih operatora treba biti poznat iz drugih programskih jezika (npr. C). Nadalje, prioritet operatore je također istovjetan prioritetima u programskom jeziku C.

2.1.10. Interne varijable

Bash ljuska ima ugrađene varijable koje utječu na ponašanje skripte. Primjerice, **$BASH** varijabla sadrži putanju do binarne inačice Bash ljuske, a **$BASH_ENV** sadrži varijablu okružja koja upućuje na datoteku iz koje se ljuska pokreće. Neke od zanimljivih internih varijabli su **$HOME** (*home* direktorij trenutnog korisnika), **$GROUPS** (ID grupe kojoj trenutni korisnik pripada), **$PATH** (putanja do binarnih datoteka), **$PPID** (ID roditeljskog procesa skripte), **$PWD** (radni direktorij), **$UID** (ID trenutnog korisnika).

2.1.11. Manipuliranje varijablama

Bash ljuska podržava velik broj operacija za manipuliranje varijablama. Pri tome, najveći broj operacija se odnosi na manipuliranje sa *string* tipovima podataka. Neke od najčešćih funkcija za manipuliranje *stringovima* su:

- *${#string}* - vraća duljinu stringa (ekvivalent *strlen()* funkciji u C-u)
- *"$string" : '$substring'* - provjerava sadrži li varijabla *$string* varijablu *$substring*
- *index $string $substring* - numerička pozicija u varijabli *$string* na kojoj započinje *$substring*
- *${string:position}* - ekstrakcija riječi iz varijable *$string* na poziciji *$position*
- *${string:position:length}* - ekstrakcija riječi duljine *$length* iz varijable *$string* na poziciji *$position*
- *${string#substring}* - briše najkraće podudaranje od početka *$substring* iz *$string*
- *${string##substring}* - briše najduže podudaranje od početka *$substring* iz *$string*
- *${string%substring}* - briše najkraće podudaranje od kraja *$substring* iz *$string*
- *${string%%substring}* - briše najduže podudaranje od kraja *$substring* iz *$string*
- *${string/substring/replacement}* - zamjenjuje prvo podudaranje *$substring* s *$replacement*
- *${string//substring/replacement}* - zamjenjuje sva podudaranja *$substring* s *$replacement*

Osim specifičnih operacija za manipuliranje *stringovima*, Bash ljuska sadržava i operacije zamjene parametara, koje manipuliraju i/ili proširuju varijable. Neke takve operacije su:

- *${parameter}* - vrijednost varijable *$parameter*
- *${parameter-default}* - ako *parameter* nije postavljen, vraća se *default*
- *${parameter=default}* - ako *parameter* nije postavljen, postavlja se na *default*
- *${parameter+alt_value}* - ako *parameter* je postavljen, vrati *alt_value*, inače *null*
- *${#var}* - broj znakova u *$var* (ili broj elemenata u polju *$var*)

2.1.12. Petlje

Petlja je dio koda koji ponavlja listu naredbi sve dok se ne ispuni određeni uvjet. Bash skripta razlikuje sljedeće petlje:

- **for *arg* in [*list*]** - kao *foreach* petlja, prolazi kroz svaku varijablu u listi (Primjer 9)
- **while** - neprestano izvršava naredbe unutar petlje sve dok je uvjet na vrhu petlje ispunjen (Primjer 10)
- **until** - suprotno od *while* petlje; petlja se izvršava sve dok se ne ispuni uvjet na vrhu petlje (Primjer 11)

Primjer 9: For petlja

```
# Ispis dana u tjednu
for day in Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday
do
  echo $day  # Svaki dan u zasebnoj liniji

# Ispis sadržaja datoteke
for file in $filename
do
  echo "Contents of $file"
  echo "---"
  cat "$file"
done

# Brojanje do 10
for a in 1 2 3 4 5 6 7 8 9 10
do
  echo -n "$a "
done
```

Primjer 10: While petlja

```
while  [ "$var1" != "end" ]  # Sve dok se ne unese riječ "end"
do
  echo "variable #1 = $var1"
  read var1     # Unošenje varijable var1
done
```

Primjer 11: Until petlja

```
var=0
until (( var > 10 ))
do
  echo -n "$var "
  (( var++ ))
done
```

2.1.13. Čitanje sadržaja datoteke

Čitanje sadržaja datoteke je usko povezano s korištenjem petlji. Za čitanje sadržaja koristi se naredba **read**. Opcija -r uz naredbu **read** ignorira *escape* znakove, tj. / znakove. Primjer 12 prikazuje skriptu za čitanje sadržaja datoteke liniju po liniju.

Primjer 12: Čitanje sadržaja datoteke liniju po liniju

```
while read -r line
do
  echo "$line"
done < "$1"   #  Prvi parametar s konzole

student50@linux:~$ chmod 777 skripta.sh
student50@linux:~$ ./skripta.sh brojke.txt
8
12
4
22
```

Za čitanje sadržaja datoteke riječ po riječ, potrebna je dodatna **for** petlja koja prolazi kroz svaku liniju, kako je pokazano na primjeru 13.

Primjer 13: Čitanje sadržaja datoteke riječ po riječ

```
while read -r line
do
  for word in $line
  do
    echo "$word"
  done
done < "$1"   #  Prvi parametar s konzole
```

2.2. Napredniji koncepti Bash skripte

2.2.1. Regularni izrazi

Da bi se u potpunosti iskoristila moć skriptiranja, potrebno je savladati regularne izraze (engl. *regular expressions*). Određene naredbe i alati ljuske (**grep**, **expr**, **sed**, **awk**) za argumente primaju regularne izraze. Regularni izraz je niz metaznakova koji se interpretiraju iznad svog doslovnog značenja, a predstavlja uzorke (engl. *patterns*) čije podudaranje se traži u testiranom *stringu*. Sastoji se od:

- Niza znakova (engl. *character set*) - znakovi čije se doslovno značenje zadržava
- *Anchor* - pozicija u tekstu od koje regularni izraz započinje testiranje (^, $)
- Modifikatori (engl. *modifiers*) - proširuju raspon teksta čije podudaranje regularni izraz testira (*, [], /)

Glavna svrha korištenja regularnih izraza je pretraživanje teksta i manipulacija *stringovima*. Regularni izraz traži podudaranje jednog znaka ili skupova znakova unutar *stringa*. Primjer 14 prikazuje neke slučajeve korištenja regularnih izraza.

Primjer 14: Korištenje regularnih izraza

```
grep "[0-9]\{5\}" # Odgovara prvih pet znamenaka
grep "ba?b"   # String bb ili bab s ? multiplikatorom što odgovara jednom ili nula ponavljanja
grep "S.*l"   # Započinje slovom S i završava sa slovom l (Small, Silly)
grep "N[oen]n"  # Započinje slovom N, završava slovom n, a između sadrži bilo o, e ili n (Non, Nen)
grep "lak+"  # Barem jedno pojavljivanje slova k (lake, lakkkk)
```

2.2.2. Preusmjeravanje ulaza i izlaza

Pri radu sa ulazom i izlazom u skriptama Bash ljuske, postoje tri uobičajene datoteke: standardni ulaz (*stdin*), standardni izlaz (*stdout*) i standardni izlaz za greške (*stderr*). Kao i svaka druga datoteka, one se mogu preusmjeriti. Preusmjeravanje jednostavno podrazumijeva spremanje rezultata iz datoteke, naredbe, programa, skripte ili bloka koda i slanje kao ulaza drugoj datoteci, naredbi, programu ili skripti. Svakoj otvorenoj datoteci se dodjeljuje deskriptor (eng. *file descriptor*). Deskriptori za *stdin*, *stdout*, *stderr* su 0, 1 i 2. Za otvaranje dodatnih datoteka, preostali deskriptori su od 3 do 9. Uobičajene naredbe za preusmjeravanje ulaza i izlaza su:

- > - preusmjerava *stdout* u datoteku (datoteka se kreira ako ne postoji, inače se prepisuje)
- >> - preusmjerava *stdout* u datoteku (datoteka se kreira ako ne postoji, inače se nadodaje na njen sadržaj)
- < - primanje ulaza iz datoteke
- | - povezivanje više izlaza datoteke ili naredbi

Primjer 15: Preusmjeravanja sadržaja datoteka *jedan_dva* u datoteku *jedan_dva*

```
student00@linux:~$ cat jedan dva > jedan_dva
student00@linux:~$ cat jedan_dva
jedan
dva
tri
dva
tri
```

2.2.3. Cjevovodi

Uz preusmjeravanje ulaza i izlaza, povezivanje naredbi putem cjevovoda (engl. *pipelines*) je mehanizam povezivanja različitih tokova podataka u svrhu postizanja efikasnijih i smislenih funkcionalnosti. Cjevovodi se koriste za slanje podataka iz jednog programa u drugi. Operator pomoću kojeg se to postiže je |. Operator | preusmjerava izlaz iz programa s lijeve strane u ulaz programa s desne strane. Moguće je povezivati proizvoljan broj programa (naredbi). Primjer 16 prikazuje slučaj korištenja cjevovoda.

Primjer 16: povezivanje naredbe ls i more

```
student00@linux:~$ ls -l /etc/ |more
total 1268
-rw-r--r--   1 root root      5902 Jun 28  2006 Muttrc
drwxr-xr-x   2 root root      4096 Mar  3  2003 ODBCDataSources
drwxr-xr-x   8 root root      4096 Feb 27 16:00 X11
....
--More--
```

2.2.4. Aliasi

Alias je mogućnost ljuske da se pozove naredba, modificirana naredba ili niz naredbi korištenjem nekog drugog simboličkog naziva. Najčešće se koristi za:

- zamjenu dugog naziva kratkim nazivom (npr. 'h' umjesto 'history')

- kreiranje jedne naredbe umjesto pozivanja niza naredbi (npr. 'dir' umjesto 'clear; ls -l')
- kreiranje alternativnog oblika postojeće naredbe (npr. 'rm -i' umjesto 'rm')

Sintaksa kreiranja *aliasa* je:

alias naziv=izraz
Primjer 17: Primjer korištenja aliasa
student00@linux:~$ a**lias** h=**history**
student00@linux:~$ h
1 ps
2 ps –ef
…
student00@linux:~$ unalias h
bash: h: command not found

2.3. Pozivanje skripte

Nakon sastavljanje skripte, ona se može pozvati direktno iz komandnog sučelja na dva načina:

- **bash** *scriptname* - nije preporučljivo jer onemogućuje efektivno čitanje sa *stdin* iz skripte
- **./***scriptname* - pokretanje skripte kao izvršne datoteke, prije toga joj je potrebno postaviti prava čitanja i izvršavanja (755, +rx)

Nakon testiranja i otklanjanja pogrešaka, preporučljivo je premjestiti skriptu i direktorij */usr/local/bin*, kako bi bila dostupna svim korisnicima u sustavu. Skripta se tada može pozvati samo navodeći njeno ime u komandnoj liniji: **scriptname [ENTER]**.

Prilikom pozivanja skripte iz komandnog sučelja, moguće joj je prenijeti parametre: *./scriptname 1 2 3*. Takvi parametri se nazivaju pozicijskim parametrima, jer im se pristupa slijedno po redoslijedu navođenja: **$0, $1, $2**, … U ovom primjeru parametar **$0** predstavlja ime skripte (*scriptname*), a parametri **$1**, **$2** i **$3** vrijednosti 1, 2 i 3. Osim direktnog pristupa parametru, Bash sadrži određene specijalne znakove koji daju dodatne informacije o predanim parametrima:

- $# - broj pozicijskih parametara
- "$*" - svi pozicijski parametri, promatrani kao jedna riječ
- "$@" - svaki parametar zasebno se promatra kao jedna riječ, a zajedno čine listu riječi
- $- - zastavice predane skripti
- $! - PID zadnjeg procesa koji se pokreće u pozadini
- $_ - specijalne varijable postavljene u zadnjem argumentu posljednje izvršene naredbe
- $? - izlazni status posljednje naredbe, funkcije ili skripte
- $$ - procesni ID (*PID)* same skripte

3. ZADACI ZA SAMOSTALNI RAD

Pristupiti poslužitelju linux.etfos.hr pomoću Putty SSH klijenta. Riješite sljedeće zadatke:

1. Pomoću uređivača teksta po izboru (Vi, Nano, Joe), kreirati skriptu 'dat_info.sh' koja kao parametar prima naziv datoteke te ispisuje svojstva datoteke u dugom obliku (*long; -l*) te njen sadržaj. U prvi redak skripte nije potrebno navoditi ljusku za izvođenje. Pokrenuti skriptu te kopirati njen sadržaj i izlaz.
2. Pomoću uređivača teksta po izboru (Vi, Nano, Joe) kreirati skriptu 'allowed_files.sh' koja ispisuje listu datoteka unutar trenutnog direktorija za koje prijavljeni korisnik ima pravo čitanja, pisanja i izvršavanja.
3. Pomoću uređivača teksta po izboru (Vi, Nano, Joe) kreirati skriptu 'usporedba_stringova.sh' koja kao parametre prima dva stringa. Skripta treba usporediti duljine predanih stringova te ispisati koji je veći string i njegovu duljinu. U prvi redak skripte nije potrebno navoditi ljusku za izvođenje. Pokrenuti skriptu te kopirati njen sadržaj i izlaz.
4. Pomoću uređivača teksta po izboru (Vi, Nano, Joe) kreirati skriptu 'datVSdir.sh' koja kao parametar prima string koji može predstavljati ime datoteke ili ime direktorija. Ako uneseni parametar predstavlja direktorij, skripta mora izlistati sve datoteke u direktoriju kojima je prvo slovo u imenu samoglasnik, a završavaju ekstenzijom '.txt'. Ako uneseni parametar predstavlja regularnu datoteku, skripta treba ispisati svaku riječ datoteke koja u sebi sadrži broj. U prvi redak skripte nije potrebno navoditi ljusku za izvođenje. Pokrenuti skriptu te kopirati njen sadržaj i izlaz.

3.1. Zadaci za vježbu

Ove zadatke nije potrebno riješiti tijekom laboratorijske vježbe. Oni služe isključivo za vježbanje za pismeni ispit, a svoja rješenja možete provjeriti na konzultacijama. Skripte možete napisati koristeći tekstualne editore Nano ili Vim.

1. Napisati Bash skriptu koja kao parametre prima ime datoteke, broj početne i broj završne linije te ispisuje sve linije u navedenoj datoteci između početne i završne.
2. Napisati Bash skriptu koja kao parametre prima string i ime datoteke. Skripta treba izbrisati svako pojavljivanje navedenog stringa unutar datoteke i potom ispisati sadržaj takve datoteke.
3. Napisati Bash skriptu koja kao parametar prima cijeli broj. Skripta računa i ispisuje faktorijel primljenog broja.
4. Napisati Bash skriptu koja kao parametar prima cijeli broj. Skripta treba provjeriti i ispisati je li primljeni broj prost.
5. Napisati Bash skriptu koja kao parametre prima naziv direktorija i jedan znak. Skripta treba ispisati sve datoteke u direktoriju koje započinju navedenim znakom.
6. Napisati Bash skriptu koja kao parametar prima jedan string. Skripta provjerava i ispisuje je li uneseni string palindrom.
7. Napisati Bash skriptu koja kao parametar prima naziv datoteke. Skripa treba ispisati broj znakova, riječi i linija u datoteci.
8. Napisati Bash skriptu koja kao parametre prima naziv direktorija i cijeli broj. Skripta treba preimenovati sve datoteke unutar navedenog direktorija koje u svom imenu sadrže primljeni broj tako da se vrijednost broja uveća za 1. Skripta ispisuje sadržaj primljenog direktorija.

4. LITERATURA

[1] Stallings, W., 2012. Operating systems: internals and design principles. Boston: Prentice Hall,.

[2] Tanenbaum, A.S. and Bos, H., 2015. Modern operating systems. Pearson.

[3] Cooper, M., 2014. Advanced bash Scripting Guide. Рипол Классик.

[4] https://www.whoishostingthis.com/resources/linux-programming/

[5] https://linuxjourney.com/

[6] https://www.geeksforgeeks.org/introduction-linux-shell-shell-scripting/