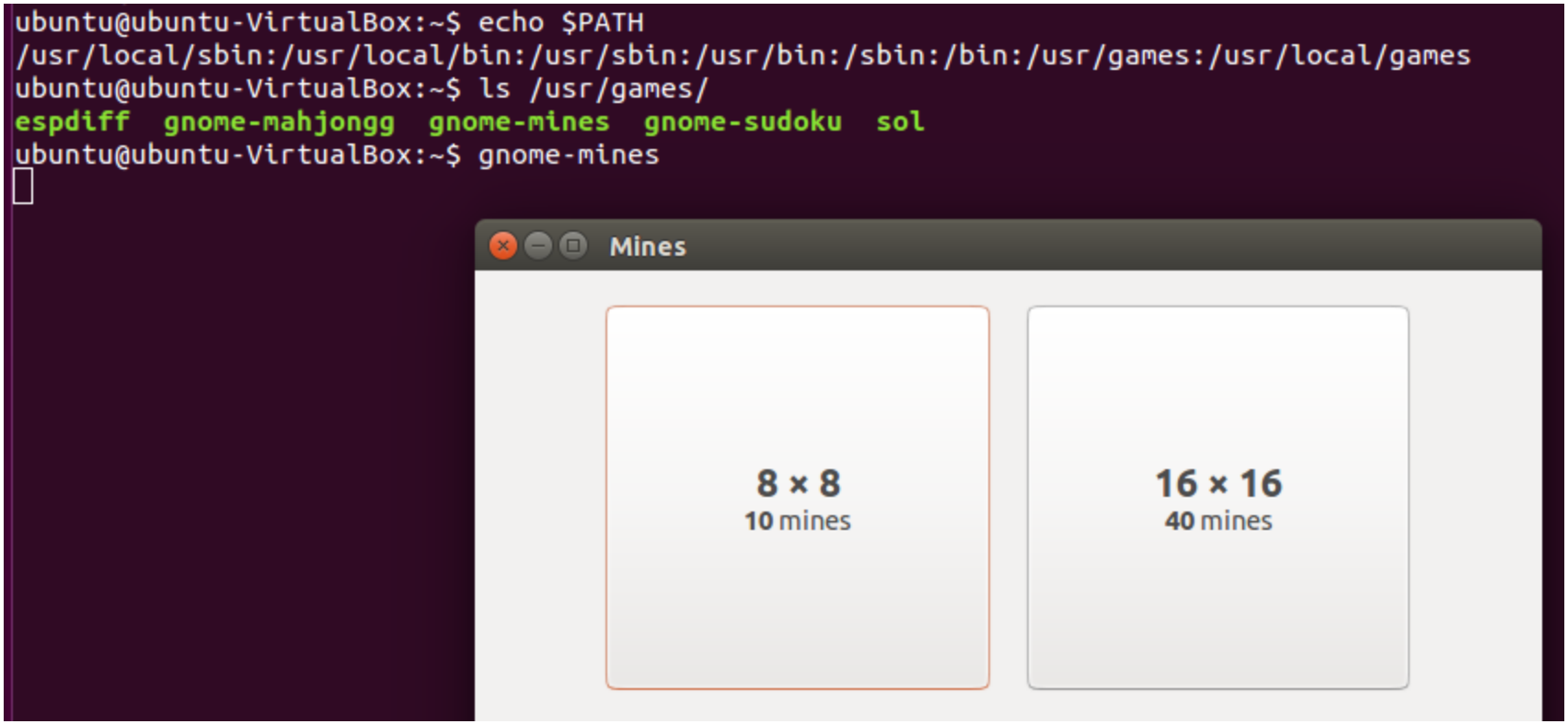
Pokretanje izvršnog fajla

- Eksterne komande koje smo do sada prešli na predavanjima su tipično bile smeštene u /bin ili /sbin direktorijumu
- Aktivirali smo komande jednostavnim kucanjem naziva komande, a koji je odgovarao nazivu izvršnog fajla te komande
- Šta se dešava ako bi izvršni fajl komande pomerili u neki drugi direktorijum? Zavisi od sadržaja promenljive $PATH.
- Ako $PATH sadrži dotični direktorijum onda bi komanda bila izvršena, a ako ne sadrži onda komanda ne bi bila izvršena.
- Takođe, ako postoji više izvršnih fajlova istog naziva, onda se izvršava onaj iz direktorijuma koji je najranije naveden u $PATH promenljivoj. Ako se želi izvršiti izvršni fajl istog naziva iz nekog drugog direktorijuma onda se mora kucati kompletna (apsolutna ili relativna) putanja tog izvršnog fajla.

Pokretanje izvršnog fajla

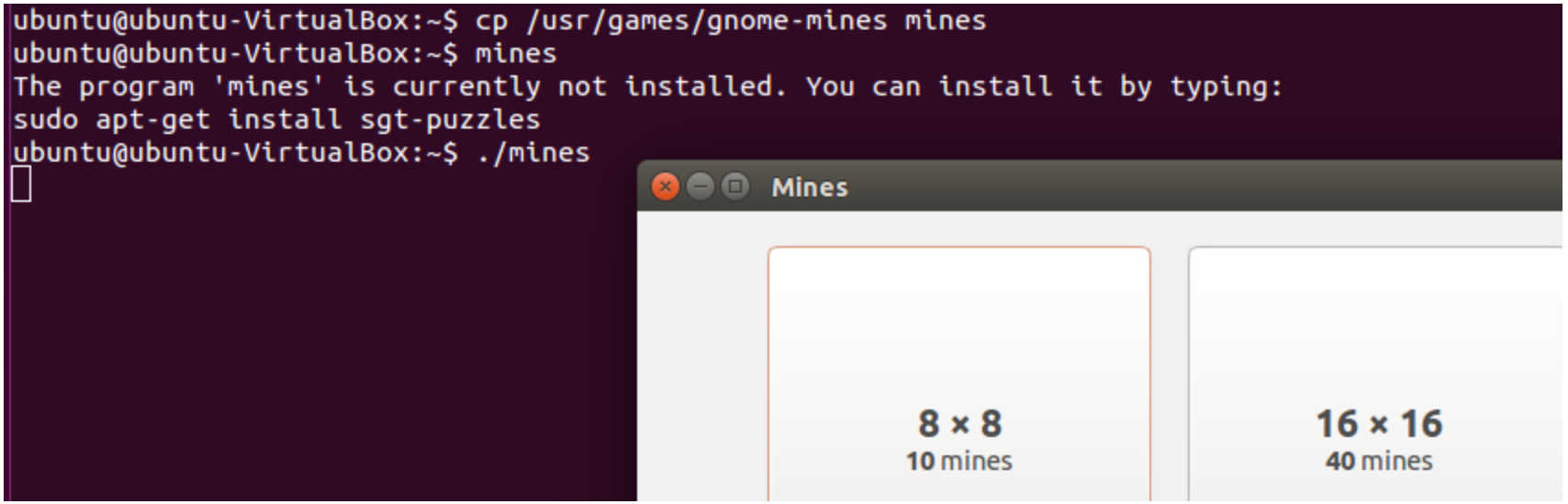
- U suštini ako /bin ne bi bio naveden u $PATH onda se komande ne bi mogle izvršavati onako kako smo do sada radili.
- Ako se izvršni fajl nalazi u direktorijumu koji nije u $PATH putanji onda se on pokreće kucanjem kompletne apsolutne putanje izvršnog fajla ili kompletne relativne putanje (pri čemu se mora navesti i trenutni direktorijum).
- Na primer, ako se izvršni fajl PRIMER nalazi u tekućem direktorijumu koji nije u $PATH, onda bi se on pokrenuo sa **./PRIMER**

Primer



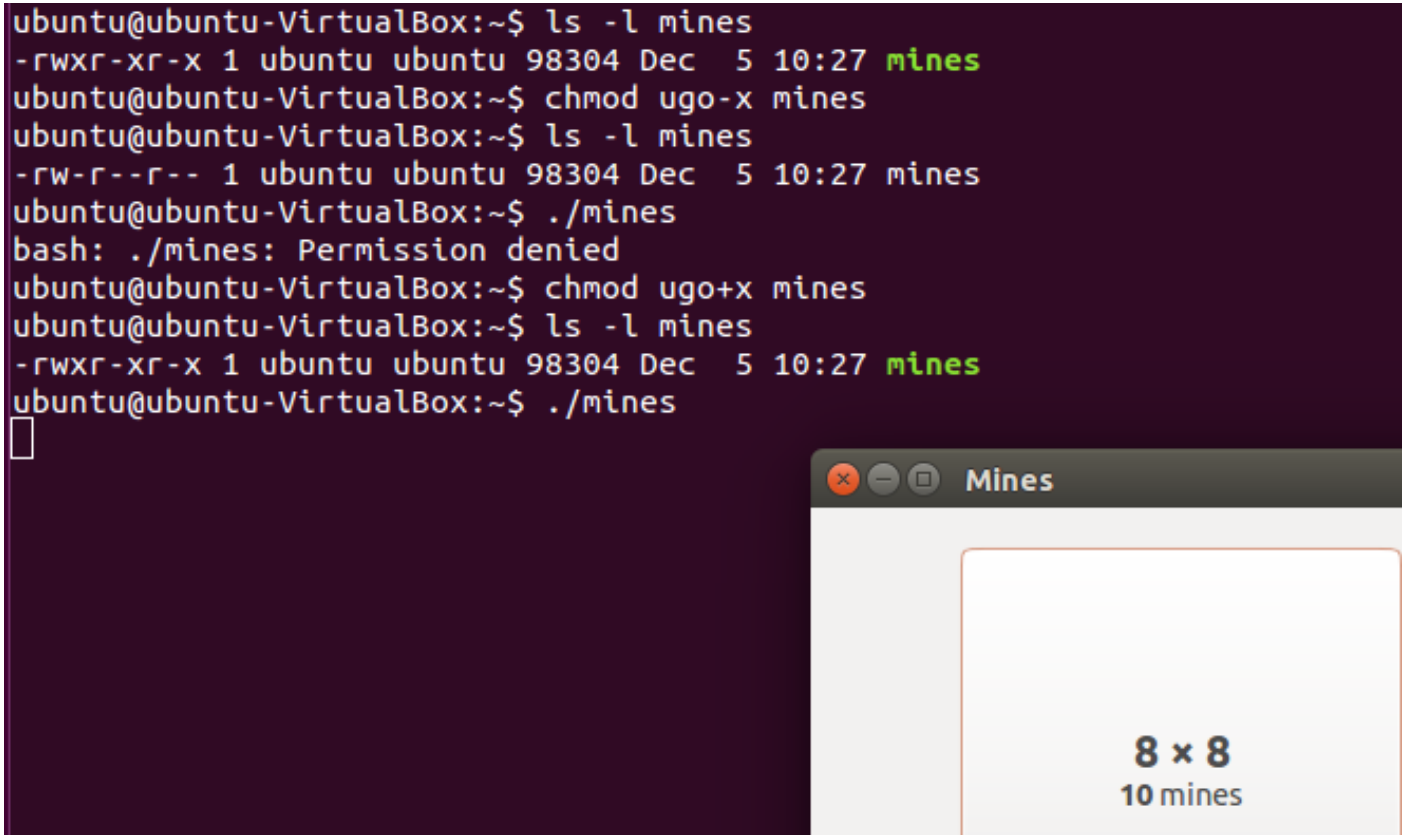
U ovom primeru je pokrenuta jedna igrica prostim kucanjem naziva izvršnog fajla jer se direktorijum u kom se nalazi izvršni fajl nalazi u $PATH. Na slici se može videti deo prozora otvorene igrice.

Primer



U ovom primeru je izvršni fajl kopiran u home direktorijum korisnika ubuntu. Naziv kopije je promenjen da se ne bi aktivirao izvršni fajl iz games direktorijuma. Home direktorijum korisnika ubuntu nije deo $PATH što se može videti sa prethodnog slajda. Sada se izvršni fajl nije pokrenuo uspešno, ali se kucanjem relativne putanje uspešno pokrenuo. Uspešno pokretanje je moglo da se uradi i sa: **/home/ubuntu/mines** tj. kucanjem apsolutne putanje. Ako bi se nalazili u /home direktorijumu tada bi pokretanje relativnom putanjom bilo **ubuntu/mines**

Podešavanja dozvola

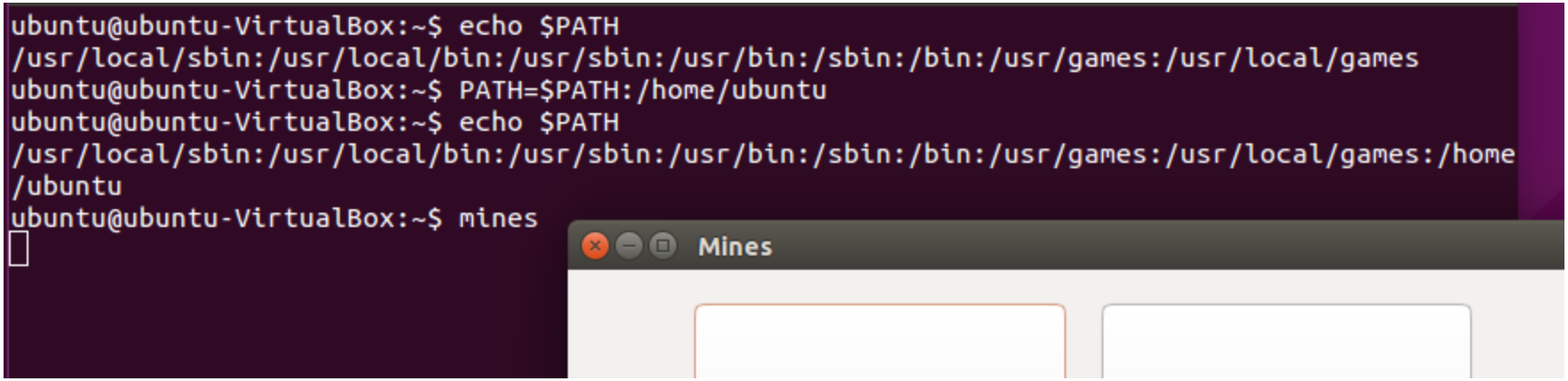


Da bi fajl mogao da se izvrši neophodno je da ima aktiviranu dozvolu za izvršavanje (x iz tripleta rwx). U datom primeru se vidi da ako se skine x dozvola, fajl ne može da se pokrene tj. izvrši.

Dodavanje direktorijuma u $PATH

- Ako se želi dodati direktorijum u $PATH onda se može uraditi ažuriranje vrednosti $PATH promenljive: **PATH=$PATH:direktorijum** gde se za direktorijum stavi apsolutna putanja željenog direktorijuma
- Navedena promena je privremena tj. kad se korisnik izloguje promena će biti izgubljena (postupak naveden u prethodnoj stavci se stoga mora ponoviti pri svakom novom logovanju)
- Ako se želi trajna promena onda je neophodno željeni direktorijum uneti u odgovarajući konfiguracioni fajl
- Na primer, /etc/environment ako se želi promena vidljiva na nivou celog sistema, odnosno odgovarajući konfiguracioni fajl iz home direktorijuma ako se želi da promena bude vidljiva na nivou korisnika

Primer



U datom primeru je dodat home direktorijum korisnika ubuntu u $PATH promenljivu. Sada je dovoljno kucati samo **mines** da bi se fajl pokrenuo tj. izvršio. Napomena: na primer, ako bi u /home/ubuntu postojao poddirektorijum test i ako u njega premestimo mines, više ne bi bilo dovoljno kucati mines već bi se trebalo kucati test/mines (ako se nalazimo u /home/ubuntu) tj. gleda se samo sadržaj direktorijuma iz $PATH promenljive, ali ne i poddirektorijumi tih direktorijuma.

Primer

```
ubuntu@ubuntu-VirtualBox:~$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/loc
al/games
ubuntu@ubuntu-VirtualBox:~$ vim .bashrc
ubuntu@ubuntu-VirtualBox:~$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/loc
al/games
```

U .bashrc fajl je dodata linija PATH=$PATH:$HOME;
Promena nije odmah vidljiva jer se .bashrc pokreće pri otvaranju terminala.

```
ubuntu@ubuntu-VirtualBox:~$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/loc
al/games:/home/ubuntu
```

Nakon ponovnog pokretanja terminala, vidljiva je promena u $PATH promenljivoj. Izmena u .bashrc je zgodna za situaciju kada se uvek pokreće terminal iz grafičkog okruženja, a ako to nije slučaj onda je potencijalno zgodnije neki drugi konfiguracioni fajl da se edituje (tipično je u pitanju fajl .profile ili fajl sličnog naziva, ali uvek je potrebno proveriti i nazive samih konfiguracionih fajlova i koji se od njih zaista čitaju prilikom aktivacije shell-a). Za trajnu promenu na nivou sistema, a ne korisnika, pogodnije je editovati /etc/environment fajl.

Skripte

- Veoma često je poželjno automatizovati određene poslove
- Isto tako, često se koristi isti niz komandi
- Stoga je poželjno imati mogućnost da se u neki fajl upiše ono što se želi izvršiti i da se samo pozivom fajla urade željeni poslovi
- Tu ulogu imaju skripte
- Pored komandi u skripte se mogu ubaciti i dodatne konstrukcije koje omogućavaju veću efikasnost u radu poput if-else, petlji, case struktura i sl.

Shebang

- Tipično se u prvoj liniji skripte navodi ko treba da interpretira skriptu
- Početak te linije predstavlja tzv. shebang string iza koga sledi navođenje interpretera skripte
- Shebang string je **#!**
- Primer: **#! /bin/bash** - ova linija navodi da se bash koristi za interpretiranje skripte tj. da će sve komande navedene u skripti biti izvršene u bash shell-u
- Navedena prva linije ne mora da postoji u skripti, ali je preporučljivo da se ipak stavi jer se time forsira izvršenje u okruženju u kom će skripta sigurno raditi kako treba
- Kada se pokrene skripta kao program (komanda), na osnovu prve linije će se utvrditi ko treba da izvrši skriptu i aktiviraće se navedeni program (npr. bash) a sam naziv skripte će biti prosleđen kao argument

Komentari i promenljive

- Komentari počinju sa # i važe do kraja linije
- Poželjno je pisati komentare da bi drugi lakše razumeli šta je urađeno u skripti (i šta skripta radi), ali i kao sopstveni podsetnik
- Unutar skripte se mogu definisati i promenljive
- Princip je isti kao kod shell promenljivih jer to ustvari i jesu shell promenljive
- Pristupanje vrednosti promenljive se radi preko $ karaktera
- Promenljive koje se definišu unutar skripte važe samo za vreme trajanja skripte (jer važe na nivou shell-a koji je pokrenut za izvršenje skripte)

Pokretanje skripte

- Skripta se pokreće isto kao i izvršni fajl
- Ako je direktorijum u kom se nalazi skripta u $PATH promenljivoj, dovoljno je samo navesti naziv skripte (pod uslovom da ne postoji izvršni fajl ili skripta istog naziva u direktorijumu koji je ranije naveden u $PATH)
- Ako direktorijum nije u $PATH, onda se pokreće navođenjem relativne ili apsolutne putanje, uz raniju napomenu da se u slučaju da se skripta nalazi u tekućem direktorijumu mora navesti ./ ispred naziva skripte u slučaju relativne putanje
- Takođe, potrebno je aktivirati dozvolu za izvršenje tj. dozvolu x

Primer

```
#! /bin/bash
# skripta koja definise jednu promenljivu i potom je ispisuje
var1=varijabla; #definisanje jedne promenljive
echo $var1; #echo ispis
~
```

Skripta može da se napiše npr. u vi editoru.

```
ubuntu@ubuntu-VirtualBox:~$ vim echo_skripta
ubuntu@ubuntu-VirtualBox:~$ echo_skripta
bash: /home/ubuntu/echo_skripta: Permission denied
ubuntu@ubuntu-VirtualBox:~$ chmod ugo+x echo_skripta
ubuntu@ubuntu-VirtualBox:~$ echo_skripta
varijabla
ubuntu@ubuntu-VirtualBox:~$ echo $var1

ubuntu@ubuntu-VirtualBox:~$
```

Treba aktivirati dozvolu za izvršenje tj. x. U nastavku se neće prikazivati ova aktivacija u primerima skripti.

Promenljiva $var1 se ne vidi nakon izvršenja skripte.

U ovom primeru, u $PATH se nalazi i home direktorijum pa je dovoljno navesti samo naziv skripte. U suprotnom, bi skriptu trebali da pokrenemo sa npr. **./echo_skripta**

Napomena: Tipično se skript fajlovi za bash imenuju sa ekstenzijom .sh. U primerima u ovoj prezentaciji su skripte imenovane bez te ekstenzije.

Pokretanje skripte u istom shell-u

- U prethodnom primeru smo videli da se promenljiva nije prenela (zadržala) van skripte
- Razlog je što se otvorilo shell dete u okviru koga je izvršena skripta i potom se taj podshell zatvorio i vratili smo se u roditeljski (originalni) shell
- Od ranije znamo da se promenljiva ne može preneti iz shell potomka u shell roditelja
- Da bi se skripta izvršila u istom shell-u, a ne u shell detetu, potrebno je navesti **source** ispred pokretanja skripte (tzv. sorsovanje skripte)
- U ovom slučaju sve promenljive kreirane u skripti će opstati i nakon završetka skripte

Primer

```
ubuntu@ubuntu-VirtualBox:~$ cat echo_skripta
#! /bin/bash
# skripta koja definise jednu promenljivu i potom je ispisuje
var1=varijabla; #definisanje jedne promenljive
echo $var1; #echo ispis
echo $SHLVL;#echo nivoa shell-a
```

Dodata linija koja ispisuje nivo shell-a.

```
ubuntu@ubuntu-VirtualBox:~$ echo $SHLVL
1
ubuntu@ubuntu-VirtualBox:~$ echo_skripta
varijabla
2
ubuntu@ubuntu-VirtualBox:~$ source echo_skripta
varijabla
1
ubuntu@ubuntu-VirtualBox:~$ echo $var1
varijabla
```

Nivo je 2 tj. skripta se izvršava u shell potomku.

Promenljiva $var1 se sada vidi nakon izvršenja skripte.

Nivo je 1 tj. skripta se izvršava u istom shell-u.

Opcija set -x u skripti

- Na ranijim predavanjima smo videli da setovanjem **set -x** možemo da aktiviramo prikaz kako se komada tačno izvršava
- Ako se ne koristi sorsing skripte, onda će skripta biti izvršena u shell detetu i trebalo bi na početku skripte staviti **set -x** komandu
- Ali, lakši način je navesti u prvoj liniji dodatno opciju -x tj. **#! /bin/bash -x**

Primer

Varijanta sa -x u prvoj liniji.

```
ubuntu@ubuntu-VirtualBox:~$ cat echo_skripta
#! /bin/bash -x
# skripta koja definise jednu promenljivu i potom je ispisuje
var1=varijabla; #definisanje jedne promenljive
echo $var1; #echo ispis
echo $SHLVL;#echo nivoa shell-a
ubuntu@ubuntu-VirtualBox:~$ echo_skripta
+ var1=varijabla
+ echo varijabla
varijabla
+ echo 2
2
```

Varijanta sa linijom set -x.

```
ubuntu@ubuntu-VirtualBox:~$ cat echo_skripta
#! /bin/bash
# skripta koja definise jednu promenljivu i potom je ispisuje
set -x;
var1=varijabla; #definisanje jedne promenljive
echo $var1; #echo ispis
echo $SHLVL;#echo nivoa shell-a
ubuntu@ubuntu-VirtualBox:~$ echo_skripta
+ var1=varijabla
+ echo varijabla
varijabla
+ echo 2
2
```

Primer

Ako je **set -x** aktiviran i radi se sorsing skripte onda ne treba ništa menjati u skripti jer se skripta izvršava u istom shell-u u kom je i aktivirana komanda **set -x**.

```
ubuntu@ubuntu-VirtualBox:~$ cat echo_skripta
#! /bin/bash
# skripta koja definise jednu promenljivu i potom je ispisuje
var1=varijabla; #definisanje jedne promenljive
echo $var1; #echo ispis
echo $SHLVL;#echo nivoa shell-a
ubuntu@ubuntu-VirtualBox:~$ echo_skripta
varijabla
2
ubuntu@ubuntu-VirtualBox:~$ set -x
ubuntu@ubuntu-VirtualBox:~$ source echo_skripta
+ source echo_skripta
++ var1=varijabla
++ echo varijabla
varijabla
++ echo 1
1
```

Test

- U okviru skripti često je potrebno izvršiti određeni test pa u zavisnosti od ishoda testa uraditi jedan skup operacija ili drugi
- Komanda **test** omogućava raznovrsne testove nad stringovima, integer vrednostima i fajlovima
- Posto je dva načina za navođenje testa
- Prvi način je **test izraz** gde izraz predstavlja sam test
- Drugi način je **[izraz]** gde uglaste zagrade imaju ulogu reči **test** iz prvog načina
- Test se može raditi i van skripte, ali mu je tipična primena unutar skripti

Primer

Rezultat **test** komande (preciznije izlazni kod) je 0 (što odgovara true) ili 1 (što odgovara false). Da bi se pristupilo rezultatu može se koristiti **$?** Što predstavlja izlazni kod poslednje izvršene komande. Tipično ovaj izlazni kod označava uspeh (0) ili neuspeh (1) komande, ali ovde zavisi od rezultata testa.

```
ubuntu@ubuntu-VirtualBox:~$ test 5 -gt 6
ubuntu@ubuntu-VirtualBox:~$ echo $?
1
ubuntu@ubuntu-VirtualBox:~$ test 5 -gt 4
ubuntu@ubuntu-VirtualBox:~$ echo $?
0
ubuntu@ubuntu-VirtualBox:~$ [ 5 -gt 6 ]
ubuntu@ubuntu-VirtualBox:~$ echo $?
1
ubuntu@ubuntu-VirtualBox:~$ [5 -gt 6]
[5: command not found
```

Primetiti da mora postojati razmak kod zagrada, u suprotnom dolazi do greške.

Test - logički operatori i stringovi

- -a označava logičku AND operaciju
- -o označava logičku ILI operaciju
- ! označava negaciju

- -n *string* - dužina stringa nije 0
- -z *string* - dužina stringa je 0
- *string1* = *string2* - string1 i string 2 su isti
- *string1* != *string2* - string1 i string 2 nisu isti

Test - integer

- *broj1* -eq *broj2* - broj1 = broj2
- *broj1* -ge *broj2* - broj1 >= broj2
- *broj1* -gt *broj2* - broj1 > broj2
- *broj1* -le *broj2* - broj1 <= broj2
- *broj1* -lt *broj2* - broj1 < broj2
- *broj1* -ne *broj2* - broj1 <> broj2

Test - fajlovi

- *fajl1* -ef *fajl2* - isti su fajlovi (imaju isti inode broj)
- *fajl1* -nt *fajl2* - fajl1 je noviji od fajla2 (vreme modifikacije se gleda)
- *fajl1* -ot *fajl2* - fajl1 je stariji od fajla2 (vreme modifikacije se gleda)
- -b *fajl* - tip fajla je b (block) i postoji
- -c *fajl* - tip fajla je c (character) i postoji
- -d *fajl* - tip fajla je d (direktorijum) i postoji
- -e *fajl* - fajl postoji
- -f *fajl* - tip fajla je - (regularan) i postoji
- -h *fajl* - fajl postoji i u pitanju je simbolički link

Test - fajlovi

- -s *fajl* - fajl postoji i veličina fajla je veća od 0
- -S *fajl* - tip fajla je s (socket) i postoji
- -r *fajl* - fajl postoji i aktivirana je dozvola za čitanje
- -w *fajl* - fajl postoji i aktivirana je dozvola za upis
- -x *fajl* - fajl postoji i aktivirana je dozvola za izvršavanje
- -G *fajl* - fajl postoji i grupni vlasnik je isti kao primarna grupa trenutnog korisnika
- -O *fajl* - fajl postoji i individualni vlasnik je trenutni korisnik
- .....
- Napomena: r, w i x testovi testiraju navedene dozvole za trenutnog korisnika (onog ko je pokrenuo test)

If konstrukcija

- Opšti oblik je:

if test1 **then**
  komande1
elif test2 **then**
  komande2
elif test3 **then**
  komande3
....
else
  komande*n*
fi

- Deo elif je opcion
- Isto važi i za else

Primer

```
ubuntu@ubuntu-VirtualBox:~$ cat if_primer
#! /bin/bash
# skripta koja pokazuje primer if konstrukcije
# primer jednostavne if konstrukcije
if [ -e proba ]; then
   echo fajl postoji;
else
   echo fajl ne postoji;
fi
# primer konstrukcije sa elif
if [ -d proba ]; then
  echo fajl je direktorijum;
elif [ -b proba ]; then
  echo fajl je blok fajl;
elif [ -f proba ]; then
  echo fajl je regularan fajl;
else
  echo fajl nije direktorijum, regularan fajl ni blok fajl;
fi
# else i elif deo ne moraju da postoje
if [ -s proba ]; then
  echo fajl postoji i nije prazan;
fi
```

```
ubuntu@ubuntu-VirtualBox:~$ if_primer
fajl postoji
fajl je regularan fajl
fajl postoji i nije prazan
ubuntu@ubuntu-VirtualBox:~$ ls -l proba
-rw-rw-r-- 1 ubuntu ubuntu 35 Nov  6 17:08 proba
```

Primetiti da je ovde pisana ; iza testa. Razlog je što je then pisan u istoj liniji kao i test i da bi se izbeglo tumačenje then kao dela testa neophodno je staviti ; iza testa. Što se tiče echo komandi, u tim linijama nije bilo potrebno pisati ; na kraju kao što ćemo videti na sledećem slajdu.

Primer

```
ubuntu@ubuntu-VirtualBox:~$ cat if_primer2
#! /bin/bash
# primer if gnezdenja
if [ -e proba ]
then
  if [ -s proba ]
  then
     echo fajl postoji
     echo fajl nije prazan
  else
     echo fajl postoji
     echo fajl je prazan
  fi
fi
```

```
ubuntu@ubuntu-VirtualBox:~$ if_primer2
fajl postoji
fajl nije prazan
```

Na kraju komande nije potrebno staviti ; ako iza u liniji ne postoji više ništa. Takođe, ovde je then pisan u zasebnim linijama pa nije potrebno iza testa staviti ;. Setite se da se sve ove komande izvršavaju u shell-u isto kao kada bi bile kucane.

Linija **echo fajl postoji** je mogla da se stavi ispred ugnježdenog if-a i time izbegne njeno dupliranje, ali je ovde namerno rađeno na prikazani način da bi se pokazalo da može i više komandi da se stavi unutar if-a.

Case konstrukcija

- Opšti oblik je:

case selektor **in**
  izbor1)
      komande1
      ;;
 izbor2)
      komande2
      ;;
....
 izbor*N*)
      komande*N*
      ;;
  *)
esac

- Case konstrukcija je zgodna alternativa za if konstrukciju koja sadrži velik broj elif delova
- Poslednji izbor tj. *) označava sve vrednosti selektora koje nisu pokrivene nekim od prethodnih izbora
- Izbor predstavlja vrednost selektora pri čemu se može navesti i pattern upotrebom karaktera *, ? i []
- Za izbor se može staviti i više vrednosti odvojenih sa |
- Izvršava se samo jedna selekcija (nije kao u C jeziku da se izvršavaju i selekcije iza izabrane)

Primer

```
ubuntu@ubuntu-VirtualBox:~$ cat case_skripta
#! /bin/bash
# primer case konstrukcije
echo Uneti broj u opsegu 0-9
read broj
case $broj in
   2|4|6|8)
     echo Unet je paran pozitivan broj
   ;;
   1|3|5|7|9)
     echo Unet je neparan pozitivan broj
   ;;
   0)
     echo Uneta je nula
   ;;
   *)
     echo Pogresan unos
esac
```

```
ubuntu@ubuntu-VirtualBox:~$ case_skripta
Uneti broj u opsegu 0-9
0
Uneta je nula
ubuntu@ubuntu-VirtualBox:~$ case_skripta
Uneti broj u opsegu 0-9
a
Pogresan unos
ubuntu@ubuntu-VirtualBox:~$ case_skripta
Uneti broj u opsegu 0-9
3
Unet je neparan pozitivan broj
```

Read se može iskoristiti unutar skripte da korisnik interaktivno unese vrednost promenljive. Može biti i više read poziva unutar skripte.

Primer

```
ubuntu@ubuntu-VirtualBox:~$ cat case_skripta2
#! /bin/bash
# drugi primer case konstrukcije
echo Uneti string
read tekst
case $tekst in
  ???)
    echo String je duzine 3
    ;;
  a*)
    echo String pocinje na a
    ;;
  c*c)
    echo String pocinje i zavrsava na c
    ;;
  *x)
    echo String zavrsava na x
    ;;
  w?)
   echo String duzine 2 i pocinje na w
    ;;
  [aeiou]*)
   echo String pocinje vokalom
    ;;
  [^aeiou]*)
   echo String ne pocinje vokalom
    ;;
  *)
   echo Ostalo
esac
```

```
ubuntu@ubuntu-VirtualBox:~$ case_skripta2
Uneti string
agf
String je duzine 3
ubuntu@ubuntu-VirtualBox:~$ case_skripta2
Uneti string
adcfrt
String pocinje na a
ubuntu@ubuntu-VirtualBox:~$ case_skripta2
Uneti string
effv
String pocinje vokalom
ubuntu@ubuntu-VirtualBox:~$ case_skripta2
Uneti string
vbbbgga
String ne pocinje vokalom
ubuntu@ubuntu-VirtualBox:~$ case_skripta2
Uneti string
cvvc
String pocinje i zavrsava na c
```

Primetiti da se izvršava samo jedna selekcija, tj. nije kao u C jeziku da se iza izabrane selekcije izvršavaju i sve preostale. Ako vrednosti selektora odgovara više izbora, biće izvršen onaj izbor koji je najranije naveden.

Napomena za read unos

```
ubuntu@ubuntu-VirtualBox:~$ cat read_skripta
#! /bin/bash
# kratak primer za read unos
echo -n "Uneti broj "
read broj
echo $broj
echo -n Uneti broj #stavljen je razmak na kraju
read broj
echo $broj
ubuntu@ubuntu-VirtualBox:~$ read_skripta
Uneti broj 4
4
Uneti broj4
4
```

Ako se želi da korisnik unese vrednost u istoj liniji gde je echo ispisao poruku, treba koristiti opciju -n echo komande (koja sprečava da se pređe u novu liniju nakon izvršenja echo komande). Ali, da se ne bi lepio unos sa porukom potreban je razmak na kraju echo komande, pa iz tog razloga poruku echo komande treba staviti pod navodnike da se ne bi razmak na kraju echo poruke ignorisao kao delimiter tokena.

Petlje

- Opšti oblik for petlje je:

for *iter* in **opseg**
do
komande
done

- Opšti oblik while petlje je:

while *uslov*
do
komande
done

- Opšti oblik until petlje je:

until *uslov*
do
komande
done

- Petlje **while** i **until** su veoma slične, razlika je što se **while** petlja izvršava dok je uslov ispunjen, a **until** petlja dok uslov nije ispunjen
- Uslov je ustvari test
- Ako se **do** stavi u istu liniju sa **for/while/until** onda iza uslova/opsega mora da ide ;
- Opseg u **for** petlji predstavlja opseg reči u stringu, ali može se koristiti komanda **seq** za generisanje numeričke petlje kao u drugim programskim jezicima

Primer za for petlju

```
ubuntu@ubuntu-VirtualBox:~$ cat for_skripta
#! /bin/bash
# primer za for petlju
echo petlja 1
for i in 1 aa bb 5 # navode se elementi direktno
do
  echo $i
done
echo petlja 2 # ovde su elementi rezultat ls komande na folderu test
for i in $( ls test ); do # do je u istoj liniji treba ; u liniji
  echo $i
done
echo petlja 3
for i in $( seq 1 3 ) # umesto $ () moze se koristiti i varijanta sa `
do
  echo $i
done
```

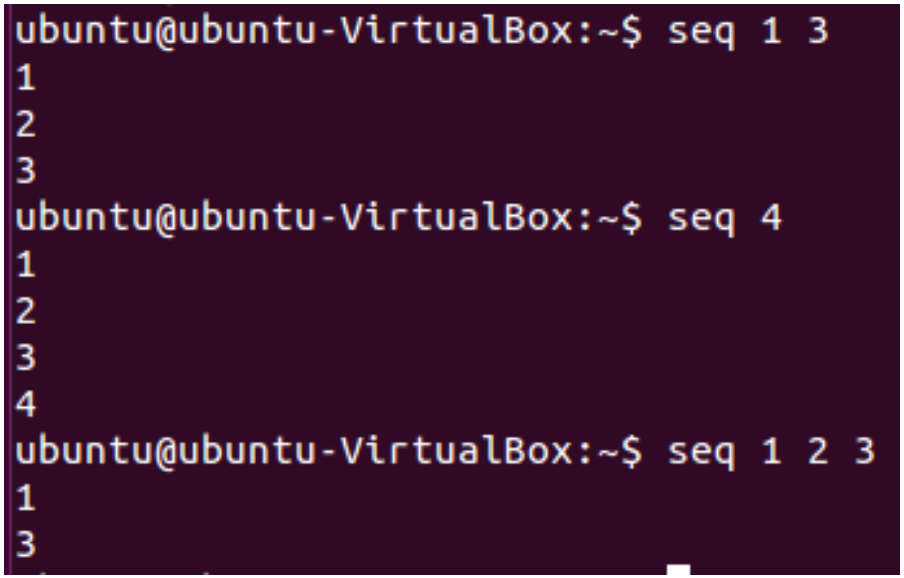
```
ubuntu@ubuntu-VirtualBox:~$ for_skripta
petlja 1
1
aa
bb
5
petlja 2
novoime
tekst
petlja 3
1
2
3
```

U prvom primeru su elementi iteracije direktno navedeni. U druga dva primera su korišćeni rezultati komandi - iz tog razloga se koristi command substitution ekspanzija. Komanda **seq** je najpribližnija onome što je for petlja u većini programskih jezika.

Komanda seq

- Komanda **seq** vrši ispis niza brojeva
- Tipične varijante su:
- **seq** *prvi poslednji* - ispisuju se brojevi od prvog do poslednjeg sa korakom 1
- **seq** *prvi korak poslednji* - ispisuju se brojevi od prvog do poslednjeg sa zadatim korakom
- Ako se navede samo *poslednji* onda se ispisuju brojevi počev od 1

Primeri


ubuntu@ubuntu-VirtualBox:~$ seq 1 3
1
2
3
ubuntu@ubuntu-VirtualBox:~$ seq 4
1
2
3
4
ubuntu@ubuntu-VirtualBox:~$ seq 1 2 3
1
3

Primer za while i until petlju

```
ubuntu@ubuntu-VirtualBox:~$ cat whun_petlje
#! /bin/bash
# primer while petlje
iter=1;
while [ $iter -lt 3 ]; do
  echo $iter
 iter=$(($iter+1))
done
#primer until petlje
iter=1;
until [ $iter -eq 3 ]; do
  echo $iter
  iter=$(($iter+1))
done
ubuntu@ubuntu-VirtualBox:~$ whun_petlje
1
2
1
2
```

U slučaju ovih petlji neophodno je vršiti manipulaciju promenljive koja se koristi za uslov da se ne bi napravila beskonačna petlja. Na primer, ovde je rađen inkrement promenljive *iter* koja je korišćena za uslov. Primetiti da se **while** petlja izvršava dok je uslov ispunjen, a **until** petlja radi obrnuto (izvršava se dok uslov nije ispunjen).

Parametri

- Skriptama se mogu pri pozivu proslediti i parametri (mogu se posmatrati kao svojevrsni argumenti skripte)
- Parametrima se pristupa sa $*i* gde *i* predstavlja redni broj parametra (brojanje počinje od 1) - ovo su tzv. pozicioni parametri
- Ako je *i* ima više od jedne cifre onda se piše ${*i*}
- $0 (0 je nula) predstavlja naziv skripte ili shell-a (ako se ovaj parametar koristi van skripte)
- $? predstavlja izlazni kod poslednje izvršene komande ili funkcije
- $# predstavlja broj pozicionih parametara
- $$ predstavlja ID procesa koji odgovara skripti

Parametri

- $! predstavlja ID procesa poslednje komande koja je stavljena u pozadinu (*background*)
- $- predstavlja trenutne opcije postavljene za shell
- $IFS predstavlja separator polja (šta se koristi kao delimiter)
- “$*” predstavlja sve pozicione parametre pri čemu se oni vide kao jedna reč
- “$@” predstavlja sve pozicione parametre ali svaki parametar se vidi kao zasebna reč
- $@ i $* predstavljaju sve pozicione parametre i to svaki parametar se vidi kao zasebna reč, ali su ti parametri podložni daljoj shell ekspanziji

Primer

```
ubuntu@ubuntu-VirtualBox:~$ cat specparam
#! /bin/bash
# ispisi specijalnih i pozicionih parametara
echo pozicioni parametar 1 je $1
echo pozicioni parametar 2 je $2
echo pozicioni parametar 3 je $3
echo naziv skripte je $0
echo izlazni kod poslednje komande je $?
echo broj pozicionih parametara je $#
echo ID procesa skripte je  $$
echo svi pozicioni parametri su $*
echo svi pozicioni parametri su $@
echo svi pozicioni parametri su "$*"
echo svi pozicioni parametri su "$@"
```

```
ubuntu@ubuntu-VirtualBox:~$ echo $$ $0
1620 bash
ubuntu@ubuntu-VirtualBox:~$ specparam par1 par2 par3
pozicioni parametar 1 je par1
pozicioni parametar 2 je par2
pozicioni parametar 3 je par3
naziv skripte je /home/ubuntu/specparam
izlazni kod poslednje komande je 0
broj pozicionih parametara je 3
ID procesa skripte je 2070
svi pozicioni parametri su par1 par2 par3
svi pozicioni parametri su par1 par2 par3
svi pozicioni parametri su par1 par2 par3
svi pozicioni parametri su par1 par2 par3
```

Primetiti da se $$ i $0 razlikuju kad su pozvani unutar skripte i u samom shell-u.

Primer

```
ubuntu@ubuntu-VirtualBox:~$ cat specparam2
#! /bin/bash
# ispisi $* i $@
echo ispis za '$*'
for i in $*
do
  echo $i
done
echo ispis za '"$*"'
for i in "$*"
do
  echo $i
done
echo ispis za '$@'
for i in $@
do
  echo $i
done
echo ispis za '"$@"'
for i in "$@"
do
  echo $i
done
```

Iz ovog primera se mogu videti razlike $@ i $# sa i bez navodnika. Tipično se koriste navodnici, pri čemu tada izbor između $@ i $# zavisi od toga da li želimo argumente da posmatramo odvojeno ili kao jedan string.

```
ubuntu@ubuntu-VirtualBox:~$ specparam2 aa bb cc
ispis za $*
aa
bb
cc
ispis za "$*"
aa bb cc
ispis za $@
aa
bb
cc
ispis za "$@"
aa
bb
cc
ubuntu@ubuntu-VirtualBox:~$ specparam2 aa "bb cc"
ispis za $*
aa
bb
cc
ispis za "$*"
aa bb cc
ispis za $@
aa
bb
cc
ispis za "$@"
aa
bb cc
```

Primetiti da su u varijanti bez navodnika bb i cc razdvojeni, a to uglavnom nije ono što želimo.

Parametri - shift

- Pozicioni parametri se mogu pomerati ulevo (od većih indeksa ka manjim)
- Koristi se komanda **shift**
- Pozicioni parametar koji je bio na indeksu 1 se gubi i više mu se ne može pristupiti

```
ubuntu@ubuntu-VirtualBox:~$ cat shiftprimer
#! /bin/bash
# primer shiftovanja pozicionih parametara
iter=1
while [ $# -gt 0 ]
do
  echo iteracija $iter argument $1
  iter=$(($iter+1))
  shift
done
ubuntu@ubuntu-VirtualBox:~$ shiftprimer aa bb cc
iteracija 1 argument aa
iteracija 2 argument bb
iteracija 3 argument cc
ubuntu@ubuntu-VirtualBox:~$ shiftprimer aa "bb cc"
iteracija 1 argument aa
iteracija 2 argument bb cc
```

let i (())

- Videli smo da pomoću $(()) možemo raditi aritmetičke kalkulacije - unutar $(()) možemo staviti i promenljive, a rezultat operacije na kraju možemo dodeliti nekoj promenljivoj što smo videli na ranijim slajdovima ove prezentacije
- Alternativa je upotreba **let** komande koja je built-in komanda
- Umesto pisanja aritmetičkih izraza unutar $(()) konstrukcije možemo pisati iste izraze iza reči (komande) **let**
- Takođe, let se može koristiti za dodelu vrednosti, pri čemu se vrši automatska konverzija u decimalni broj (npr. Iz heksadecimalnog formata)
- (()) se može koristiti kao test gde su argumenti brojevi i integer promenljive

Primer

```
ubuntu@ubuntu-VirtualBox:~$ cat numprimer
#! /bin/bash
# primer za let i (())
iter=3
while (( iter > 0 ))
do
  let iter--
  #alternativa 1 let iter=iter-1
  #alternativa 2 let iter=$iter-1
  echo $iter
done
ubuntu@ubuntu-VirtualBox:~$ numprimer
2
1
0
```

Funkcije

- Dešava se da u skripti postoji na više mesta isti kod
- Iz tog razloga u skriptama se mogu kreirati i koristiti funkcije (slično kao u većini programskih jezika)
- Funkcije mogu da imaju i ulazne parametre, a izlaz funkcije je izlazni kod (sa **return** se može definisati izlazni kod funkcije)
- Ulazni parametri se prosleđuju po sličnom principu kao što se prosleđuju pozicioni parametri skripti
- Ako se u funkciji koriste promenljive definisane u skripti one se ponašaju kao globalne promenljive
- Dozvoljena je i rekurzija funkcije

Funkcije

- Sama konstrukcija je jednostavna

*ime_funkcij*e()
{
    telo funkcije
}

Primer

```
ubuntu@ubuntu-VirtualBox:~$ cat func_skripta
#! /bin/bash
# primeri funkcija
faktorijel() #racunanje faktorijela
{
  if [ $1 -eq 0 ]; then
     return 1
  else
     faktorijel $(($1-1))
     return $(($?*$1))
  fi
}
faktorijel 5 # poziv funkcije
echo $? # ispis rezultata
globf() #promena vrednosti promenljive x
{
  let x=x+1
  let y=0x10
}
x=1
echo pre poziva funkcije $x $y
globf
echo posle poziva funkcije $x $y
ubuntu@ubuntu-VirtualBox:~$ func_skripta
120
pre poziva funkcije 1
posle poziva funkcije 2 16
```

Primetiti da je u funkciji *faktorijel* korišćen izlazni kod (kome se može pristupiti preko $?) za vraćanje rezultata. Samom argumentu se pristupa preko $1. Da su postojali i drugi argumenti njima bi se pristupalo preko $2, $3, ...

Primetiti da *y* kreiran u funkciji važi (postoji) i van funkcije. Takođe, primetiti da je **let** izvršio konverziju iz heksadecimalnog zapisa u decimalni broj.

Opcije

- Skriptama se mogu proslediti i opcije, na isti način kao što se prosleđuju i komandama
- Opcijama se može finije podesiti rad skripte - na primer, koji deo koda skripte će se izvršiti ili kako će biti procesirani argumenti i sl. (kao što je slučaj i sa komandama)
- Za procesiranje opcija se koristi **getopts**
- U najprostijem formatu se poziv vrši sa **getopts** *opcije promenljiva*
- *Opcije* predstavljaju listu svih opcija koje su predviđene da skripta koristi
- *Promenljiva* predstavlja promenljivu skripte u koju će se stavljati koja opcija je navedena prilikom poziva skripte

Opcije

- Kod komandi postojalo je više formata za navođenje opcija poput -a, -ab, --dug_naziv
- U slučaju upotrebe **getopts** po difoltu se radi sa kratkim nazivima opcija, a ne sa dugim nazivima (napomenimo da postoji način da se **getopts** prilagodi i radu sa dugim nazivima opcija)
- Takođe, neke opcije zahtevaju i navođenje argumenta
- Tada se u *opcije* iza dotične opcije koja zahteva argument navodi :
- Najlakši način za prolazak kroz sve opcije navedene prilikom poziva skripte je while petlja
- Prilikom poziva skripti čak i ako se koriste opcije i dalje se mogu navoditi i argumenti (pozicioni parametri) skripte iza opcija (kao i kod komandi)

Primer

```
ubuntu@ubuntu-VirtualBox:~$ cat plusminus
#! /bin/bash
# primer skripte sa opcijama
# opcija -m radi oduzimanje, a opcija -s radi sabiranje
tip=0
while getopts pm opcija; do
   case $opcija in
      m)
        if [ $tip -eq 2 ]; then
           echo Nije dozvoljena upotreba opcija m i p istovremeno
           exit 1
        else
           tip=1
        fi
      ;;
      p)
        if [ $tip -eq 1 ]; then
           echo Nije dozvoljena upotreba opcija m i p istovremeno
           exit 1
        else
           tip=2
        fi
      ;;
      *)
        echo Nevalidna opcija
   esac
done
```

Ova skripta podržava dve opcije -m i -p, gde prva opcija vrši oduzimanje, a druga opcija sabiranje. Skripta obezbeđuje da ako su obe opcije aktivirane da se prijavi greška. Takođe, prijavljuje i prisustvo nevalidne opcije, ali se ne izlazi iz skripte u toj situaciji.

Primer - nastavak

```
echo Broj argumenata sa opcijama $#
shift $((OPTIND-1)) # skidanje opcija
echo Broj argumenata nakon skidanja opcija $#
if [ $# -lt 2 ]; then
   echo Potrebna su dva argumenta
   exit
fi
if [ $tip -eq 1 ]; then
        let rezultat=$1-$2
        echo Rezultat oduzimanja $rezultat
elif [ $tip -eq 2 ]; then
        let rezultat=$1+$2
        echo Rezultat sabiranja $rezultat
else
        echo Nije zadata validna opcija
fi
```

Nastavak skripte sa prethodnog slajda.

Primeri rada skripte.

```
ubuntu@ubuntu-VirtualBox:~$ plusminus -m 2 3
Broj argumenata sa opcijama 3
Broj argumenata nakon skidanja opcija 2
Rezultat oduzimanja -1
ubuntu@ubuntu-VirtualBox:~$ plusminus -p 2 3
Broj argumenata sa opcijama 3
Broj argumenata nakon skidanja opcija 2
Rezultat sabiranja 5
ubuntu@ubuntu-VirtualBox:~$ plusminus -p -m 2 3
Nije dozvoljena upotreba opcija m i p istovremeno
ubuntu@ubuntu-VirtualBox:~$ plusminus -pm 2 3
Nije dozvoljena upotreba opcija m i p istovremeno
ubuntu@ubuntu-VirtualBox:~$ plusminus -pn 2 3
/home/ubuntu/plusminus: illegal option -- n
Nevalidna opcija
Broj argumenata sa opcijama 3
Broj argumenata nakon skidanja opcija 2
Rezultat sabiranja 5
ubuntu@ubuntu-VirtualBox:~$ plusminus -n 2 3
/home/ubuntu/plusminus: illegal option -- n
Nevalidna opcija
Broj argumenata sa opcijama 3
Broj argumenata nakon skidanja opcija 2
Nije zadata validna opcija
```

Opcije se mogu zadavati odvojeno (-p -m), ali i zajedno (-pm). Primetiti da je skidanje opcija potrebno da bi se pristupilo argumentima počev od $1 (pogledati ispis $#, takođe videti isto na sledećem slajdu). OPTIND promenljiva nakon procesiranja svih opcija ukazuje na indeks prvog argumenta koji nije opcija, pa **shift $((OPTIND-1))** vrši pomeranje za onoliko mesta koliko je bilo opcija.

Primer - nastavak

```
ubuntu@ubuntu-VirtualBox:~$ plusminus -n -t -m 2 3
Nevalidna opcija
Nevalidna opcija
OPTIND vrednost 4
Broj argumenata sa opcijama 5
Broj argumenata nakon skidanja opcija 2
Rezultat oduzimanja -1
ubuntu@ubuntu-VirtualBox:~$ plusminus -ntm 2 3
Nevalidna opcija
Nevalidna opcija
OPTIND vrednost 2
Broj argumenata sa opcijama 3
Broj argumenata nakon skidanja opcija 2
Rezultat oduzimanja -1
ubuntu@ubuntu-VirtualBox:~$ plusminus -mn -t 2 3
Nevalidna opcija
Nevalidna opcija
OPTIND vrednost 3
Broj argumenata sa opcijama 4
Broj argumenata nakon skidanja opcija 2
Rezultat oduzimanja -1
```

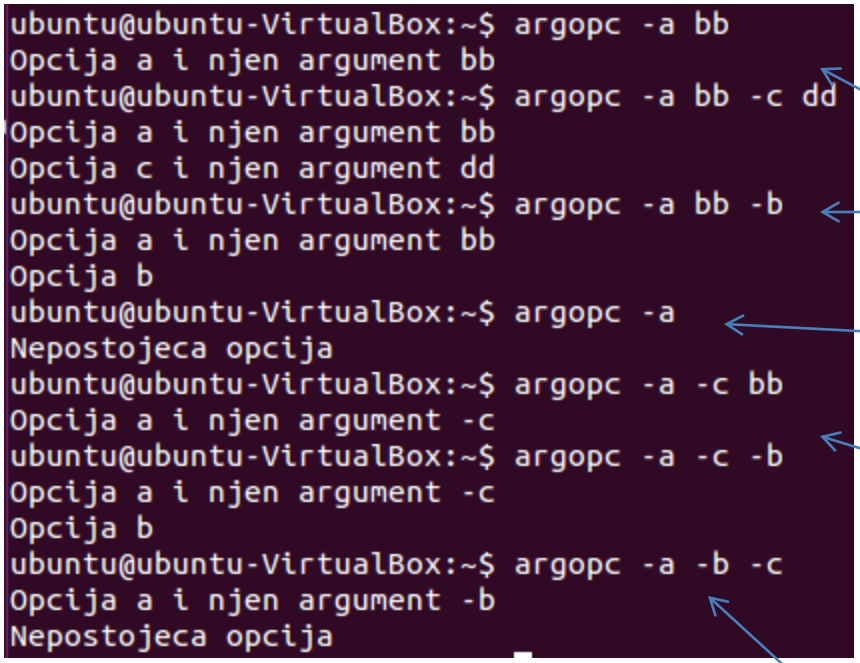
U ovom primeru je korišćena skripta sa prethodna dva slajda, sa dve male modifikacije. Prva modifikacija je u liniji while getopts :pm... (dodata je dvotačka), a druga modifikacija je ispis OPTIND promenljive nakon while petlje. Stavljanje dvotačke ispred liste argumenata (prva modifikacija) potiskuje prijavljivanje greške kad se naiđe na nepoznat parametar. Primetiti da na ovom slajdu nije prijavljivana greška nepostojeće opcije kao na prethodnom slajdu. Takođe primetiti da OPTIND vrednost zavisi od načina pozivanja parametara. Nije isto -ntm i -n -t -m.

Primer

```
ubuntu@ubuntu-VirtualBox:~$ cat argopc
#! /bin/bash
# jednostavan primer skripte sa opcijama koje zahtevaju argumente
# opcije a i c zahtevaju argumente, a opcija b ne
while getopts :a:bc: opc
do
  case $opc in
    a)
      echo Opcija a i njen argument $OPTARG
    ;;
    b)
      echo Opcija b
    ;;
    c)
      echo Opcija c i njen argument $OPTARG
    ;;
    *)
      echo Nepostojeca opcija
  esac
done
```

Primer rada sa opcijama koje zahtevaju argumente.
Promenljiva OPTARG sadrži argument opcije.

Primer



Ispravni primeri.

Nije naveden argument.

U ova dva primera je -c protumačen kao argument opcije -a.

Ovde je -b protumačen kao argument opcije -a, a -c je protumačen kao nevalidna opcija jer nije naveden argument iza nje.