Računalniški praktikum 1

Uvod v bash

Vida Groznik

Source: Ryan‘s tutorials.

Kaj je skripta?

- Skripta „pove“ računalniku, kaj naj naredi.
- Z bash skripto povemo lupini Bash, kaj naj naredi.
- Bash skripta je **navadna tekstovna datoteka**, ki vsebuje **zaporedje ukazov**.
  - Vse, kar lahko sicer poženete v ukazni vrstici, lahko date v skripto in vse se bo izvedlo enako.
- Po konvenciji imajo Bash skripte končnico **.sh**.

Kako lahko poženem bash skripto?

- Preden zaženemo skripto, moramo nastaviti ustrezne pravice za izvajanje skripte.
  (iz varnostnih razlogov je to običajno onemogočeno)
- Skripto poženete z ukazom:

```
./scriptName.sh
```

```
user@bash: ./script.sh
bash: ./script.sh: Permission denied
user@bash: chmod 755 script.sh
user@bash: ./script.sh
Hello world!
```

- Če bi v ukazno vrstico vpisali le ime datoteke, jo bash skuša poiskati v množici direktorijev, ki so shranjeni v spremenljivki $PATH, pri čemer **ne upošteva poddirektorijev vašega trenutnega direktorija.**
  - Trenutno vrednost te spremenljivke lahko preverimo z uporabo ukaza **echo.**
- Če se skripta ne nahaja v katerem izmed direktorijev shranjenih v spremenljivki $PATH, potem jo lahko zaženete tako, da bashu poveste, kje naj jo poišče.
- Pred imenom skripte lahko podate absolutno ali relativno pot do nje.
  - **Pika (.)** je referenca do vašega **trenutnega direktorija**. Če je skripta v vašem domačem direktoriju, jo lahko poženete tudi s tem, da podate absolutno pot.

Kako izgleda bash skripta?

```
1. #!/bin/bash
2. # Vzorec bash skripte
3.
4. echo Hello World!
```

- **Vrstica 1** - specifikacija okolice.
  - **#!/bin/bash** je **vedno prva vrstica skripte**.
    Sekvenci lojtra - klicaj (**#!**) rečemo „Shebang“. Sledi ji pot do interpreterja, ki naj se uporabi za interpretacijo sledečih vrstic v dokumentu.
    (Za bash skripte bo to pot do basha.)
  - pred **#** ali med **!** in potjo do interpreterja **ni presledkov**.

- **Vrstica 2** - To je **komentar**. Vse, kar se nahaja za #, se ne izvede. Je le za našo referenco.
- **Vrstica 4** - Ukaz **echo**, ki bo izpisal na ekran besedilo, ki mu sledi.

Spremenljivke

- Spremenljivka je začasna shramba za del informacije. Obstajata dve akciji, ki jih lahko izvedemo nad spremenljivkami:
  - **Določanje vrednosti** spremenljivki.
  - **Branje vrednosti** spremenljivke.
- Če želimo izvesti branje vrednosti spremenljivke, napišemo njeno ime, pred katerim je znak $.
- Preden bash interpretira vsako vrstico naše skripte, najprej preveri, če so v njej kakšna imena spremenljivk.
- Za vsako spremenljivko, ki jo najde, nadomesti ime spremenljivke z njeno vrednostjo. Zatem požene to vrstico kode in ponovi isti proces na naslednji vrstici.
- Sintaksa:
  - Ko se **referenciramo na ali beremo spremenljivko** dodamo **znak $** pred imenom spremenljivke.
  - Ko **določamo vrednost spremenljivki, spustimo** znak **$**.
  - Nekateri pišejo imena spremenljivk z velikimi črkami, da izstopajo iz skripte. Stvar preference.

Argumenti ukazne vrstice

```
1. #!/bin/bash
2. # Vzorec skripte
3.
4. cp $1 $2
5.
6. # Preverimo vrednosti
7.
8. echo Details for $2
9. ls –l $2
```

- Za podajanje argumentov skripti, uporabljamo spremenljivke **$1** ki predstavlja **prvo spremenljivko ukazne vrstice**, **$2** predstavlja **drugo spremenljivko ukazne vrstice** itd.
- **Vrstica 4** - poženi ukaz **cp** s prvim argumentom ukazne vrstice kot virom in drugim argumentom ukazne vrstice kot ciljem.
- **Vrstica 8** - poženi ukaz **echo** za izpis sporočila.
- **Vrstica 9** - Ko je bilo kopiranje izvedeno, poženi ukaz **ls** za cilj. Vključili smo opcijo **-l**, da nam pokaže podrobnosti in se lahko prepričamo, če je bilo kopiranje uspešno izvedeno.

Posebne spremenljivke

$0 - Ime skripte bash.

$1 - $9 - Prvih 9 argumentov, ki so podani skripti.

$# - Koliko argumentov je bilo podanih skripti.

$@ - Vsi argumenti, ki so bili podani skripti.

$? - Izhodno stanje procesa, ki je bil nazadnje izveden.

$$ - ID trenutne skripte.

$USER - Uporabniško ime uporabnika, ki trenutno poganja skripto.

$HOSTNAME - Ime gostitelja na katerem teče skripta.

$SECONDS - Koliko sekund že teče skripta.

$RANDOM - Vrne drugo naključno številko vsakokrat, ko se referenciramo nanjo.

$LINENO - Vrne trenutno številko vrstico skripte.

Če v ukazno vrstico napišete ukaz **env** se vam bo izpisal seznam še ostalih spremenljivk, ki jih lahko referencirate.

Določanje vrednosti spremenljivki

```
1. #!/bin/bash
2. # Enostaven primer spremenljivk
3.
4. myvariable=Hello
5.
6. anothervar=Vida
7.
8. echo $myvariable $anothervar
9. echo
10.
11. sampledir=/etc
12.
13. ls $sampledir
```

Osnovni način določanja vrednosti spremenljivki:

`spremenljivka=vrednost`

- Med obema stranema enačaja (=) **ni presledkov**. Prav tako pred ime spremenljivke ne pišemo znaka $, ko nastavljamo njeno vrednost.
- **Vrstici 4 in 6** - določanje vrednosti spremenljivkama `myvariable` in `anothervar`.
- **Vrstica 8** - poženi ukaz **`echo`** za preverjanje vrednosti spremenljivk.
- **Vrstica 9** - poženi ukaz **`echo`** brez argumentov, da dobiš prazno vrstico.
- **Vrstica 11** - vrednost spremenljivke je pot do direktorija
- **Vrstica 13** - poženi ukaz **`ls`** in preveri vsebino direktorija, ki je shranjen kot naslov v spremenljivki `sampledir`

Vaja

Z uporabo skripte izvedi operacijo seštevanja vrednosti dveh spremenljivk in rezultat shrani v tretjo spremenljivko.

Vzemimo a=5, b=8, c=a+b?

Z uporabo skripte izvedi operacijo seštevanja vrednosti dveh spremenljivk in rezultat shrani v tretjo spremenljivko. Vrednosti spremenljivk naj bodo podane kot argumenti skripti.

Narekovaji

```
user@bash: myvar='Hello World'
user@bash: echo $myvar
Hello World
user@bash: newvar="More $myvar"
user@bash: echo $newvar
More Hello World
user@bash: newvar='More $myvar'
user@bash: echo $newvar
More $myvar
```

Ko želimo v spremenljivke shraniti **kompleksnejše vrednosti**, moramo uporabiti **narekovaje**. V normalnih razmerah bash uporabi **presledke za določanje ločenih delov**.

- Ko podamo vsebino v narekovajih, povemo bashu, da naj bo vsebina tretirana kot en del. Uporabimo lahko enojne narekovaje (') ali dvojne narekovaje (").
- **Enojni narekovaji** bodo tretirali vsak znak **dobesedno**.
- **Dvojni narekovaji** nam **dopuščajo** da naredimo **zamenjavo** (torej vključitev spremenljivk v nastavljanje vrednosti).

Zamenjava ukaza

```
1. user@bash: ls
2. bin Documents Desktop ...
3. Downloads public_html ...
4. user@bash: myvar=$( ls )
5. user@bash: echo $myvar
6. bin Documents Desktop Downloads
   public_html ...
```

Zamenjava ukaza nam dopušča, da **izhod** ukaza ali programa (kar bi bilo sicer izpisano na ekran) **shranimo kot vrednst** spremenljivke. Da lahko to naredimo, ukaz podamo **med oklepaje pred katerim podamo znak $**.

- Zamenjava ukaza je enostavna, če je izhod ukaza ena beseda ali vrstica. Če je izhod podan preko večih vrstic, se nove vrstice odstranijo, rezultat pa je podan v eni vrstici.
- **Vrstica 1** - Poženi ukaz **`ls`**. Običajno je rezultat ukaza zapisan v večih vrsticah.
- **Vrstica 4** - Ko shranimo ukaz v spremenljivko **`myvar`**, se vsi prehodi v novo vrstico izbrišejo, rezultat pa je zapisan le v eni vrstici.

Uporabnikov vnos

```
1. #!/bin/bash
2. # Vprašaj uporabnika po imenu
3.
4. echo Hello, who am I talking to?
5.
6. read varname
7.
8. echo It\'s nice to meet you $varname
```

```
user@bash: ./introduction.sh
Hello, who am I talking to?
Vida
It's nice to meet you Vida
user@bash:
```

Uporabi ukaz **`read`** in vprašaj uporabnika po vnosu. Skripta vzame vnos in ga shrani v spremenljivko.

```
read var1
```

- **Vrstica 4** - Izpiši sporočilo, ki bo povprašalo uporabnika po vnosu.
- **Vrstica 6** - Poženi ukaz **`read`** in shrani uporabnikov odgovor v spremenljivko **`varname.`**
- **Vrstica 8** - z ukazom **`echo`** izpiši še eno sporočilo, da se prepričaš, če je shranjevanje uspelo.

Uporabnikov vnos (2)

```
1. #!/bin/bash
2. # Vprašaj uporabnika po podatkih za vpis.
3.
4. read -p 'Username: ' uservar
5. read -sp 'Password: ' passvar
6. echo
7. echo Thank you $uservar we now have
   your login details
```

```
user@bash: ./login.sh
Username: vida
Password:
Thank you vida we now have your login
details
user@bash:
```

Da spremenimo način vedenja ukaza **`read`** obstaja več možnih nastavitev. Dve najpogosteje uporabljani sta:

`-p` za določitev poziva in

`-s` naredi vhod „tih“ (se ne izpiše na zaslonu).

To nam omogoča, da lahko uporabnika enostavno povprašamo po uporabniškem imenu in geslu (glej primer).

Uporabnikov vnos (3)

```
1. #!/bin/bash
2. # Prikaz delovanja ukaza read
3.
4. echo What cars do you like?
5. read car1 car2 car3
6.
7. echo Your first car was: $car1
8. echo Your second car was: $car2
9. echo Your third car was: $car3
```

```
user@bash: ./cars.sh
What cars do you like?
Jaguar Maserati Bentley Lotus
Your first car was: Jaguar
Your second car was: Maserati
Your third car was: Bentley Lotus
user@bash:
```

Ukazu **read** lahko podamo več spremenljivk. Vhod bo razdelil glede na presledke, ki se pojavijo.

- Prvi vnos bo dodeljen prvi spremenljivki, drugi vnos bo dodeljen drugi spremenljivki itd.
- Če je **več vnosov, kot je imen spremenljivk**, bodo vsi preostali vnosi dodeljeni **zadnji spremenljivki**.
- Če je **manj vnosov**, kot je imen spremenljivk, bodo preostale vrednosti spremenljivk nastavljene na **prazno ali null**.

Branje iz STDIN

```
1. #!/bin/bash
2. # Povzetek prodaje
3.
4. echo Here is a summary of the sales data:
5. echo ============================
6. echo
7.
8. cat /dev/stdin | cut -d' ' -f 2,3 | sort
```

V Linuxu je običajno, da se združuje (**pipe**) niz enostavnih ukazov in s tem oblikuje rezultat, ki ga potrebujemo. Tak način združevanja ukazov lahko uporabimo tudi v skriptah.

- Za izvajanje združevanja in preusmeritev, bash uporablja posebne datoteke.
- Vsak proces dobi svoj nabor datotek (eno za STDIN, STDOUT in STDERR) s katerimi se povežejo, ko se uporabi preusmeritev ali združevanje:
  - **STDIN** - `/dev/stdin` ali `/proc/self/fd/0`
  - **STDOUT** - `/dev/stdout` ali `/proc/self/fd/1`
  - **STDERR** - `/dev/stderr` ali `/proc/self/fd/2`
- **Vrstica 8** - uporabi ukaz **cat** na datoteki STDIN, uporabi ukaz **`cut`** pri čemer nastaviš ločevanje vnosa glede na presledek, uporabi polji 2 ter 3 in nato razvrsti izhod.

Branje iz STDIN (2)

```
1. #!/bin/bash
2. # Povzetek prodaje
3.
4. echo Here is a summary of the sales data:
5. echo ==============================
6. echo
7.
8. cat /dev/stdin | cut -d' ' -f 2,3 | sort
```

```
user@bash: cat salesdata.txt
Fred apples 20 November 4
Susy oranges 5 November 7
Mark watermelons 12 November 10
Terry peaches 7 November 15
user@bash:
user@bash: cat salesdata.txt | ./summary
Here is a summary of the sales data:
====================================

apples 20
oranges 5
peaches 7
watermelons 12
user@bash:
```

Aritmetika: funkcija **let**

```
1. #!/bin/bash
2. # Osnovna aritmetika s funkcijo let
3.
4. let a=5+4
5. echo $a # 9
6.
7. let "a = 5 + 4"
8. echo $a # 9
9.
10. let a++
11. echo $a # 10
12.
13. let "a = 4 * 5"
14. echo $a # 20
15.
16. let "a = $1 + 30"
17. echo $a # 30 + prvi argument ukazne vrstice
```

`let` je v bashu vgrajena funkcija, ki nam omogoča osnovne aritmetične operacije.

```
let <aritmetični izraz>
```

Kot lahko vidite v primeru, lahko uporabite različne formate. **Prvi del je spremenljivka** v katero shranimo rezultat.

- **Vrstica 4** – To je osnovni format. Če okoli izraza ne podamo narekovajev, **ne smemo pisati presledkov**.
- **Vrstica 7** – Narekovaji nam omogočajo, da lahko v izrazu uporabljamo presledke da je ukaz bolj razumljiv.
- **Vrstica 10** – Skrajšana različica za povečanje vrednosti spremenljivke za 1.
  Naredi isto, kot če bi napisali "a = a + 1".
- **Vrstica 16** – V izraz lahko vključimo tudi druge spremenljivke.

Aritmetika: osnovni izrazi

Operator	Operacija
`+`	Seštevanje
`-`	Odštevanje
`/*`	Množenje
`/`	Deljenje
`var++`	Povečaj vrednost spremenljivke `var` za 1
`var--`	Zmanjšaj vrednost spremenljivke `var` za 1
`%`	modul (vrni ostanek po deljenju)

Aritmetika: funkcija **expr**

```
1. #!/bin/bash
2. # Osnovna aritmetika z uporabo expr
3.
4. expr 5 + 4 # 9
5.
6. expr "5 + 4" # 5 + 4
7.
8. expr 5+4 # 5+4
9.
10. Expr 11 % 2 # 1
11.
12. expr 5 \* $1 # rezultat (5 * prvi argument)
13.
14. a=$( expr 10 - 3 )
15. echo $a # 7
```

expr deluje podobno kot `let` z izjemo, da namesto shranjevanja rezultata v spremenljivko **izpiše** rezultat.

- Izraza ni potrebno dajati v narekovaje.
- Med posameznimi deli izraza uporabi presledke.
- Ukaz **expr** se običajno uporabi pri zamenjavi ukaza in s tem shrani izhod v spremenljivko.

```
expr item1 operator item2
```

- **Vrstica 4** - Osnovni format. Upoštevaj, da morajo biti med posameznimi deli presledki in da ni narekovajev.
- **Vrstica 6** - Če damo okoli izraza narekovaje, bo izraz izpisan.
- **Vrstica 8** - Če med dele izraza ne damo presledkov, bo izraz izpisan.
- **Vrstica 10** - Prikaz delovanja operatorja **modul**.
- **Vrstica 12** - Nekateri ukazi imajo v bashu poseben pomen, zato moramo pred njih dati \, da odstanimo ta pomen.
- **Vrstica 14** - V tem primeru uporabimo izraz expr pri operaciji zamenjave ukaza in shranimo rezultat v spremenljivko **a**.

Aritmetika: dvojni oklepaji

```
1.  #!/bin/bash
2.  # Osnovna aritmetika z uporabo dvojnih oklepajev
3.
4.  a=$(( 4 + 5 ))
5.  echo $a # 9
6.
7.  a=$((3+5))
8.  echo $a # 8
9.
10. b=$(( a + 3 ))
11. echo $b # 11
12.
13. b=$(( $a + 4 ))
14. echo $b # 12
15.
16. (( b++ ))
17. echo $b # 13
18.
19. (( b += 3 ))
20. echo $b # 16
21.
22. a=$(( 4 * 5 ))
23. echo $a # 20
```

Za shranjevanje rezultata aritmetičnih operacij, lahko uporabimo dvojne oklepaje:

```
$(( izraz ))
```

- **Vrstica 4** – To je osnovni format. Kot lahko vidite, lahko poljubno uporabljamo presledke, da dosežemo večjo berljivost. Pri tem ne potrebujemo narekovajev.
- **Vrstica 7** – Enako deluje tudi brez presledkov.
- **Vrstica 10** – Če v izrazu uporabljamo spemenljivke, lahko spustimo znak $ pred njimi.
- **Vrstica 13** – Lahko pa tudi uporabimo znak $ pred spremenljivkami.
- **Vrstica 16** – Vrednost spremenljivke b povečamo za 1. Pri tem ne potrebujemo znaka $ pred oklepaji.
- **Vrstica 19** – Podobno kot v prejšnjem primeru, tu povečamo vrednost spremenljivke b za 3.
- **Vrstica 22** – Za razliko od ostalih metod, pri tej ne potrebujemo znaka \ pred znakom za množenje (*).

Dolžina spremenljivke

```
1. #!/bin/bash
2. # Dolžina spremenljivke.
3.
4. a='Hello World'
5. echo ${#a} # 11
6.
7. b=4953
8. echo ${#b} # 4
```

Da ugotovimo dolžino spremenljivke (koliko znakov hrani), lahko uporabimo:

```
${#spremenljivka}
```

Stavek IF

```
1. #!/bin/bash
2. # Osnovni stavek IF
3.
4. if [ $1 -gt 100 ]
5. then
6.       echo Hey that\'s a large number.
7.       pwd
8. fi
9.
10. date
```

```
user@bash: ./if_example.sh 15
Mon 6 Nov 17:20:40 2017
user@bash: ./if_example.sh 150
Hey that's a large number.
/home/vida/bin
Mon 6 Nov 17:20:40 2017
user@bash:
```

Osnovni stavek `if` pravi, če je vrednost določenega testa prava (true), potem izvedi podan set ukazov. Če vrednost ni prava (false), potem teh ukazov ne izvedi.

```
if [ <test> ]
then
        <ukazi>
fi
```

Vse med besedama `then` in `fi` (if brano nazaj) bo izvedeno le v primeru, če test (med oglatimi oklepaji) vrne vrednost true.

- **Vrstica 4** - Če je vrednost prvega elementa ukazne vrstice večja od 100.
- **Vrstici 6 in 7** - Se bosta izvedli le v primeru, če test v vrstici 4 vrne vrednost true. Tu je lahko poljubno število ukazov.
- **Vrstica 6** - Obratna poševnica (\ ) pred enojnimi narekovaji (') je potrebna, ker imajo enojni narekovaji poseben pomen, ki ga v tem primeru ne želimo.
- **Vrstica 8** - `fi` označuje konec stavka if. Vsi nadaljnji ukazi bodo izvedeni kot običajno.
- **Vrstica 10** - Ta ukaz bo izveden ne glede na rezultat testa v vrstici 4.

Testni ukazi

Oglati oklepaji([ ]) v stavku `if` predstavljajo referenco na ukaz `test`. To pomeni, da lahko tu uporabimo vse operatorje, s katerimi deluje ukaz test. Nekateri bolj pogosti operatorji so v tabeli.

Operator	opis
! IZRAZ	IZRAZ ima vrednost *false*.
-n NIZ	Dolžina NIZa je večja kot 0.
-z NIZ	Dolžina NIZa je enaka 0 (niz je prazen).
NIZ1 = NIZ2	NIZ1 je enak kot NIZ2
NIZ1 != NIZ2	NIZ1 ni enak kot NIZ2
ŠTEVILO1 -eq ŠTEVILO2	ŠTEVILO1 je numerično enako kot ŠTEVILO2
ŠTEVILO1 -gt ŠTEVILO2	ŠTEVILO1 je numerično večje kot ŠTEVILO2
ŠTEVILO1 -lt ŠTEVILO2	ŠTEVILO1 je numerično manjše kot ŠTEVILO2
-d DOKUMENT	DOKUMENT obstaja in je direktorij.
-e DOKUMENT	DOKUMENT obstaja.
-r DOKUMENT	DOKUMENT obstaja in nad njim so določene pravice branja.
-s DOKUMENT	DOKUMENT obstaja, njegova velikost je večja kot nič (ni prazen).
-w DOKUMENT	DOKUMENT obstaja in nad njim so določene pravice pisanja.
-x DOKUMENT	DOKUMENT obstaja in nad njim so določene pravice izvajanja.

OPOMBE:

- **=** se razlikuje od **-eq**.
  [001 = 1] bo vrnilo rezultat false, saj = primerja nize (ie.posamezne znake), medtem ko -eq primerja numerične vrednosti [001 -eq 1] in bo true.
- Ko rečemo DOKUMENT imamo v mislih pot do dokumenta. Pot je lahko absolutna ali relativna in se lahko nanaša na dokument ali direktorij.
- Ker so [ ] referenca na ukaz **test**, lahko preizkušamo ukaz in njegove nastavitve v ukazni vrstici, da se prepričamo, če pravilno razumemo njegovo delovanje.

Testni ukazi (2)

```
1.  user@bash: test 001 = 1
2.  user@bash: echo $?
3.  1
4.  user@bash: test 001 -eq 1
5.  user@bash: echo $?
6.  0
7.  user@bash: touch myfile
8.  user@bash: test -s myfile
9.  user@bash: echo $?
10. 1
11. user@bash: ls /etc > myfile
12. user@bash: test -s myfile
13. user@bash: echo $?
14. 0
15. user@bash:
```

- **Vrstica 1** - Primerjaj izraz kot niza. Rezultat se ne prikaže na zaslonu, zato moramo preveriti izhodni status ukaza, kar naredimo v naslednji vrstici.
- **Vrstica 2** - Spremenljivka **$?** Hrani izhodni status ukaza, ki je bil nazadnje izveden (v tem primeru je to ukaz test).
  0 pomeni TRUE (uspeh). 1 = FALSE (neuspeh).
- **Vrstica 4** - Tokrat izvajamo numerično primerjavo.
- **Vrstica 7** - Ustvari nov prazen dokument **`myfile`** (predpostavimo, da še ne obstaja).
- **Vrstica 8** - Ali je velikost `myfile` večja od 0?
- **Vrstica 11** - Preusmeri nekaj vsebine v `myfile`
- **Vrstica 12** - Ponovno preveri velikost `myfile`.
  Tokrat je rezultat TRUE.

Vaja

Napiši skripto, ki bo seštela dve števili, ki sta podani kot argumenta ukazni vrstici. Če števili nista podani, prikaži napako in navodila kako se uporablja skripto.

Vgnezdeni stavki IF

```
1.  #!/bin/bash
2.  # Vgnezdeni stavki IF
3.
4.  if [ $1 -gt 100 ]
5.  then
6.      echo Hey that\'s a large number.
7.
8.      if (( $1 % 2 == 0 ))
9.      then
10.         echo And is also an even number.
11.     fi
12. fi
```

V skripti imate lahko toliko stavkov **`if`** kolikor je potrebno. Prav tako je mogoče, da imate stavek `if` znotraj nekega drugega stavka `if`.

- **Vrstica 4** – Izvedi sledeče ukaze, če je vrednost prvega argumenta ukazni vrstici večja od 100.
- **Vrstica 8** – To je variacija stavka `if`. Če želimo preveriti aritmetični izraz, lahko uporabimo dvojne oklepaje.
- **Vrstica 10** – Se izvede le v primeru, da sta vrednosti obeh stavkov `if` pravi (true).

Stavek IF - ELSE

```
1.  #!/bin/bash
2.  # if-else primer
3.  # beri iz dokumenta, če je podan kot
4.  # argument ukazne vrstice, sicer beri
5.  # iz STDIN.
6.
7.  if [ $# -eq 1 ]
8.  then
9.      nl $1
10. else
11.     nl /dev/stdin
12. fi
```

Včasih si želimo izvesti niz ukazov, ko je določena trditev prava (true) in nek drug niz ukazov, če je trditev napačna (false). To lahko naredimo z uporabo mehanizma `else`.

```
if [ <test> ]
then
        <ukazi>
else
        <drugi ukazi>
fi
```

Vaja

Napiši skripto, ki ugotovi ali določen dokument obstaja ali ne. Ime dokumenta je podano kot argument ukazne vrstice. Prav tako preveri, če je podanih dovolj argumentov ukazne vrstice.

Stavek IF – ELIF – ELSE

Primer: Če ste stari 18 let ali več, greste lahko na zabavo.
Če niste dovolj stari, ampak imate pisno soglasje staršev, greste lahko a morate biti doma pred polnočjo.
V nasprotnem primeru ne smete na zabavo.

```
1.  #!/bin/bash
2.  # ukazi elif
3.
4.  if [ $1 -ge 18 ]
5.  then
6.      echo Lahko greste na zabavo.
7.  elif [ $2 == 'yes' ]
8.  then
9.     echo Lahko greste na zabavo a se
       morate vrniti domov pred polnočjo.
10. else Ne smete na zabavo.
12. fi
```

Včasih imamo lahko vrsto pogojev, ki lahko vodijo do različnih ciljev.

```
if [ <test> ]
then
        <ukazi>
elif [ <test> ]
then
        <drugi ukazi>
else
        <drugi ukazi>
fi
```

Lahko imate kolikor si želite ukazov `elif`. Tudi končni ukaz `else` je neobvezen.

Vaja

Napišite skripto, ki bo poiskala najvišje število izmed podanih treh števil. Števila so podana kot argumenti ukazne vrstice.

Izpišite napako, če ni podanih zadostno število argumentov v ukazni vrstici.