Računalniški praktikum 1

Uvod v bash

Vida Groznik

Vir: Ryan‘s tutorials.

Funkcije

```
1. #!/bin/bash
2. # Osnovna funkcija
3.
4. print_something () {
5.     Echo Hello, I am a function
6. }
7.
8. print_something
9. print_something
```

```
user@bash: ./function.sh
Hello, I am a function
Hello, I am a function
user@bash:
```

Funkcije lahko zapišemo v dveh različnih formatih:

```
ime_funkcije () {
    <ukazi>
}
```

ALI

```
function ime_funkcije {
    <ukazi>
}
```

- Pri drugih programskih jezikih je običajno, da se znotraj oklepajev () poda argumente funkciji. V bashu tega **ne** počnemo.
- Definicija funkcije (torej dejanska funkcija) mora biti v skripti navedena **preden** jo dejansko uporabimo.

Funkcije – podajanje argumentov

```
1. #!/bin/bash
2. # Podajanje argumentov funkciji
3.
4. print_something () {
5.     Echo Hello, $1
6. }
7.
8. print_something Mark
9. print_something Nina
```

```
user@bash: ./function_arguments.sh
Hello, Mark
Hello, Nina
user@bash:
```

Podajanje argumentov funkciji poteka podobno, kot podajanje argumentov ukazne vrstice skripti.

Argumente podamo takoj za klicem funkcije. Znotraj same funkcije so dostopni kot spremenljivke **$1, $2, itd**.

Funkcije – vračanje vrednosti

```
1.  #!/bin/bash
2.  # Določitev izhodnega statusa funkcije
3.
4.  print_something () {
5.      Echo Hello, $1
6.      return 5
7.  }
8.  print_something Mark
9.  print_something Nina
10. echo Prejsnja funkcija ima izhodno vrednost $?
```

Funkcije v bashu nam ne omogočajo, da vrnemo podatke (return) nazaj na klicno lokacijo.

Omogočajo nam zgolj nastavitev izhodnega statusa na podoben način, kot to deluje pri programih oz. posameznih ukazih, kjer se le-ti zaključijo s statusom, ki nam pove, ali se je program/ukaz uspešno izvedel ali ne.

Za določitev izhodnega statusa uporabimo besedo **`return`**.

```
user@bash: ./return_status.sh
Hello, Mark
Hello, Nina
Prejsnja funkcija ima izhodno
vrednost 5
user@bash:
```

Če elite iz funkcije vrniti številko (npr. rezultat izračuna), lahko za to poskusite uporabiti izhodni status.
(Sicer to ni namenjeno temu, ampak bo delovalo.)

Funkcije – vračanje vrednosti (2)

```
1.  #!/bin/bash
2.  # Določitev izhodnega statusa funkcije
3.
4.  lines_in_file () {
5.      cat $1 | wc -l
6.  }
7.
8.  num_lines=$( lines_in_file $1 )
9.
10. echo Datoteka $1 ima $num_lines vrstic.
```

```
user@bash: ./return2.sh myfile.txt
Datoteka myfile.txt ima 5 vrstic.
user@bash:
```

En način, kako lahko rešimo problem vračanja vrednosti iz funkcije je upraba koncepta *Zamenjave ukaza*.

Vrstica 5 – Ukaz bo izpisal število vrstic v dokumentu, ki je podan kot argument funkciji (naslovimo ga z `$1`).

Vrstica 8 – Uporabimo koncept zamenjave ukaza in v spremenljivko `num_lines` shranimo vrednost, ki bi bila sicer izpisana.

Doseg spremenljivk

```
1.  #!/bin/bash
2.  # Doseg spremenljivk
3.
4.  var_change () {
5.      Local var1='local 1'
6.      echo Znotraj funkcije: var1 je $var1
7.      echo Znotraj funkcije: var2 je $var2
8.      var1='ponovno spremenjena'
9.      var2='2 ponovno spremenjeno'
10. }
11.
12. var1='global 1'
13. var2='global 2'
14.
15. echo Pred klicem funkcije: var1 je $var1
16. echo Pred klicem funkcije: var2 je $var2
17.
18. var_change
19.
20. echo Po klicu funkcije: var1 je $var1
21. echo Po klicu funkcije: var2 je $var2
```

Doseg se nanaša na to **kateri deli skripte lahko vidijo katere spremenljivke.**

Spremenljivke so **privzeto globalne**. To pomeni, da je **vidna povsod** v skripti.

Kreiramo lahko tudi **lokalne** spremenljivke. Ko kreiramo lokalno spremenljivko znotraj funkcije, je **vidna zgolj znotraj** te **funkcije**.

To naredimo z uporabo besede **`local`** pred prvo nastavitvijo vrednosti spremenljivki.

```
local spremenljivka=<vrednost>
```

```
user@bash: ./local_variables.sh
Pred klicem funkcije: var1 je global 1
Pred klicem funkcije: var2 je global 2
Znotraj funkcije: var1 je local 1
Znotraj funkcije: var2 je global 2
Po klicu funkcije: var1 je global 1
Po klicu funkcije: var2 je 2 ponovno
spremenjeno
user@bash:
```

Nadomestitev ukazov

```
1.  #!/bin/bash
2.  # Izdelaj wrapper okoli ukaza ls
3.
4.  ls () {
5.      command ls -lh
6.  }
7.
8. ls
```

Bash omogoča, da funkcijo poimenujemo z istim imenom, kot ga ima ukaz, ki bi ga sicer uporabili v ukazni vrstici.

To nam omogoča, da izdelamo wrapper.

Npr. vsakič ko zaženemo ukaz **`ls`** znotraj naše skripte, pravzaprav želimo ukaz **`ls -lh`**.

Vrstica 5 – Ko imamo funkcijo, ki ima enako ime kot se imenuje ukaz, moramo uporabiti besedo **`command`** pred imenom ukaza, ko želimo uporabiti dejanski ukaz in ne funkcije (ki ima sicer prednost pred ukazom).

Naloga 1

Napiši funkcijo, ki izračuna ploščino pravokotnika.

Pri klicu funkcije podate dolžini obeh stranic.

Primer:

ploscina 10 20

Rezultat: 200

Naloga 2

Napišite skripto, ki kot argumente v ukazni vrstici dobi največ 9 števil.

a) Skripta naj vsebuje različne funkcije, ki vam izračunajo:
   - Vsoto podanih števil
   - Razliko podanih števil
   - Zmnožek podanih števil

   Program naj izpiše vse dobljene rezultate.

b) Program vpaša uporabnika, katero računsko operacijo želi izvesti in na podlagi vnosa uporabnika naredi izračun ter ga izpiše na seznam. (Sprogramiraj verzijo, da lahko uporabnik večkrat vnese različne operacije znotraj enkratnega klica skripte.)

Naloga 3

Napiši skripto v bashu za implementacijo izjave getopts.

Vaša skripta mora razumeti naslednje argumente ukazne vrstice (skripta ima ime options.sh):

```
./options.sh -c –d -u
```

Funkcije opcij so:

`-c` počisti zaslon

`-d` pokaži seznam dokumentov v trenutnem direktoriju

`-u` izpiši podrobnosti uporabnika, ki poganja program

- Prikaži pomoč, kako se požene skripta, če ni podanega nomebega argumenta ali če argument ni znan.

Naloga 4

Napiši skripto, ki prejme ime dokumenta in 3 besede. Z uporabo ukaza **grep** naj skripta v dokumentu poišče vse 3 besede. Poleg tega naj izpiše število pojavitev posamezne besede in številke vrstic, v katerih se pojavlja beseda.

- Skripta naj ne izpiše ničesar drugega na zaslon, kot to, kar zahteva naloga.
- Če skripta ne more prebrati dokumenta, se mora skripta zaključiti z izhodno vrednostjo 1, sicer z vrednostjo 0. (poglej „help exit“, če ne poznate delovanja ukaza exit.)