

Operacijski sustavi (OS)

Nositelj: doc. dr. sc. Ivan Lorencin

Asistent: Luka Blašković, mag. inf.

Ustanova: Sveučilište Jurja Dobrile u Puli, Fakultet informatike u Puli



Fakultet informatike u Puli

(3) Bash skriptiranje

#3

OS

Osim interaktivne upotrebe, bash se široko primjenjuje za izradu skripti – jednostavnih programa kojima se automatiziraju različiti zadaci. Takve se skripte često koriste u kontekstu računalnog oblaka, DevOps praksi te za upravljanje udaljenim poslužiteljima, primjerice prilikom izrade sigurnosnih kopija, nadzora sustava, postavljanja aplikacija ili provedbe CI/CD procesa. Iako jednostavan, bash i dalje zauzima važno mjesto u automatizaciji, kako u suvremenim infrastrukturnim rješenjima poput kontejnerskih platformi i hibridnih cloud sustava, tako i u povijesnom kontekstu, gdje je predstavljao ključni alat za skriptiranje u tradicionalnim Unix okruženjima.

 Posljednje ažurirano: 9.4.2025.

Sadržaj

- [Operacijski sustavi \(OS\)](#)
- [\(3\) Bash skriptiranje](#)
 - [Sadržaj](#)
- [1. Uvod](#)
- [2. Priprema skripte](#)
 - [2.1 CLI uređivači teksta](#)
 - [2.1.1 nano uređivač](#)
 - [2.1.2 vim uređivač](#)
- [3. Programski koncepti u bashu](#)
 - [3.1 Varijable i argumenti](#)
 - [3.1.1 Supstitucija varijabli](#)
 - [3.1.2 Argumenti](#)

- [3.2 Uvjetni izrazi i operatori](#)
 - [3.2.1 Tablica operatora za usporedbu brojeva](#)
 - [3.2.2 Tablica operatora za usporedbu stringova](#)
 - [3.2.3 Tablica operatora za provjere datoteka](#)
- [3.3 Kombiniranje više naredbi](#)
- [Zadatak 1: Provjera datoteke prema apsolutnoj putanji](#)
- [3.4 Petlje \(iteracije\)](#)
 - [3.4.1 `for` petlje](#)
 - [Iteracija kroz niz](#)
 - [C-style](#)
 - [Iteracija kroz datoteke](#)
 - [3.4.2 `while` i `until` petlje](#)
 - [Aritmetika](#)
- [3.5 Funkcije](#)
- [3.6 Kratki pregled](#)
- [Zadaci za Vježbu 3](#)

1. Uvod

Do sad smo naučili osnovne koncepte **interaktivnog rada** u bash shellu, odnosno navigaciju datotečnim sustavom, korištenje osnovnih naredbi za rad s datotekama i direktorijima te koncept zastavica.

Drugi način korištenja basha je **skriptiranje** gdje skripte predstavljaju tekstualne datoteke u koje pišemo bash naredbe koje želimo izvršiti. Ipak, kako se radi o programskom jeziku, možemo koristiti i ostale programske koncepte, poput varijabli, uvjetnih izraza, petlji, funkcija itd.

Zašto učimo bash skriptiranje?

Vjerojatno se pitate zašto učiti bash skriptiranje kada postoje puno moderniji programski jezici pomoću kojih možemo učinkovitije razvijati složene projekte. Navest ćemo nekoliko razloga:

1. **Automatizacija zadataka:** bash skripte idealne su za programiranje automatskog *backupa*, sustavsku administraciju, *deployment*, analizu logova i ostale rutinske zadatke koje želimo obavljati automatski.
 - Na primjer, možemo napisati skriptu koja će automatski preuzeti sigurnosnu kopiju našeg web poslužitelja svaki dan u određeno vrijeme.
2. **Jednostavan i brz:** bash je jednostavan za korištenje i već prisutan na gotovo svim Unix/Linux sustavima - nema potrebe za instalacijom.
3. **Idealan za "glue code":** bash je izvrstan za povezivanje različitih alata i skripti, neovisno o programskom jeziku, tehnologiji ili platformi. Možemo ga zamisliti kao ljepilo koje drži sve dijelove našeg sustava zajedno jednom kad ga *deployamo* u produkcijsko okruženje.

4. **DevOps i CI/CD:** alati često koriste bash skripte za automatizaciju procesa izgradnje, testiranja i isporuke softvera ([CI/CD](#)). Razumijevanje basha poboljšat će vašu produktivnost i omogućiti vam da bolje razumijevanje kako aplikacije rade na nižoj razini apstrakcije.

✗ Kada nećemo pisati bash skriptu

1. Kada logika postane dovoljno kompleksna i počinje se javljati potreba za naprednijim strukturama podataka, bibliotekama... → tada je bolje koristiti nešto modernije: Python, Go, JavaScript/TypeScript, Java, C#...
2. Za izradu aplikacija namijenjenih krajnjim korisnicima.
3. Kada trebamo raditi s velikim količinama podataka, pisati složene algoritme i sl.



U ovoj skripti studenti će se upoznati s osnovama pisanja bash skripti, s ciljem lakšeg obavljanja sistemskih operacija u budućim kolegijima, osobito onima vezanim uz DevOps praksu. Fokus će biti na osnovnim vještinama koje omogućuju automatizaciju rutinskih zadataka, čime će studenti steći čvrstu osnovu za razumijevanje i primjenu skriptiranja u kontekstu administracije sustava.

2. Priprema skripte

Bash skripte definiramo kraticom `.sh` i one su jednostavne tekstualne datoteke koje sadrže niz bash naredbi koje se izvršavaju redom.

Stvorit ćemo novu datoteku `hello.sh` u radnom direktoriju:

```
→ touch hello.sh
```

Osim s naredbom `touch`, zadnji put smo vidjeli da datoteku možemo stvoriti i pomoću naredbe `echo` i odmah ju popuniti zadanim sadržajem:

```
→ echo "Pozdrav iz bash skripte" > hello.sh
```

Sadržaj datoteke možemo ispisati pomoću naredbe `cat`:

```
→ cat hello.sh
```

Rezultat:

```
Pozdrav iz bash skripte
```

Ipak, skripta iznad nije izvršna, tj. ne možemo je pokrenuti kao program budući da nema niti jedne ispravne naredbe.

- za ispis u terminal koristimo naredbu `echo` pa ćemo nju dodati u sadržaj datoteke

```
→ echo "echo 'Pozdrav iz bash skripte'" > hello.sh
```

Uočite sljedeće:

- Prvi `echo` koristimo za pisanje u datoteku (i stvaranje nove ako ne postoji)
- Drugi `echo` upisujemo u datoteku, ali ga ne izvršavamo (još)

Skripta je sada ispravna budući da sadrži bash naredbu, možemo ju pokrenuti naredbom `bash`:

```
→ bash hello.sh
```

Bash skripta će se sada izvršiti:

```
Pozdrav iz bash skripte
```

Uočite razlike koje smo napravili:

1. Upisali smo obični tekstualni sadržaj u datoteku `hello.sh` koji nije izvršiv (jer nije ispravna naredba - tj. nema `echo`) i ispisali ga pomoću `cat` naredbe

2. Upisali smo ispravnu bash naredbu u datoteku `hello.sh` i pokrenuli je pomoću `bash` naredbe

[Shebang](#) je posebna oznaka (`#!`) na Unix sustavima koja se koristi na početku skripte kako bi se odredilo koji interpreter treba koristiti za izvršavanje same skripte.

U slučaju basha, koristimo `#!/bin/bash`

- Ova oznaka omogućuje operacijskom sustavu da prepozna da je skripta **napisana u bash jeziku** i da ju **izvrši pomoću bash interpretera**.

Oznaku dodajemo na **početak skripte**:

```
#!/bin/bash
echo "Pozdrav iz bash skripte"
```

- za sada možete dodati ručno (npr. GUI editor), a nastavku ćemo naučiti kako to učiniti pomoću CLI uređivača teksta

Nakon definiranja skripte, moramo dodati i **dozvolu za izvršavanje** skripte pomoću `chmod` naredbe:

`chmod` je skraćenica za "change mode" i koristi se za promjenu dozvola datoteka i direktorija na Unix sustavima

Sintaksa:

```
→ chmod [FLAGS] <mode> <putanja_do_datoteke>
```

`+x` označava da dodajemo dozvolu za izvršavanje (`+` je operator koji dodaje dozvolu, a `x` označava dozvolu za izvršavanje)

- ovo nije zastavica, već `<mode>` argument

```
→ chmod +x hello.sh
```

To je to! Sada ju možemo pokrenuti jednostavno navodeći relativnu (ili apsolutnu) putanju - **bez posebne naredbe**:

```
./hello.sh
```

```
# Oprez: ne možemo napisati samo "hello.sh" jer će bash to interpretirati kao naredbu.
Umjesto toga navodimo putanju, počevši od trenutnog direktorija: "./"
```


Rezultat:

```
Pozdrav iz bash skripte
```

2.1 CLI uređivači teksta

CLI uređivači teksta (*eng. CLI text editors*) su programski alati koji omogućuju izravno uređivanje tekstualnih datoteka unutar terminalskog sučelja, bez potrebe za grafičkim okruženjem.

CLI uređivači teksta često se koriste u radu s udaljenim poslužiteljima, razvoju softvera, administraciji sustava, radu s virtualnim strojevima, gdje je rad u terminalu učinkovitiji ili pak **jedina dostupna opcija**.

 **Hint:** Iako se novim korisnicima rad u ovim uređivačima u početku može činiti kompliciranim, sporim i zahtjevnim, redovitom vježbom, prilagodbom postavki i upoznavanjem s tipkovničkim prečacima, oni omogućuju znatno učinkovitiju obradu teksta.

Postoje mnogi CLI uređivači, a neki od poznatijih su:

- `nano` - jednostavan uređivač teksta, idealan za početnike
- `vim` - napredniji uređivač teksta koji nudi mnoge mogućnosti, jako prilagodljiv, ali teži za naučiti i usavršiti
- `emacs` - familija uređivača teksta koji su vrlo moćni i prilagodljivi, razvijani još 70-ih godina
- `neovim` - modernija verzija `vim` uređivača koja nudi poboljšanja i dodatne mogućnosti - danas dosta popularan u programerskoj zajednici
- `micro` - moderan uređivač teksta koji je jednostavan za korištenje i nudi mnoge mogućnosti *out-of-the-box*, poput sintaktičkog isticanja, automatskog dovršavanja, multijezične podrške i sl.

Preporuka je da koristite `nano` ili `vim` uređivače, a ako ste već upoznati s nekim drugim uređivačem, slobodno ga koristite.

Provjerite imate li instalirane uređivače:

```
→ nano --version
→ vim --version
```

Ako nisu instalirani, dobit ćete grešku.

- Ako koristite **WSL** ili **Git Bash**, oba uređivača su vjerojatno već instalirana. Ako nisu, preporuka je da ih instalirate pomoću `apt` alata

```
→ sudo apt install nano
→ sudo apt install vim
```

- Ako koristite **macOS**, uređivači oba su vjerojatno već instalirana. Ako nisu, preporuka je da ih instalirate pomoću [Homebrew](#) alata

```
→ brew install nano
→ brew install vim
```

- Ako koristite **Linux**, ovisno o distribuciji možete imati jedan ili drugi već instaliran. Ako nisu, preporuka je da ih instalirate pomoću `apt` alata kao što već prikazano iznad

Jednom kad ste instalirali uređivače, možete ih koristiti za uređivanje datoteka iz terminala 😎

2.1.1 nano uređivač

Nano ([GNU Nano](https://www.nano-editor.org/)) je jednostavan uređivač teksta koji je idealan za početnike. Ima jednostavno sučelje i jednostavan je za korištenje. Podržava sve osnovne funkcije uređivanja teksta. Razvijen je 1999. godine kao slobodna zamjena za [Pico](#), uređivač koji je bio dio e-mail klijenta [Pine](#). Danas se koristi zbog svoje dostupnosti u gotovo svim Linux distribucijama i zbog jednostavnosti u radu s konfiguracijskim i skriptnim datotekama u terminalu.

```
      :~:
iLE88Dj. .:jd88888Dj:
.LGite888D.f8GjjL8888E;
iE .8888Et. .G8888.
;i  E888. .8888.
   D888. .8888.
   D888. .8888.
   D888. .8888.
   W88W. .8888.
   W88W. .8888.
   DGD: .8888.
      :W888:
      :8888:
      :E888i
      :tW88D
```



Logotip nano uređivača teksta, web: <https://www.nano-editor.org/>

Sintaksa:

```
→ nano <putanja_do_datoteke>
```

- gdje je `<putanja_do_datoteke>` relativna ili apsolutna putanja do datoteke koju želite otvoriti
- ako datoteka na putanji ne postoji, nano će je stvoriti**

Na primjer, da otvorimo datoteku `hello.sh` u nano uređivaču, jednostavno upišemo:

```
→ nano hello.sh
```



Izgled nano uređivača. Uočite osnovne naredbe prikazane na dnu sučelja

Unutar nano uređivača možemo uređivati datoteku navigacijom pomoću tipkovnice, a jednom kad završimo s uređivanjem i želimo pohraniti promjene, koristimo naredbe:


- CTRL + o** - spremi datoteku (Write Out)
- CTRL + x** - izađi iz uređivača (Exit)

Tablica korisnih naredbi `nano` uređivača:

Naredba	Objašnjenje
<code>Ctrl + O</code>	Spremi datoteku (Write Out)
<code>Ctrl + X</code>	Izađi iz uređivača. Ako postoje nespremljene promjene, bit ćete upitani želite li ih spremiti.
<code>Ctrl + K</code>	Izreži (izbriši) trenutni redak i spremi ga u međuspremnik.
<code>Ctrl + U</code>	Zalijepi (ubaci) prethodno izrezani sadržaj iz međuspremnika.
<code>Ctrl + W</code>	Pretraži tekst. Unosi se ključna riječ koju uređivač zatim traži unutar datoteke.
<code>Ctrl + C</code>	Prekida izvršavanje trenutne naredbe (Nije kopiranje!)
<code>Ctrl + A</code>	Pomakni kursor na početak retka
<code>Ctrl + E</code>	Pomakni kursor na kraj retka

Kako biste brže navigirali kroz datoteku, možete koristiti `CTRL + SPACE` za preskakanje riječi (ovisno o verziji može biti i `ALT + →` odnosno `ALT + ←`). Osim toga, možete koristiti i `CTRL + Y` za pomicanje prema gore, odnosno `CTRL + V` za pomicanje prema dolje.

`nano` cheat sheet: [dostupan ovdje](#)

 **Napomena:** Ako koristite `nano` uređivač unutar `WSL` ili `Git Bash`, možda ćete primijetiti da se neki od tipkovničkih prečaca razlikuju od onih u drugim verzijama `nano` uređivača. To je zbog razlika u terminalima i njihovim postavkama. Ako naiđete na probleme, provjerite dokumentaciju za svoj terminal ili pokušajte koristiti drugi uređivač. Možete upotrijebiti naredbu `CTRL + G` koja otvara *Manual* u kojem možete pronaći sve naredbe i njihove definicije.

2.1.2 vim uređivač

`vim` je napredniji uređivač teksta koji nudi mnoge mogućnosti i prilagodbe. Iako je moćan, može biti teži za naučiti od `nano` uređivača zbog [svojih načina rada](#) i načina navigacije. Razvijen je 1991. godine kao poboljšana verzija starijeg uređivača [vi](#), a autor mu je Bram Moolenaar. Danas se koristi među programerima i naprednim korisnicima zbog svoje učinkovitosti, brzine i širokih mogućnosti i nadogradnje baznih funkcionalnosti.



 Logotip `vim` uređivača teksta, web: <https://www.vim.org/>

Sintaksa:

```
→ vim <putanja_do_datoteke>
```

- gdje je `<putanja_do_datoteke>` relativna ili apsolutna putanja do datoteke koju želite otvoriti u uređivaču
- ako datoteka na putanji ne postoji, `vim` će stvoriti novu**


Na primjer, da otvorimo datoteku `hello.sh` u `vim` uređivaču, jednostavno upišemo:

```
→ vim hello.sh
```

```
# ili možemo stvoriti novu datoteku, koristeći apsolutnu putanju
```

```
→ vim ~/Documents/nova_datoteka.js
```



 Izgled `vim` uređivača. Uočite da je sučelje dosta jednostavnije od `nano` sučelja. Ipak, `vim` je kompleksniji za korištenje.

Načini rada:

`vim` uređivač ima nekoliko načina rada, a najvažniji su:

- **Normal** - zadani način rada u kojem **možete navigirati kroz tekst i koristiti različite naredbe**
 - zadani način rada u `vim` uređivaču, kako bi se vratili u njega pritisnemo tipku `ESC`
- **Insert** - način rada u kojem možete **unositi znakove**.
 - u ovaj način rada ulazimo pritiskom na `i` tipku
 - na dnu sučelja ćete vidjeti oznaku `-- INSERT --`
- **Visual** - način rada u kojem možete **označavati tekst** tipkovnicom.
 - u ovaj način rada ulazimo pritiskom na `v` tipku
 - na dnu sučelja ćete vidjeti oznaku `-- VISUAL --`
- **Command** - način rada u kojem možete **unositi različite naredbe**.
 - u ovaj način rada ulazimo pritiskom na `:` tipku
 - na dnu sučelja ćete vidjeti oznaku `:`

Primjer rada u Vimu: Želimo urediti datoteku `hello.sh` i dodati `shebang` oznaku na početak datoteke.

1. Upisujemo naredbu `vim hello.sh` i otvaramo datoteku
2. Otvaramo `Insert` način rada pritiskom na `i` tipku
3. Upisujemo `#!/bin/bash`
4. Pritiskom na `ESC` tipku vraćamo se u `Normal` način rada
5. Otvaramo `Command` način rada pritiskom na `:` tipku
6. Unosimo naredbu `wq` i pritisnemo `ENTER` tipku kako bi spremili promjene i izašli iz uređivača

Uobičajene naredbe u **Normal** načinu rada


- **Normal** način rada je zadani način rada u `vim` uređivaču i omogućuje nam navigaciju kroz tekst i korištenje različitih naredbi.

Naredbe u Normal	Opis
i	Ulaz u Insert način rada neposredno prije trenutnog znaka
a	Ulaz u Insert način rada neposredno nakon trenutnog znaka
o	Dodaje novi red ispod trenutnog i prelazi u Insert način rada
x	Briše znak na kojem se nalazi kursor
dd	Briše (delete) cijeli trenutni red i sprema ga u međuspremnik
yy	Kopira (yank) cijeli trenutni red u međuspremnik
p	Zalijepi (paste) prethodno kopirani ili izrezani sadržaj neposredno nakon kursora
/tekst	Pretražuje niz "tekst" u datoteci; n za sljedeći rezultat, N za prethodni rezultat
u	Poništava posljednju izvršenu akciju (undo)
CTRL + r	Ponavlja prethodno poništenu akciju (redo)

Za izlazak iz **vim** uređivača moramo ući u **Command** način rada:

- Command** način rada otvaramo pritiskom unosom dvotočja :

Naredbe u Command	Opis
:w	Spremi trenutnu datoteku
:q	Izađi iz uređivača (ako nema nespremljenih promjena)
:wq	Spremi datoteku i izađi
:q!	Izađi bez spremanja promjena
:set nu	Prikaži brojeve linija (<i>eng. line numbers</i>) u uređivaču (korisno)

 **Napomena:** Ako koristite **vim** uređivač unutar **WSL** ili **Git Bash**, možda ćete primijetiti da se neki od tipkovničkih prečaca razlikuje od ovih u tablicama. Ako naiđete na probleme, provjerite dokumentaciju za svoj terminal ili pokušajte koristiti drugi uređivač ako ne ide.

U vimu će naredba **:help** otvoriti *Manual* u kojem možete pronaći naredbe i njihove definicije.

vim cheat sheet: [dostupan ovdje](#)

Odabir uređivača ovisi o vašim potrebama i preferencijama. Početnicima se svakako preporučuje **nano** uređivač zbog svoje jednostavnosti, dok napredniji korisnici mogu preferirati **vim** zbog složenijih funkcionalnosti i raznih mogućnosti prilagodbe kroz [uređivanje konfiguracijske datoteke](#). Oba uređivača su vrlo moćna i često se koriste u CLI okruženju, ali imaju različite karakteristike i ciljane korisnike.

Za potrebe ovog kolegija možete odabrati bilo koji uređivač od navedenih (ili neki drugi), ali mora biti CLI uređivač!

3. Programski koncepti u bashu

Jednom kad smo naučili kako pokrenuti i uređivati bash skripte, posvetit ćemo se osnovnim programskim konceptima koji će nam pomoći u pisanju skripti za obavljanje složenijih zadataka. Zapamtite da moramo u svaku skriptu dodati `shebang` oznaku: `#!/bin/bash`, a zatim i dozvolu za izvršavanje pomoću `chmod +x` naredbe.


Ako se prisjetite, zastavica `-l` naredbe `ls` ispisuje detaljne informacije o datoteci, uključujući dozvole.

Napravit ćemo novu skriptu `main.sh`:

```
→ touch main.sh
```

Otvorite je u uređivaču po izboru i dodajte sljedeći sadržaj:

```
# main.sh
#!/bin/bash
echo "Pozdrav iz main skripte"
```

 **Napomena:** Oznaka `# main.sh` na početku dodana je isključivo kao komentar da lakše uočite koju skriptu u konkretnom primjeru uređujemo.

Provjerite dozvole datoteke:

```
→ ls -l
```

Rezultat nam pokazuje detaljne informacije o datoteci `main.sh`, fokusiramo se na dozvole (prvi stupac):

```
-rw-r--r--  1 lukablaskovic  staff   43 Apr  2 12:51 main.sh
```

Uočite da datoteka `main.sh` nema dozvolu za izvršavanje (`x`) - **dodat ćemo ju naredbom** `chmod`:

```
→ chmod +x main.sh
```

Ponovnom provjerom možemo vidjeti da je dozvola za izvršavanje dodana:

```
-rwxr-xr-x  1 lukablaskovic  staff   43 Apr  2 12:51 main.sh
```

Sada možemo pokrenuti skriptu:

```
Pozdrav iz main skripte
```

Ekvivalentno, ako bismo htjeli **izbrisati dozvolu** za izvršavanje, koristimo `chmod -x` (`-` je operator koji uklanja, a `x` označava dozvolu za izvršavanje):

```
bash: ./main.sh: Permission denied
```

3.1 Varijable i argumenti

Varijable definiramo bez ključnih riječi i tipizacije, tj. bez `var`, `let`, `const`, `int` i sl. - samo im dodijelimo vrijednost:

Sintaksa:

```
naziv_varijable=vrijednost # bez razmaka!
```

Primjer:

```
# main.sh
#!/bin/bash
ime="Marko"
prezime=Marković # može i bez navodnika
godina_rodenja=1999
```


Varijable ispisujemo naredbom `echo` i koristimo `$` znak ispred varijable (**obavezno je korištenje dvostrukih navodnika ako kombiniramo varijable i tekst**):

```
# main.sh
echo "Pozdrav, $ime $prezime, rođen $godina_rodenja" # ispisuje Pozdrav, Marko Marković, rođen 1999

echo 'Pozdrav, $ime $prezime, rođen $godina_rodenja' # ispisuje Pozdrav, $ime $prezime, rođen $godina_rodenja
```

Zapamtite sljedeće:

- Varijable **ne smiju sadržavati razmake**, tj. ne možemo pisati `ime = Sanja` već samo `ime=Sanja`
- Varijable **ne smiju početi s brojem**, tj. ne možemo pisati `1ime=Marko`
- Varijable **ne smiju sadržavati posebne znakove, osim `_`**, tj. ne možemo pisati `ime@=Marko`, ali možemo `ime_1=Marko`
- Varijable su **osjetljive na velika i mala slova**, tj. `ime` i `IME` predstavljaju dvije različite varijable
- Vrijednosti varijabli **nije potrebno definirati s navodnicima, zadani tip podataka je string**, dakle možemo pisati `ime=Petar` ili `ime="Petar"` → ekvivalentno je

 **Napomena:** bash ne podržava tipizaciju varijabli, sve se pohranjuje kao znakovni niz (string), ali ovisno o kontekstu može interpretirati varijable kao brojeve, datume i sl.

3.1.1 Supstitucija varijabli

U bashu, **rezultate izvođenja naredbi možemo spremiti u varijable** pomoću tzv. supstitucije varijabli. To nam omogućuje da pohranimo izlaz (rezultat) izvođenja neke naredbe u varijablu i kasnije ga koristimo u skripti.

Sintaksa:

```
naziv_varijable=$(bash_naredba)
```

Na primjer, nalazimo se u direktoriju `/vjezba` koji sadrži datoteke `main.sh` i `test.sh`. Želimo spremiti naziv trenutnog radnog direktorija u varijablu `trenutni_direktorij`:

```
# main.sh
trenutni_direktorij=$(pwd)
echo "Trenutni direktorij je: $trenutni_direktorij"
```

Rezultat:

```
Trenutni direktorij je: /Users/lukablaskovic/Github/FIPU-OS/OS3 - Bash skriptiranje/vjezba
```

Možemo pohraniti sadržaj radnog direktorija u varijablu `sadrzaj_vjezba`:

```
# main.sh
trenutni_direktorij=$(pwd) # u varijablu "trenutni_direktorij" spremamo rezultat naredbe "pwd"
sadrzaj_vjezba=$(ls) # u varijablu "sadrzaj_vjezba" spremamo rezultat naredbe "ls"
echo "Trenutni direktorij je: $trenutni_direktorij"
echo "Sadržaj direktorija: $sadrzaj_vjezba"
```

Rezultat:

```
Trenutni radni direktorij je: /Users/lukablaskovic/Github/FIPU-OS/OS3 - Bash
skriptiranje/vjezba
Sadržaj direktorija: main.sh
test.sh
```

Moguće je koristiti i zastavice u supstituciji varijabli, npr. `ls -l`:

```
# main.sh
detaljni_sadrzaj=$(ls -l)
echo "Detaljni sadržaj direktorija: $detaljni_sadrzaj"
```

Rezultat:

```
Detaljni sadržaj direktorija: total 8
-rwxr-xr-x  1 lukablaskovic  staff   245 Apr  2 13:35 main.sh
-rw-r--r--  1 lukablaskovic  staff     0 Apr  2 13:31 test.sh
```

3.1.2 Argumenti

U bash skriptama možemo čitati argumente koji se prosleđuju prilikom pokretanja skripte.

Argumente označavamo posebnim varijablama: `$1`, `$2`, `$3`, ... do `$n`, gdje `$1` predstavlja prvi argument, `$2` drugi argument, itd. do `$n` koji predstavlja n-ti argument.

Zamislimo da je naša **skripta ustvari bash naredba**, a njene argumente navodimo nakon naziva naredbe (skripte).

Sintaksa:

```
→ ./main.sh <vrijednost_arg_1> <vrijednost_arg_2>
```

Primjer: pozivanje skripte s argumentima `FIPU` i `Pula`.

```
→ ./main.sh FIPU Pula
```

Primjer: čitanje i ispisivanje argumenata unutar skripte.

```
# main.sh
#!/bin/bash
echo "Prvi argument je: $1"
echo "Drugi argument je: $2"
```

Rezultat:

```
Prvi argument je: FIPU
Drugi argument je: Pula
```

Možemo koristiti i posebnu oznaku `$@` koja predstavlja **sve argumente proslijeđene skripti**:

```
# main.sh
#!/bin/bash

echo "Svi argumenti su: $@"
```

Primjer pozivanja:

```
→ ./main.sh arg_1 2 treci_argument 4_argument
```

Rezultat:

```
Svi argumenti su: arg_1 2 treci_argument 4_argument
```

Oznakom `$#` možemo dobiti **broj proslijeđenih argumenata**:

```
# main.sh
#!/bin/bash
echo "Broj proslijeđenih argumenata je: $#"
```

Primjer pozivanja:

```
→ ./main.sh arg_1 2 treci_argument 4_argument
```

Rezultat:

```
Broj proslijeđenih argumenata je: 4
```

Prisjetite se naše varijable `$0`. U interaktivnom načinu rada smo rekli da će ispisati **naziv aktivnog shella**:

```
→ echo $0

# ispisuje: bash
```

Međutim, ako ju koristimo unutar skripte, ispisat će **naziv bash skripte** koju smo pokrenuli:

```
# main.sh
#!/bin/bash
echo "Naziv skripte je: $0"

# ispisuje: Naziv skripte je: ./main.sh
```

3.2 Uvjetni izrazi i operatori

Kao i drugim programskim jezicima, uvjetni izrazi (*eng. conditional expressions*) koriste se za donošenje logičkih odluka unutar različitih blokova koda na temelju evaluiranih izraza.

Uvjetni izraz `if` s jednostrukim uglatim zagradama:

Sintaksa:

```
if [ uvjet ]; then
    # blok koda koji se izvršava ako je uvjet točan
else
    # blok koda koji se izvršava ako je uvjet netočan
fi # zatvara if blok
```

Ako imamo treći uvjet, koristimo `elif` oznaku (kao u Pythonu):


```

if [ uvjet ]; then
    # blok koda koji se izvršava ako je uvjet točan
elif [ uvjet_2 ]; then
    # blok koda koji se izvršava ako je uvjet_2 točan
else
    # blok koda koji se izvršava ako je uvjet netočan
fi # zatvara if blok

```

Zapamtite sljedeće:

- Uvjeti se definiraju unutar uglatih zagrada `[]` (ili ugniježđenih `[[]]` - u nastavku), a **između njih se nalaze uvjetni izrazi**
- `then` oznaka označava početak bloka koda koji se izvršava ako je uvjet točan
- **Postoji razmak** između uvjeta i uglatih zagrada, kao i između uglatih zagrada i `then` oznake. **Bez razmaka, bash će javiti grešku!**

Primjer: Provjeravamo je li varijabla `grad` jednaka stringu `"Zagreb"`.

```

# main.sh
#!/bin/bash

grad="Zagreb"
if [ "$grad" = "Zagreb" ]; then # uočite razmake između uglatih zagrada i uvjeta i navodne
znakove oko varijable
    echo "Grad je Zagreb"
else
    echo "Grad nije Zagreb"
fi

```

Primjer pogrešnog `if` bloka:

```

# main.sh
#!/bin/bash

grad="Zagreb"
if [ "$grad" = "Zagreb"]; then # Greška! Nismo dodali razmak između uglatih zagrada i
uvjeta
    echo "Grad je Zagreb"
else
    echo "Grad nije Zagreb"
fi

```

Primjer: Kombinirat ćemo uvjetne izraze i argumente. Ako pokrenemo skriptu bez argumenata, ispisat ćemo poruku da nisu proslijeđeni argumenti, a ako jesu, ispisat ćemo sve proslijeđene argumente.

```
# main.sh
#!/bin/bash

if [ $# = 0 ]; then
    echo "Nema proslijeđenih argumenata"
else
    echo "Proslijeđeni argumenti su: $@"
fi
```

Rezultat:

```
→ ./main.sh
```

```
Nema proslijeđenih argumenata
```

```
→ ./main.sh nesto_smo ipak proslijedili
```

```
Proslijeđeni argumenti su: nesto_smo ipak proslijedili
```

Primjer: Želimo provjeriti 2 uvjeta: ako je prvi argument jednak "admin123", a drugi argument jednak "tajna_lozinka", ispisujemo poruku "Prijava uspješna", inače ispisujemo "Prijava neuspješna".

Složene uvjete definiramo logičkim operatorima `&&` (AND), `||` (OR) i `!` (NOT)

```
# main.sh
#!/bin/bash

if [ "$1" = "admin123" ] && [ "$2" = "tajna_lozinka" ]; then # uvjete odvajamo operatorom
&&, a svaki pišemo unutar novog para uglatih zagrada
    echo "Prijava uspješna"
else
    echo "Prijava neuspješna"
fi
```

Rezultat:

```
→ ./main.sh admin123 tajna_lozinka
```

```
Prijava uspješna
```

```
→ ./main.sh admin123 pogresna_lozinka
```

```
Prijava neuspješna
```

Osim sintakse `[uvjet]` (jedan par uglatih zagrada), možemo koristiti i **dvostruke uglate zagrade** `[[uvjet]]` koje su fleksibilnije i podržavaju više operatora i funkcija.

- sintaksu jednostrukih uglatih nazivamo još i *POSIX-style*
- sintaksu dvostrukih uglatih nazivamo još *bash-style* (ova je modernija i preporučuje se u novim verzijama basha)

Uvjetni izraz `if` s dvostrukim uglatim zagradama:

Sintaksa:

```
if [[ uvjet ]]; then
    # blok koda koji se izvršava ako je uvjet točan
else
    # blok koda koji se izvršava ako je uvjet netočan
fi # zatvara if blok
```

Ovdje sintaksu uvjetnog izraza koristimo na nešto drugačiji način:

- kod provjere stringova koristimo `==` umjesto `=`
- ako želimo provjeriti više uvjeta, možemo koristiti `&&` i `||` **bez dodatnog para uglatih zagrada**, tj. navodimo sve uvjete unutar istog para

Raniji primjer s *bash-style* sintaksom bi izgledao ovako:

```
# main.sh
#!/bin/bash

if [[ "$1" == "admin123" && "$2" == "tajna_lozinka" ]]; then # uvjete odvajamo logičkim
    echo "Prijava uspješna"
else
    echo "Prijava neuspješna"
fi
```

3.2.1 Tablica operatora za usporedbu brojeva

Kako bismo uspoređivali numeričke vrijednosti ovom sintaksom, koristimo različite operatore numeričke usporedbe (pišemo ih kao kratke zastavice, oznakom `-`):

- Ove operatore **možemo koristiti s brojevima**, ali ne i sa stringovima
- Operatore možemo koristiti s obje sintakse, tj. i s jednostrukim i s dvostrukim uglatim zagradama

Operator	Značenje	Primjer
<code>-eq</code>	Jednako (<i>Equal to</i>)	<code>["\$a" -eq "\$b"]</code>
<code>-ne</code>	Nejednako (<i>Not equal to</i>)	<code>["\$a" -ne "\$b"]</code>
<code>-gt</code>	Veće od (<i>Greater than</i>)	<code>["\$a" -gt "\$b"]</code>
<code>-lt</code>	Manje od (<i>Less than</i>)	<code>["\$a" -lt "\$b"]</code>
<code>-ge</code>	Veće ili jednako (<i>Greater than or equal to</i>)	<code>["\$a" -ge "\$b"]</code>
<code>-le</code>	Manje ili jednako (<i>Less than or equal to</i>)	<code>["\$a" -le "\$b"]</code>

Primjer: Želimo provjeriti je li varijabla `broj` veća od `10`.

```
# main.sh
#!/bin/bash

broj=15
if [ "$broj" -gt 10 ]; then
    echo "Broj je veći od 10"
else
    echo "Broj nije veći od 10"
fi
```

Primjer: Želimo provjeriti je li godina između `2000` i `2025`.

```
# main.sh
#!/bin/bash

godina=2023
if [[ "$godina" -ge 2000 && "$godina" -le 2025 ]]; then
    echo "Godina je između 2000 i 2025"
else
    echo "Godina nije između 2000 i 2025"
fi
```

3.2.2 Tablica operatora za usporedbu stringova

Kako bismo uspoređivali stringove ovom sintaksom, koristimo različite operatore tekstualne usporedbe:

- Ove operatore **možemo koristiti sa stringovima**, ali ne i sa brojevima.
- Operatore možemo koristiti s obje sintakse, tj. i s jednostrukim i s dvostrukim uglatim zagrada

Operator	Značenje	Primjer
<code>=</code> or <code>==</code>	Jednako (<i>Equal to</i>)	<code>["\$a" = "\$b"]</code> or <code>[[\$a == \$b]]</code>
<code>!=</code>	Nejednako (<i>Not equal to</i>)	<code>["\$a" != "\$b"]</code>
<code><</code>	Manje od (leksikografska usporedba)	<code>[["\$a" < "\$b"]]</code>
<code>></code>	Veće od (leksikografska usporedba)	<code>[["\$a" > "\$b"]]</code>
<code>-z</code>	String je null (tj. duljine nula)	<code>[-z "\$a"]</code>
<code>-n</code>	String nije null	<code>[-n "\$a"]</code>

Primjer: Želimo provjeriti postoji li varijabla `ime` i je li jednaka stringu `"Marko"`.

```
# main.sh
#!/bin/bash

ime="Marko"

if [[ -n "$ime" && "$ime" == "Marko" ]]; then
    echo "Ime postoji i jednako je Marko"
else
    echo "Ime ne postoji i nije jednako Marko"
fi
```

3.2.3 Tablica operatora za provjere datoteka

Jedna od najčešćih primjena uvjetnih izraza je provjera datoteka i njihovog stanja.

- s ovim operatorima koristimo **putanju do datoteke** - relativnu ili apsolutnu
- bolja opcija je putanju pohraniti u varijablu, npr. `file="/Users/lukablaskovic/Github/FIPU-OS/OS3 - Bash skriptiranje/vjezba/main.sh"` pa koristiti to kao argument

Operator	Značenje	Primjer
<code>-e</code>	Datoteka ili direktorij postoji	<code>[-e "\$file"]</code>
<code>-f</code>	Zapis je regularna datoteka.	<code>[-f "\$file"]</code>
<code>-d</code>	Zapis je direktorij.	<code>[-d "\$dir"]</code>
<code>-s</code>	Datoteka postoji i nije prazna.	<code>[-s "\$file"]</code>
<code>-r</code>	Datoteka se može pročitati (<i>readable</i>)	<code>[-r "\$file"]</code>
<code>-w</code>	U datoteku se može pisati (<i>writable</i>)	<code>[-w "\$file"]</code>
<code>-x</code>	Datoteku se može pokretati (<i>executable</i>)	<code>[-x "\$file"]</code>
<code>-nt</code>	Datoteka <code>a</code> je novija od datoteke <code>b</code> .	<code>["\$a" -nt "\$b"]</code>
<code>-ot</code>	Datoteka <code>a</code> je starija od datoteke <code>b</code> .	<code>["\$a" -ot "\$b"]</code>

Primjer: Želimo provjeriti postoji li datoteka `main.sh` u trenutnom radnom direktoriju.

1. Pohranit ćemo radni direktorij u varijablu `radni_direktorij`
2. Definirat ćemo putanju do datoteke `main.sh` u varijablu `putanja_do_datoteke`
3. Provjerit ćemo postoji li datoteka `main.sh` u radnom direktoriju uvjetom `-e`

```
# main.sh
#!/bin/bash

radni_direktorij=$(pwd)
putanja_do_datoteke="$radni_direktorij/main.sh"
if [ -e "$putanja_do_datoteke" ]; then
    echo "Datoteka $putanja_do_datoteke postoji"
else
    echo "Datoteka $putanja_do_datoteke ne postoji"
fi
```

Primjer: Unutar `main.sh` skripte želimo provjeriti ima li skripta `test.sh` dozvolu za izvršavanje. Ako nema, dodat ćemo ju naredbom `chmod`.

```
# main.sh
#!/bin/bash

test=./test.sh

if [ -x "$test" ]; then
    echo "Datoteku je moguće pokretati"
else
    chmod +x $test
    echo "Omogućeno pisanje u datoteku"
fi
```

U bashu ne postoje boolean operatori u pravom smislu riječi (true/false), već ih nazivamo **izlaznim statusima** (eng. *exit status*) ili **izlaznim kodovima** (eng. *exit codes*).

Svaka naredba u bashu ima svoj izlazni status koji označava je li naredba bila uspješna ili ne.

- ovaj izlazni status možemo provjeriti pomoću `$?` varijable koja **vraća status posljednje izvršene naredbe**.

Ono što može biti zbunjujuće je da se u bashu izlazni statusi označavaju brojevima i to:

`0` **označava uspješno izvršenje naredbe** (da, 0 je uspješan status 😊)

`non-0` **označava neuspješno izvršenje naredbe** (npr. 1, 2, 3, ...)

Primjer u interaktivnom načinu rada:

```
→ ls

→ echo $? # provjera statusa posljednje izvršene naredbe
# Rezultat: 0

→ ls nepostojeca_datoteka
→ echo $? # provjera statusa posljednje izvršene naredbe
# Rezultat: 1
```

Na jednak način možemo provjeriti status posljednje izvršene naredbe unutar skripte, a logički izrazi koje definiramo unutar `if` bloka koristeći operatore dane u tablicama iznad, će se **naposljetku interpretirati kao boolean izraz** ovisno o tome je li uvjet točan ili nije.

Primjer u skriptnom načinu rada:

```
# main.sh
#!/bin/bash

echo "Pokrećem ls naredbu"
ls
echo "Status posljednje izvršene naredbe je: $?" # provjera statusa posljednje izvršene naredbe

if [ -e "main.sh" ]; then # izraz će se izvršiti kao 0 ili 1, u ovom slučaju 0 (istina)
    echo "Datoteka main.sh postoji"
else
    echo "Datoteka main.sh ne postoji"
fi
```

Ako bismo htjeli ručno prekinuti izvršavanje skripte, to možemo učiniti naredbom `exit` i proslijediti joj izlazni status koji želimo:

```
# main.sh
#!/bin/bash

if [ -e "skripta.sh" ]; then
    echo "Datoteka skripta.sh postoji, sve ok"
else
    echo "Datoteka skripta.sh ne postoji"
    exit 1 # izlazimo iz skripte s statusom 1 (neuspjeh)
fi
```

Provjerom statusa nakon pokretanja skripte možemo vidjeti da je skripta prekinuta s statusom `1`:

```
→ ./main.sh
Datoteka skripta.sh ne postoji
→ echo $?
# Rezultat: 1
```

Naglasimo još da je logičke izraze koje definiramo operatorima iz tablica iznad moguće pisati i u interaktivnom načinu rada. Na primjer, možemo provjeriti postoji li datoteka `main.sh` u radnom direktoriju:

```
# neće ispisati ništa, ali vraća status 0 (true) ako datoteka postoji
→ [ -e main.sh ]

# ispisuje: Datoteka postoji (zato što su oba uvjeta istinita: true && true)
→ [ -e main.sh ] && echo "Datoteka postoji"

# ako datoteka main.sh ne postoji, vrijednosti se evaluiraju u false && true || true, tj.
ispisuje: Datoteka ne postoji
→ [ -e main.sh ] && echo "Datoteka ne postoji" || echo "Datoteka ne postoji"
```

 **Ukratko:** Sve bash naredbe vraćaju izlazne statusne (`0` je uspješan, a ostali su neuspješni).

- Posljednji status možemo provjeriti sa `$?` varijablom.
- Logički izrazi koje definiramo unutar uglatih zagrada i pišemo operatorima iz tablica iznad, će se naposljetku interpretirati kao boolean izrazi, ali i oni vraćaju izlazne statuse.
- Skripti možemo prosljeđivati i argumente prilikom pozivanja, a dohvaćamo ih pomoću varijabli `$1` , `$2` , `$3` , ... do `$n` .
- Također možemo koristiti i posebne varijable `$#` (svi argumenti) i `$#` (broj prosljeđenih argumenata).
- Sve navedeno možemo izvršavati i u interaktivnom načinu rada, ne samo unutar bash skripti.

3.3 Kombiniranje više naredbi

U nastavku ćemo demonstrirati kako se može kombinirati više naredbi unutar Bash skripte, čime ćemo se osloniti na prethodno usvojeno gradivo i proširiti dosad stečeno znanje.

Primjer: Napisat ćemo bash skriptu `main.sh` koja će izvršiti sljedeće:

1. Prvo će provjeriti poziva li skriptu korisnik s argumentom `"admin"`, ako da, u varijablu `ime` pohranit ćemo `HOSTNAME` korisnika
2. Zatim će ispisati: `"Pozdrav, $ime"`
3. Ako korisnik nije `"admin"`, ispisat ćemo poruku `"Niste admin"`
4. Ako je korisnik `"admin"`, ispisat ćemo detaljni sadržaj radnog direktorija, uključujući skrivene datoteke i direktorije i omogućit ćemo pokretanje skripte `test.sh` naredbom `chmod`, ako znamo da se nalazi u istom direktoriju kao i `main.sh` skripta.
5. Nakon izmjene, napraviti ćemo opet detaljan ispis kako bi se uvjerali da je `test.sh` dobila dozvolu za izvršavanje.

Rješenje:

```
# main.sh
#!/bin/bash
if [[ -n $1 && $1 == "admin" ]]; then
ime=$HOSTNAME
    echo "Pozdrav $ime"
    ls -la
    chmod +x "$(pwd)/test.sh"
ls -la
else
echo "Niste admin"
fi
```

Primjer: Imamo direktorij `skriptni_jezici_dz`, a u njemu 3 datoteke: `index.html`, `index.js` i `style.css`. Napraviti ćemo ove datoteke i urediti ih CLI uređivačem, a zatim u direktorij dodati `main.sh` skriptu koja će izvršiti sljedeće:

1. Provjeriti postoje li datoteke `index.html`, `index.js` i `style.css`, ako ne postoje, ispisati poruku da jedna ili više datoteka ne postoji

2. Dodat ćemo argument `ispis` koji će, ako je prisutan u skripti, ispisati sadržaje svake datoteke u terminal
3. Ako argument `ispis` nije proslijeđen, zaustavljamo rad skripte

Rješenje:

```
→ touch index.html index.js style.css
```

Unosimo sadržaj u datoteku `index.html`:

```
→ nano index.html
```

```
<!-- index.html -->
<!DOCTYPE html>

<head>
  <title>Ovo je moja web stranica</title>
  <script src="script.js"></script>
</head>

<body>
  <h1>Dobrodošli na moju web stranicu</h1>
</body>

</html>
```

Unosimo sadržaj u datoteku `index.js`:

```
→ nano index.js
```

```
// index.js
console.log("Ovo je moj JavaScript kod");
```

Unosimo sadržaj u datoteku `style.css`:

```
→ nano style.css
```

```
/* style.css */
body {
  background-color: lightblue;
}
```

Sada možemo implementirati skriptu `main.sh`:

```
# main.sh
#!/bin/bash

# 1. dio
if [ -e index.html ] && [ -e index.js ] && [ -e style.css ]; then
    echo "Sve datoteke postoje."
else
    echo "Jedna ili više datoteka ne postoji"
    exit 1
fi
```

```
# main.sh

# 2. dio
if [ "$1" = "ispis" ]; then
    echo "Sadržaj datoteke index.html:"
    cat index.html
    echo ""
    echo "Sadržaj datoteke index.js:"
    cat index.js
    echo ""
    echo "Sadržaj datoteke style.css:"
    cat style.css
else
    echo "Nema proslijeđenog argumenta"
    exit 1 # nepotrebno, skripta će svakako ovdje završiti
fi
```

Rezultat:

```
→ ./main.sh
```

```
Sve datoteke postoje
Nema proslijeđenog argumenta
```

Ili pokrećemo skriptu s argumentom `ispis`:

```
→ ./main.sh ispis
```

```
Sve datoteke postoje.

Sadržaj datoteke index.html:
<!DOCTYPE html>

<head>
  <title>Ovo je moja web stranica</title>
  <script src="script.js"></script>
</head>
```

```
<body>
  <h1>Dobro dosli na moju web stranicu</h1>
</body>
```

```
</html>
```

```
Sadržaj datoteke index.js:
console.log("Ovo je moj JavaScript kod")
```

```
Sadržaj datoteke style.css:
body {
background-color: lightblue;
}
```

Zadatak 1: Provjera datoteke prema apsolutnoj putanji

Napišite bash skriptu koja će izvršiti sljedeće:

1. Skripta očekuje jedan argument koji predstavlja **apsolutnu putanju do neke datoteke na vašem računalu**
2. Ako je korisnik prosljedio više od jednog argumenta ili niti jedan, ispisujemo poruku da je potrebno prosljediti točno jedan argument i prekidamo izvršavanje skripte
3. Ako je korisnik ispravno prosljedio jedan argument, pohranjujete vrijednost argumenta u varijablu `ABS_FILE_PATH` i provjeravate složeni izraz:
 - postoji li zapis na danoj putanji?
 - i je li zapis regularna datoteka?
4. Ako su prethodne provjere točne, ispišite poruku da datoteka na putanji `ABS_FILE_PATH` postoji i ispišite njezin sadržaj.
5. Ako datoteka ne postoji, ispišite poruku da datoteka na putanji `ABS_FILE_PATH` ne postoji ili nije regularna datoteka.

Dodijelite dozvolu za izvršavanje ovoj skripti i pokrenite ju. Testirajte njeno izvršavanje sa i bez argumenata.

Primjer izvršavanja skripte:

```
→ ./provjera_datoteke.sh # ispisuje: Potrebno je proslijediti jedan argument

→ ./provjera_datoteke.sh ~/Documents/salabahter_OS.txt # ispisuje potvrdu i sadržaj
datoteke ako postoji
```

3.4 Petlje (iteracije)

Petlje (*eng. loops*) su kontrolne strukture koje omogućuju **ponavljanje određenog bloka koda više puta**, sve dok je neki uvjet zadovoljen. Kao i u drugim programskim jezicima, petlje se koriste za automatizaciju ponavljajućih zadataka i olakšavaju rad s kolekcijama podataka.

U bashu postoje 3 glavne vrste petlji:

- **for** petlja: koristi se kada je unaprijed poznat broj ponavljanja ili kada se želi iterirati kroz elemente niza
- **while** petlja: izvršava blok naredbi sve dok je uvjet istinit.
- **until** petlja: slično kao **while**, ali s obrnutom logikom: izvršava se sve dok je uvjet neistinit.

3.4.1 **for** petlje

for petlja se koristi za ponavljanje bloka koda za svaki element u nizu ili kolekciji podataka. Postoje 3 osnovna načina korištenja **for** petlje u bashu:

1. **Iteracija kroz niz**: oblik petlje koji se koristi za prolazak kroz svaki element niza/liste.
2. **C-style for petlja**: oblik petlje koji se koristi za ponavljanje bloka koda određeni broj puta, sve dok ne zadovoljimo uvjet.
3. **Iteracija kroz datoteke**: oblik petlje koji nalikuje iteraciji kroz niz, ali se koristi za prolazak svake datoteke odnosno direktorija na danoj putanji

Iteracija kroz niz

Nizove (liste) u bashu definiramo navođenjem elemenata unutar **običnih zagrada** `()` i odvajamo ih samo razmakom:

Sintaksa:

```
niz=(element_1 element_2 element_3 ... element_n)

for element in "${niz[@]}; do # iteracija kroz svaki element niza "niz"
    echo "Element je: $element"
done
```

Primjer:

```
#main.sh
#!/bin/bash

# Primjer: niz brojeva
niz=(1 2 3 4 5)

# Primjer: niz stringova
voce=("jabuka" "banana" "kivi")

# Primjer: niz stringova i brojeva
mix=("jabuka" 1 "banana" 2 kivi 3) # prisjetite se da stringovi u bashu mogu i ne moraju
biti u navodnicima
```

Primjer: Iteracija kroz niz direktnim unosom u `for` petlju:

- u ovom slučaju **niz navodimo bez zagrada**

```
# Primjer: iteracija kroz niz brojeva
for element in 1 2 3 4 5; do # gdje "element" predstavlja svaki element niza u trenutnoj
instanci iteracije
    echo "Element je: $element"
done

# ili

# Primjer: iteracija kroz niz stringova
for element in "more" "palme" "pijesak"; do # gdje "element" predstavlja svaki element
niza u trenutnoj instanci iteracije
    echo "Element je: $element"
done
```

Rezultat:

```
Element je: 1
Element je: 2
Element je: 3
Element je: 4
Element je: 5

Element je: more
Element je: palme
Element je: pijesak
```

Kako bismo pristupili određenom elementu niza, pišemo **naziv niza** i **indeks elementa unutar vitičastih zagrada** → `${niz[index]}`

- **Indeksi počinju od 0**, tj. prvi element niza ima indeks `0`, drugi element niza ima indeks `1`, itd. do `n-1` (gdje je `n` broj elemenata niza).

```
# main.sh
#!/bin/bash

niz=(1 2 3 4 5)

echo "Element na indeksu 2 je: ${niz[2]}" # ispisuje element na indeksu 2 → 3
echo "Element na indeksu 4 je: ${niz[4]}" # ispisuje element na indeksu 4 → 5

echo $niz[1] # Greška: ispisuje 1[1] → echo $niz će ispisati prvi element niza, a [1] će se interpretirati kao string
```

Česta greška: `echo $niz[1]` će ispisati `1[1]` budući da će bash interpretirati `$niz` kao varijablu (koja pri pozivu bez indeksiranja implicitno vraća prvi element niza), a `[1]` kao dio stringa koji se pritom konkatenuira.

Naravno, prilikom iteriranja češće se koristi varijabla koja sadrži niz, umjesto da se niz izravno definira unutar same petlje.

U tom slučaju koristimo oznaku `@` kao indeks odnosno: `${niz[@]}` **kako bismo prošli kroz sve elemente niza** (prisjetite da smo oznakom `@` već dobivali listu svih argumenata prosljeđenih skripti).

```
# main.sh
#!/bin/bash

niz=(1 2 3 4 5)

for element in "${niz[@]}"; do # koristimo "${niz[@]}" kako bismo prošli kroz sve elemente niza
    echo "Element je: $element"
done
```

Možemo dodavati nove vrijednosti u niz koristeći operator `+=`:

```
# main.sh
#!/bin/bash

niz=(1 2 3 4 5)

# Primjer: Dodajemo elemente 6, 7 i 8 u "niz"
niz+=(6 7 8)

echo "Elementi niza su: ${niz[@]}" # ispisuje sve elemente niza (1 2 3 4 5 6 7 8)
```

Operatorom `+=` možemo konkatenuirati i stringove:

```
# main.sh
#!/bin/bash

string="Hello"

string+=" World!" # dodajemo string " World!" u varijablu "string". Uočite da ne koristimo
()

echo "$string" # ispisuje: Hello World!
```

Veličinu niza možemo dobiti dodavanjem znaka `#` ispred izraza za dohvaćanje niza `${#niz[@]}`:

```
# main.sh
#!/bin/bash

zivotinje=(pas macka ptica slon labud)

# Primjer: ispisujemo veličinu niza dodavanjem znaka "#" ispred izraza za dohvaćanje
cijelog niza
echo "Veličina niza je: ${#niz[@]}"
```

Primjer: Za sljedeći niz: `niz=(index.html index.js helpers.js style.css)` napišite bash skriptu `create.sh` koja će stvoriti novu praznu datoteku za svaki element niza u trenutnom radnom direktoriju.

```
# create.sh
#!/bin/bash

niz=(index.html index.js helpers.js style.css)

for datoteka in "${niz[@]"; do # prolazimo kroz svaki element niza
    touch "$datoteka" # stvaramo datoteku s imenom trenutnog elementa niza
    echo "Datoteka $datoteka je stvorena"
done
```

Zapamtite sljedeće:

- `for` petlja se koristi za ponavljanje bloka koda za svaki element u nizu ili kolekciji podataka
- **nizove definiramo** navođenjem elemenata unutar **običnih zagrada** `()`, npr. `niz=(1 2 3 4 5)`
- za **pristup određenom elementu** niza koristimo `${niz[index]}` (indeksi počinju od `0`)
- za **prolazak kroz sve elemente** niza koristimo `${niz[@]}`
- za **dodavanje elemenata** u niz koristimo operator `+=`
- za **provjeru veličine** niza koristimo `${#niz[@]}`

C-style

Što se tiče **C-style** `for` petlje, ona se koristi kada želimo ponoviti blok koda određeni broj puta, sve dok je ispunjen određeni uvjet, koji je najčešće izražen aritmetičkim izrazom.

Ovaj oblik `for` petlje koristit ćemo rjeđe u našim primjerima, no važno je znati da postoji i da je njezina sintaksa slična onoj u većini programskih jezika.

Sintaksa:

```
for (( inicijalizacija; uvjet; inkrementacija )); do # Uočite dvostruke oklepe (obične)
    zagrada
    # blok koda koji se izvršava dok je uvjet zadovoljen
done
```

Primjer: Iteriranje kroz brojeve od 0 do 4.

```
# main.sh
#!/bin/bash

# Primjer C-style for petlje
for (( i=0; i<5; i++ )); do # gdje "i" predstavlja brojač
    echo "Brojač je: $i"
done
```

Rezultat:

```
Brojač je: 0
Brojač je: 1
Brojač je: 2
Brojač je: 3
Brojač je: 4
```

Primjer: Možemo koristiti `break` i `continue` naredbe unutar `for` petlje:

- `break` se koristi za prekid izvršavanja aktualne petlje (ili unutarnje petlje ako je ugniježđena)
- `continue` se koristi za preskakanje trenutne iteracije i prelazak na sljedeću iteraciju

```
# main.sh
#!/bin/bash

for (( i=0; i<5; i++ )); do
    if [ $i -eq 2 ]; then
        echo "Preskočena iteracija s brojačem 2"
        continue # preskoči trenutnu iteraciju (i = 2)
    fi
    if [ $i -eq 4 ]; then
        echo "Prekid petlje s brojačem 4"
        break # prekini petlju (i = 4)
    fi
    echo "Brojač je: $i"
```

done

Rezultat:

```
Brojač je: 0
Brojač je: 1
Preskočena iteracija s brojačem 2
Brojač je: 3
Prekid petlje s brojačem 4
```

Treba naglasiti da varijabla definirana unutar petlje prilikom inicijalizacije **nema lokalni doseg** kao što je to slučaj u većini drugih programskih jezika. Drugim riječima, **varijabla će zadržati svoju vrijednost i nakon završetka petlje**, tj. bit će dostupna i izvan petlje.

Dakle, u primjeru ispod, varijabla `i` će zadržati svoju vrijednost i nakon završetka petlje:

```
# main.sh
#!/bin/bash

for (( i=0; i<5; i++ )); do
    echo "Brojač je: $i"
done

echo "Brojač izvan petlje je: $i" # Ispisuje: Vrijednost brojača izvan petlje je 5
```

Općenito, varijable koje definiramo možemo **deallocirati** naredbom `unset`:

```
# main.sh
#!/bin/bash

for (( i=0; i<5; i++ )); do
    echo "Brojač je: $i"
done

unset i # deallociramo varijablu "i"
echo "Brojač izvan petlje je: $i" # Ispisuje: Brojač izvan petlje je: (prazno)

# Primjer: naredbom možemo deallocirati i nizove

niz=(1 2 3 4 5)
unset niz # deallociramo varijablu "niz"

# Primjer: naredbom unset možemo deallocirati i pojedinačne elemente niza

voce=("jabuka" "banana" "kivi")
unset voce[1] # deallociramo drugi element niza (banana)
echo "Elementi niza su: ${voce[@]}" # Ispisuje: jabuka kivi
```



Napomena: U bashu postoji i naredba `set` međutim ona se ne koristi za dodjeljivanje vrijednosti varijablama već za **izmjenu opcija aktivnog shella**. Za dodjeljivanje vrijednosti nekoj varijabli koristimo jednostavnu sintaksu `varijabla=vrijednost` bez razmaka.

Iteracija kroz datoteke

Iteracija kroz datoteke u bashu omogućuje čitanje i manipulaciju datotekama i direktorijima unutar bash skripti, čime se olakšava automatizacija obrade podataka, upravljanje sadržajem te izvršavanje ponavljajućih zadataka nad većim brojem elemenata datotečnog sustava.

Ova vrsta petlje osobito je korisna kada je potrebno obraditi više datoteka ili direktorija istovremeno, a njezina je sintaksa slična onoj za iteraciju kroz nizove. Razlika je u tome što se **umjesto niza navodi putanja do direktorija koji želimo iterirati**.

Za dohvat svih datoteka unutar određenog direktorija koristi se zamjenski znak `*` (wildcard).

Sintaksa:

```
for datoteka in /putanja/do/direktorija/*; do # prolazimo kroz svaku datoteku u
direktoriju na danoj putanji
    echo "Datoteka je: $datoteka"
done
```

- gdje `/putanja/do/direktorija` može biti **relativna (u odnosu na skriptu)** ili **apsolutna putanja do direktorija**
- `*` wildcard označava sve datoteke u tom direktoriju
- ako putanja do direktorija sadrži naziv direktorija s razmacima, moramo ju omotati u navodne znakove (`" "`)

Primjer: želimo ispisati sadržaj direktorija gdje se nalazi ova skripta: `"/Users/lukablaskovic/Github/FIPU-OS/OS3 - Bash skriptiranje"`.

Obzirom da postoje razmaci u nazivu direktorija, omotat ćemo ga navodnim znakovima:

```
for zapis in "/Users/lukablaskovic/Github/FIPU-OS/OS3 - Bash skriptiranje"/*; do # primjer
iteriranja prema apsolutnoj putanji
    echo "Zapis: $zapis" # ispisuje apsolutnu putanju do svakog zapisa
done
```

Rezultat:

```
Zapis: /Users/lukablaskovic/Github/FIPU-OS/OS3 - Bash skriptiranje/illustrations
Zapis: /Users/lukablaskovic/Github/FIPU-OS/OS3 - Bash skriptiranje/kviz
Zapis: /Users/lukablaskovic/Github/FIPU-OS/OS3 - Bash skriptiranje/OS3 - Bash
skriptiranje.md
Zapis: /Users/lukablaskovic/Github/FIPU-OS/OS3 - Bash skriptiranje/OS3 - Bash
skriptiranje.pdf
Zapis: /Users/lukablaskovic/Github/FIPU-OS/OS3 - Bash skriptiranje/screenshots
Zapis: /Users/lukablaskovic/Github/FIPU-OS/OS3 - Bash skriptiranje/vjezba_sat
```

U rezultatima vidimo da su ispisane apsolutne putanje **i direktorija i datoteka** za sve zapise `"OS3 - Bash skriptiranje"`.

Kako bismo **ispisali samo datoteke**, možemo kombinirati ovu petlju odgovarajućom `if` selekcijom:

```
for zapis in "/Users/lukablaskovic/Github/FIPU-OS/OS3 - Bash skriptiranje"/*; do
    if [ -f "$zapis" ]; then # provjeravamo je li "zapis" regularna datoteka
        echo "Datoteka: $zapis" # ispisuje apsolutnu putanju do datoteke
    fi
done
```

Rezultat:

```
Datoteka: /Users/lukablaskovic/Github/FIPU-OS/OS3 - Bash skriptiranje/OS3 - Bash
skriptiranje.md
Datoteka: /Users/lukablaskovic/Github/FIPU-OS/OS3 - Bash skriptiranje/OS3 - Bash
skriptiranje.pdf
```

Naravno, kao i kod nizova, možemo pohraniti putanju u varijablu i koristiti ju unutar petlje.

Primjer: Iterirajmo trenutni radni direktorij i ispišimo sve datoteke u njemu.

```
# main.sh
#!/bin/bash

radni_dir=$(pwd) # pohranjujemo trenutni radni direktorij u varijablu

for zapis in "$radni_dir"/*; do # prolazimo kroz svaki "zapis" u radnom direktoriju
    if [ -f "$zapis" ]; then # provjeravamo je li "zapis" regularna datoteka
        echo "Datoteka: $zapis" # ispisuje apsolutnu putanju do datoteke
    fi
done
```

Primjer: Ako bismo htjeli iterirati datoteke samo s određenom ekstenzijom, npr. samo `.js` datoteke, možemo nakon `*` dodati ekstenziju. Napraviti ćemo novi direktorij i u njemu nekoliko datoteka s različitim ekstenzijama.

```
→ mkdir projekt_js
→ cd projekt_js

→ touch index.html index.js style.css utils.js router.js

→ nano filter.sh
```

Sada ćemo iterirati samo `.js` datoteke izrazom `*.js`:

```
# filter.sh
#!/bin/bash

radni_dir=$(pwd) # pohranjujemo trenutni radni direktorij u varijablu

for datoteka in "$radni_dir"/*.js; do # prolazimo kroz svaku .js datoteku u radnom
direktoriju
    if [ -f "$datoteka" ]; then # provjeravamo je li zapis regularna datoteka
        echo "Datoteka je: $datoteka"
    fi
done
```

Rezultat:

```
Datoteka je: /Users/lukablaskovic/Github/FIPU-OS/OS3 - Bash
skriptiranje/projekt_js/index.js
Datoteka je: /Users/lukablaskovic/Github/FIPU-OS/OS3 - Bash
skriptiranje/projekt_js/utils.js
Datoteka je: /Users/lukablaskovic/Github/FIPU-OS/OS3 - Bash
skriptiranje/projekt_js/router.js
```

Ako bismo htjeli ispisati **samo nazive datoteka bez apsolutne putanje**, možemo koristiti `basename` naredbu i kombinirati ju supstitucijom varijabli:

Sintaksa:

```
basename "$datoteka" # ispisuje samo naziv datoteke bez putanje

# Primjer apsolutne putanje do datoteke "test.sh"
basename "/Users/lukablaskovic/Github/FIPU-OS/OS3 - Bash skriptiranje/vjezba_sat/test.sh"
# ispisuje: test.sh
```

```
for datoteka in "$radni_dir"/*.js; do # prolazimo kroz svaku .js datoteku u radnom
direktoriju
    if [ -f "$datoteka" ]; then # provjeravamo je li zapis regularna datoteka
        echo "Datoteka je: $(basename "$datoteka")" # ispisujemo samo naziv datoteke bez
putanje (supstitucija varijabli)
    fi
done
```

Rezultat:

```
Datoteka je: index.js
Datoteka je: utils.js
Datoteka je: router.js
```

Moguće je i često ćemo koristiti relativne putanje do direktorija umjesto apsolutnih. U tom slučaju, **relativna putanja je relativna u odnosu na lokaciju skripte koja se izvršava**.

Primjer: Definirat ćemo novi direktorij `server` i unutar njega datoteke `index.js` i `package.json`. Također ćemo dodati direktorij `routes` i unutar njega datoteke: `main.js`, `shop.js` i `cart.js`.

Dakle, struktura nam izgleda ovako:

```
server
├── index.js
├── package.json
└── routes
    ├── cart.js
    ├── main.js
    └── shop.js
```

Nalazimo se u direktoriju `server` i želimo unutar njega napraviti novu bash skriptu `prepare.sh`.

```
→ cd server
→ nano prepare.sh
```

Iteriranje kroz sve zapise unutar direktorija `routes` koristeći relativnu putanju:

Rezultat:

```
# prepare.sh
#!/bin/bash

for zapis in routes/*; do # prolazimo kroz svaki zapis u direktoriju "routes" - relativna
putanja
    if [ -f "$zapis" ]; then
        echo "Datoteka: ${basename "$zapis"}"
    fi
done
```

Jednom kad smo usvojili načine iteracije i rada s listama, kako možemo na jednostavan način u listu pohraniti nazive datoteka iz direktorija `routes`?

Rezultat:

```
# prepare.sh
#!/bin/bash

datoteke=() # inicijaliziramo praznu listu
for zapis in routes/*; do # prolazimo kroz svaki zapis u direktoriju "routes"
    if [ -f "$zapis" ]; then
        datoteke+=("${basename "$zapis"}") # dodajemo naziv datoteke u listu
    fi
done
echo "Datoteke su: ${datoteke[@]}" # ispisujemo sve datoteke
```

3.4.2 while i until petlje

`while` i `until` petlje se koriste za ponavljanje bloka koda sve dok je neki uvjet zadovoljen odnosno nije ispunjen (za `until` petlju).

Sintaksa:

`while` petlja će se izvršavati sve **dok je uvjet zadovoljen**:

```
while [ uvjet ]; do
    # blok koda koji se izvršava dok je uvjet zadovoljen
done
```

`until` petlja će se izvršavati sve **dok uvjet nije zadovoljen**:

```
until [ uvjet ]; do
    # blok koda koji se izvršava dok uvjet nije zadovoljen
done
```

Primjer: Iterirati ćemo sve dok `brojac` ne bude jednak nuli.

```
# main.sh
#!/bin/bash

brojac=10 # inicijaliziramo brojac

while [ $brojac -gt 0 ]; do # WHILE brojac is greater than 0
    echo "Brojač je: $brojac"
    brojac=$((brojac - 1)) # smanjujemo brojac za 1
    # ili
    ((brojac--))
done

# odnosno s until petljom

until [ $brojac -eq 0 ]; do # UNTIL brojac is equal to 0
    echo "Brojač je: $brojac"
    brojac=$((brojac - 1)) # smanjujemo brojac za 1
done
```

Aritmetika

Ono što nas vjerojatno trenutno buni u primjeru iznad je izraz: `((brojac--))`, odnosno zagrade oko izraza `brojac--`.

Dvostrukim zagradama `(())` možemo označavati **aritmetički izraz** (eng. *arithmetic expression*) u kojem možemo izvoditi aritmetičke operacije:

Sintaksa:

```
$(aritmeticki_izraz)
```

- gdje je `aritmeticki_izraz` bilo koji aritmetički izraz koji želimo izračunati
- ako navodimo varijable unutar aritmetičkog izraza, navodimo ih bez oznake `$`, npr. `$(varijabla + varijabla_2)`

Primjer: Ispravan i pogrešan način korištenja aritmetičkog izraza `++`.

```
# main.sh
#!/bin/bash

brojac=10 # inicijaliziramo brojac

brojac=$brojac -1 # GREŠKA! bash će ovo tumačiti kao dvije odvojene naredbe
(brojac=$brojac i -1)
brojac=$((brojac - 1)) # ispravno! bash će ovo tumačiti kao aritmetički izraz
brojac=$((brojac--)) # ispravno! bash će ovo tumačiti kao aritmetički izraz
```

Primjer: U nastavku je prikazano nekoliko aritmetičkih izraza koje možemo koristiti u bashu.

```
# main.sh
#!/bin/bash

operand_1=10
operand_2=5

rezultat_neispravno=$operand_1 + $operand_2 # GREŠKA! +: command not found

zbroj=$((operand_1 + operand_2))
echo "Zbroj je: $zbroj"

razlika=$((operand_1 - operand_2)) # ispravno! razlika 5
echo "Razlika je: $razlika" # ispisuje: Razlika je: 5

umnozak=$((operand_1 * operand_2)) # ispravno! umnožak 50
echo "Umnožak je: $umnozak" # ispisuje: Umnožak je: 50

kolicnik=$((operand_1 / operand_2)) # ispravno! kolicnik 2
echo "Količnik je: $kolicnik" # ispisuje: Količnik je: 2

ostatak=$((operand_1 % operand_2)) # ispravno! ostatak 0
echo "Ostatak je: $ostatak" # ispisuje: Ostatak je: 0

potencija=$((operand_1 ** operand_2)) # ispravno! potencija 100000
echo "Potencija je: $potenciranje" # ispisuje: Potencija je: 100000
```

Primjer: Napišite bash skriptu koja će izvršiti sljedeće:

- U varijablu `broj_zapisa` će pohraniti **broj datoteka i direktorija** u trenutnom radnom direktoriju

2. Iterirati će kroz sve datoteke i direktorije u trenutnom radnom direktoriju
3. Za svaku datoteku odnosno direktorij ćemo dodati prefiks: `"n_"` gdje je `n` redni broj zapisa (datoteke ili direktorija)

```
# main.sh
#!/bin/bash

broj_zapisa=1
radni_dir=$(pwd)

for zapis in "$radni_dir"/*; do # prolazimo kroz svaku datoteku u radnom direktoriju
    ime_datoteke=$(basename "$zapis")
    if [ -e "$zapis" ]; then # provjeravamo postoji li zapis
        mv "$zapis" "$radni_dir"/"$broj_zapisa"_"$ime_datoteke" # mv <izvor> <odredište>
        broj_zapisa=$((broj_zapisa + 1)) # povećavamo broj_zapisa za 1
    fi
done

echo "Broj zapisa u radnom direktoriju je: $broj_zapisa" i svi zapisi su preimenovani
```

Uočite kako smo preimenovali datoteke:

- `mv` naredba se koristi za premještanje ili preimenovanje (ako su izvor i odredište u istom direktoriju)
- `mv <izvor> <odredište>` - premješta datoteku iz `<izvora>` u `<odredište>`
- `<izvor>` je `$zapis` (apsolutna putanja) - lokalna varijabla u iteraciji radnog direktorija
- `<odredište>` je `$radni_dir` (apsolutna putanja) + `$broj_zapisa` (redni broj zapisa) + `_` (donja crta) + `$ime_datoteke` (ime datoteke koje smo dobili pomoću `basename` naredbe)

Aritmetičke izraze napisane unutar dvostrukih zagrada možemo koristiti i unutar `if` naredbi:

```
# main.sh
#!/bin/bash

# Primjer: provjeravamo je li broj paran ili neparan
broj=5

if [[ $((broj % 2)) -eq 0 ]]; then # provjeravamo je li ostatak pri dijeljenju s 2 jednak nuli
    echo "$broj je paran broj"
else
    echo "$broj je neparan broj"
fi
```

Međutim, u ovom slučaju je moguće i skratiti sintaksu i ukloniti dvostruke zagrade oko aritmetičkog izraza (ako je aritmetički uvjet jedini uvjet):

```
# main.sh
#!/bin/bash

# Primjer: provjeravamo je li broj paran ili neparan
broj=5
if (( broj % 2 == 0 )); then # provjeravamo je li ostatak pri dijeljenju s 2 jednak nuli
    echo "$broj je paran broj"
else
    echo "$broj je neparan broj"
fi
```

Ako aritmetički izraz nije jedini uvjet već je dio složenijeg uvjeta, moramo koristiti dvostruke uglate zagrade:

```
# main.sh
#!/bin/bash

# Primjer: provjeravamo je li broj paran i postoji li datoteka na putanji

broj=5
putanja="/Users/lukablaskovic/Github/FIPU-OS/OS3 - Bash skriptiranje/main.sh"

if [[ -f "$putanja" && $((broj % 2)) -eq 0 ]]; then # provjeravamo je li ostatak pri
dijeljenju s 2 jednak nuli
    echo "$broj je paran broj i datoteka postoji"
else
    echo "$broj je neparan broj ili datoteka ne postoji"
fi
```

3.5 Funkcije

Funkcije su blokovi koda koji se mogu višekratno koristiti unutar skripte. Funkcije omogućuju organizaciju koda, ponovnu upotrebu i olakšavaju održavanje.

Definiraju se pomoću ključne riječi `function` ili jednostavno nazivom funkcije, nakon čega slijedi tijelo funkcije unutar `{}` zagrada. Argumentima koje šaljemo funkciji možemo direktno pristupati pomoću oznaka `$1`, `$2`, `$3`, itd. (slično kao argumenti u kontekstu bash skripte).

Sintaksa:

```
function naziv_funkcije() { # puni oblik definicije funkcije
    # tijelo funkcije
}

# ili

naziv_funkcije() { # skraćeni oblik definicije funkcije
    # tijelo funkcije
}
```

Argumenti unutar funkcije se koriste na isti način kao i argumenti skripte, tj. `$1`, `$2`, `$3`, itd. do `$n` (gdje je `n` broj argumenata proslijeđenih funkciji). Također, unutar funkcije možemo koristiti i varijable definirane izvan funkcije.

Jednako kao kod petlji, **varijable definirane unutar funkcije nemaju lokalni doseg**, tj. zadržavaju svoju vrijednost i nakon završetka funkcije. Ako želimo da varijabla bude lokalna, moramo ju definirati s `local` ključnom riječi unutar tijela funkcije.

Sintaksa:

```
function naziv_funkcije() {  
    varijabla="vrijednost globalne varijable" # globalna varijabla  
    # tijelo funkcije  
    local lokalna_varijabla="vrijednost lokalne varijable" # lokalna varijabla  
    # tijelo funkcije  
    ...  
}
```

Primjer: Funkcija koja dodaje dva broja i ispisuje njihov zbroj.

```
function dodaj_brojeve() {  
    sum=$(( $1 + $2 )) # zbrajamo dva argumenta  
    echo "Zbroj je: $sum" # ispisujemo zbroj  
}
```

Funkciju pozivamo bez operatora `()`

```
# main.sh  
dodaj_brojeve 5 10 # pozivamo funkciju s argumentima 5 i 10  
echo "Sum vrijedi i izvan funkcije: $sum" # varijabla sum nije lokalna i može se koristiti  
izvan funkcije
```

Rezultat:

```
Zbroj je: 15  
Sum vrijedi i izvan funkcije: 15
```

Ipak, ako ju definiramo kao lokalnu, nećemo moći koristiti varijablu izvan funkcije:

```
# main.sh  
function dodaj_brojeve() {  
    local sum=$(( $1 + $2 )) # zbrajamo dva argumenta  
    echo "Zbroj je: $sum" # ispisujemo zbroj  
}  
  
dodaj_brojeve 5 10 # pozivamo funkciju s argumentima 5 i 10  
echo "Sum više ne vrijedi izvan funkcije: $sum" #
```

Bash funkcije ne vraćaju povratne vrijednosti kao što je to slučaj u većini drugih programskih jezika. Umjesto toga, možemo koristiti `echo` naredbu unutar funkcije kako bismo ispisali rezultat koji možemo pohraniti u varijablu prilikom poziva funkcije (kao što smo prikazali iznad s varijablom `sum`).

Ipak, postoji mogućnost vraćanja **izlaznog statusa** funkcije pomoću `return` naredbe, slično kao što smo vidjeli kod `if` naredbi i složenijih izraza.

Izlazni status funkcije može biti bilo koji cijeli broj između `0` i `255`, gdje `0` označava uspješan završetak funkcije, a svi ostali brojevi označavaju različite vrste pogrešaka ili neuspjeha.

Sintaksa:

```
function naziv_funkcije() {  
    # tijelo funkcije  
    return 0 # uspješan završetak funkcije  
    # ili  
    return 1 # neuspješan završetak funkcije  
}
```

Primjer: Funkcija koja dodaje dva broja i vraća njihov zbroj. Ako je broj argumenata različit od `2`, vraća neuspješan status, inače vraća uspješan status i sprema zbroj u varijablu `sum`.

```
# main.sh  
function dodaj_brojeve() {  
    if [ $# -ne 2 ]; then # provjeravamo broj argumenata  
        echo "Pogrešan broj argumenata"  
        return 1 # vraćamo neuspješan status  
    fi  
    sum=$(( $1 + $2 )) # zbrajamo dva argumenta  
    echo "Zbroj je: $sum" # ispisujemo zbroj  
    return 0 # vraćamo uspješan status  
}  
  
dodaj_brojeve 5 10 # pozivamo funkciju s argumentima 5 i 10 → vraća statusni kod 0, a sum je 15  
dodaj_brojeve 5 # pozivamo funkciju s jednim argumentom → vraća statusni kod 1, a sum nije definiran
```

Primjer: Funkcija koja provjerava je li broj paran ili neparan. Ako je broj paran, vraća `0`, inače vraća `1`.

```
# main.sh
function provjeri_parnost() {
    if [ $(( $1 % 2 )) -eq 0 ]; then # provjeravamo je li broj paran
        echo "$1 je paran broj"
        return 0 # vraćamo uspješan status
    else
        echo "$1 je neparan broj"
        return 1 # vraćamo neuspješan status
    fi
}

provjeri_parnost 5 # pozivamo funkciju s argumentom 5 → vraća statusni kod 1
provjeri_parnost 10 # pozivamo funkciju s argumentom 10 → vraća statusni kod 0
```

Primjer: Funkcija koja provjerava sve datoteke na danoj apsolutnoj putanji i ispisuje samo one datoteke s nastavkom `.html` i pohranjuje ih u niz `html_datoteke`. Za svaku takvu datoteku, druga funkcija će ispisati sadržaj datoteke.

```
# main.sh
#!/bin/bash

function provjeri_html() {
    local putanja=$1 # pohranjujemo putanju u lokalnu varijablu
    local html_datoteke=() # inicijaliziramo praznu listu

    for datoteka in "$putanja"/*.html; do # prolazimo kroz svaku datoteku u direktoriju
        if [ -f "$datoteka" ]; then # provjeravamo je li zapis regularna datoteka
            html_datoteke+=("${basename "$datoteka"}") # dodajemo naziv datoteke u listu "html_datoteke"
            echo "Datoteka: ${basename "$datoteka"}" # ispisujemo naziv datoteke
        fi
    done

    echo "HTML datoteke su: ${html_datoteke[@]}" # ispisujemo sve HTML datoteke
}

provjeri_html "/Users/lukablaskovic/Github/fipu-js/1. Javascript osnove" # pozivamo funkciju s apsolutnom putanjom do direktorija "1. JavaScript osnove"
```

Rezultat:

```
HTML datoteke su: index.html jos_jedna.html
```

Dodajemo funkciju koja će ispisati sadržaj datoteke:

```
function ispiši_sadržaj() {
    local datoteka=$1 # pohranjujemo putanju do datoteke u lokalnu varijablu
    echo "Sadržaj datoteke $datoteka je:"
    cat "$datoteka" # ispisujemo sadržaj datoteke
}
```

```
function provjeri_html() {
    local putanja=$1
    local html_datoteke=()

    for datoteka in "$putanja"/*.html; do
        if [ -f "$datoteka" ]; then
            html_datoteke+=("${basename "$datoteka"}")
            echo "Datoteka: ${basename "$datoteka"}"
            ispiši_sadržaj "$datoteka" # pozivamo funkciju za ispis sadržaja datoteke
        fi
    done

    echo "HTML datoteke su: ${html_datoteke[@]}"
}
```

Primjer: U bashu možemo koristiti naredbu `zip` za kompresiju/komprimiranje datoteka i direktorija. U ovom primjeru ćemo definirati funkciju koja će komprimirati sve `.html` datoteke na danoj putanji i pohraniti ih u `.zip` datoteku.

Sintaksa:

```
→ zip <naziv_zip_datoteke> <datoteka_1> <datoteka_2> ... <datoteka_n>
```

```
# ili
```

```
→ zip -r <naziv_zip_datoteke> <direktorij> # dodajemo rekurzivnu zastavicu -r koja će komprimirati sve datoteke unutar direktorija
```

```
→ nano zip_html.sh
```

```
# zip_html.sh
#!/bin/bash

function zip_html() {
    local putanja=$1 # pohranjujemo putanju u lokalnu varijablu
    local zip_datoteka="html_datoteke.zip" # naziv zip datoteke, može biti bilo koji naziv

    zip -r "$zip_datoteka" "$putanja"/*.html # komprimiramo sve .html datoteke u zip datoteku
    echo "HTML datoteke su komprimirane u $zip_datoteka"
}
```

Dodatno, skriptu možemo unaprijediti na način da **kao argument prima putanju do direktorija** koji želimo komprimirati.

```
# zip_html.sh
#!/bin/bash
```

```

direktorij=$1

function zip_html() {
    local putanja=$1 # pohranjujemo putanju u lokalnu varijablu
    local zip_datoteka="html_datoteke.zip" # naziv zip datoteke, može biti bilo koji naziv

    zip -r "$zip_datoteka" "$putanja"/*.html # komprimiramo sve .html datoteke u zip
    datoteku
    echo "HTML datoteke su komprimirane u $zip_datoteka"
}

zip_html "$direktorij" # pozivamo funkciju s apsolutnom putanjom do direktorija "1.
JavaScript osnove"

```

Recimo da imamo sljedeću datotečnu strukturu:

```

1. Javascript osnove
├─ index.html
├─ structure.html
├─ test.html
├─ slika.png
└─ zip_html.sh

```

Sada možemo pokrenuti skriptu i proslijediti joj putanju do direktorija `1. Javascript osnove`:

```

→ cd 1. Javascript osnove
→ zip_html.sh "/Users/lukablaskovic/Github/FIPU-OS/OS3 - Bash skriptiranje/vjezba_sat/1.
JavaScript osnove"

```

Raspakirajte datoteku `html_datoteke.zip` i vidjet ćete da smo uspješno komprimirali sve `.html` datoteke unutar direktorija `1. Javascript osnove`, **ali i sve korijenske direktorije definirane prema apsolutnoj putanji!**

```

Users
├─ lukablaskovic
│   └─ Github
│       └─ FIPU-OS
│           └─ OS3 - Bash skriptiranje
│               └─ vjezba_sat
│                   └─ 1. Javascript osnove
│                       ├── index.html
│                       ├── structure.html
│                       ├── test.html
│                       ├── slika.png
│                       └─ zip_html.sh

```

- To je zato što smo koristili apsolutnu putanju do direktorija kod naredbe `zip`.

💡 **Napomena:** Ova se situacija često javlja prilikom korištenja apsolutnih putanja (ne samo kod naredbe zip), no moguće ju je izbjeći na više načina. Najjednostavniji i ujedno najpouzdaniji pristup, primjenjiv u gotovo svim slučajevima, jest da se **prije izvođenja željene operacije prebacimo u direktorij nad kojim želimo raditi**.

Prebacivanjem u direktorij koji želimo komprimirati ćemo izbjeći da se u zip datoteku dodaju svi korijenski direktoriji.

```
# zip_html.sh
#!/bin/bash

direktorij=$1

function zip_html() {
    local putanja=$1
    local zip_datoteka="html_datoteke.zip"

    cd "$putanja" # prebacujemo se u direktorij koji želimo komprimirati
    zip -r "$zip_datoteka" *.html
    echo "HTML datoteke su komprimirane u $zip_datoteka"
}
zip_html "$direktorij" # pozivamo funkciju s apsolutnom putanjom do direktorija "1.
JavaScript osnove"
```

Sada kada pokrenemo skriptu, dobit ćemo samo `.html` datoteke unutar direktorija `1. Javascript osnove`.

💡 **Napomena:** Datoteke možemo raspakirati ručno ili ekvivalentom naredbom `unzip`

3.6 Kratki pregled

Shebang navodimo na početku svake skripte i on označava koji interpreter će se koristiti za izvršavanje skripte. U ovom slučaju koristimo `bash` interpreter.

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

Skriptu otvaramo koristeći CLI editor, npr. `nano` ili `vim`:

```
→ nano ime_skripte.sh
→ vim ime_skripte.sh
```

Prije nego što pokrenemo skriptu, moramo joj dodati dozvolu za izvršavanje:

```
→ chmod +x ime_skripte.sh
→ ./ime_skripte.sh # navodimo putanju do skripte
```

Varijable:


```

varijabla="vrijednost" # definiranje varijable s navodnicima
varijabla=vrijednost # definiranje varijable bez navodnika
echo $varijabla # ispisivanje varijable
echo "${varijabla}" # ispisivanje varijable unutar navodnih znakova

radni_dir=$(pwd) # supstitucija varijabli - koristeći $()
slozeni_ispis=$(ls -la) # supstitucija varijabli - koristeći $()

```

Argumenti skripte:

```

echo "Argument 1: $1" # ispis prvog argumenta
echo "Argument 2: $2" # ispis drugog argumenta
echo "Niz svih proslijeđenih argumenata: $@" # ispis svih proslijeđenih argumenata
echo "Broj proslijeđenih argumenata: $# " # ispis broja proslijeđenih argumenata

→ ./ime_skripte.sh "argument 1" arg_2 # poziv skripte s argumentima

```

Selekcija:

```

if [ uvjet ]; then # provjeravamo uvjet
    # blok koda koji se izvršava ako je uvjet zadovoljen
elif [ uvjet ]; then # provjeravamo drugi uvjet
    # blok koda koji se izvršava ako je drugi uvjet zadovoljen
else
    # blok koda koji se izvršava ako nijedan od uvjeta nije zadovoljen
fi

# koristeći dvostruke uglate zagrade

if [[ uvjet ]]; then # provjeravamo uvjet
    # blok koda koji se izvršava ako je uvjet zadovoljen
elif [[ uvjet ]]; then # provjeravamo drugi uvjet
    # blok koda koji se izvršava ako je drugi uvjet zadovoljen
else
    # blok koda koji se izvršava ako nijedan od uvjeta nije zadovoljen
fi

# ili koristeći aritmetičke izraze

if (( uvjet )); then # provjeravamo uvjet
...

```

Operatori za **aritmetičke** usporedbe jesu: `-eq`, `-ne`, `-gt`, `-lt`, `-ge`, `-le`

Operatori za **string** usporedbe su: `=`, `!=`, `<`, `>`, `-z`, `-n`

Operatori za **logičke** usporedbe su: `-a`, `-o`, `!`

Operatori za provjeru **datoteka/direktorija** su: `-e`, `-f`, `-d`, `-r`, `-w`, `-x`, `-s`

Ne postoje boolean vrijednosti u bashu, ali postoje **izlazni statusi** koji predstavljaju uspješan ili neuspješan završetak naredbe.

- Uspješan završetak je `0`
- Neuspješan završetak su svi ostali brojevi između `1` i `255`

Izlazni status posljednje naredbe možemo provjeriti pomoću `$?` varijable.

```
if [ $? -eq 0 ]; then # provjeravamo je li posljednja naredba završila uspješno
    echo "Posljednja naredba je završila uspješno"
else
    echo "Posljednja naredba nije završila uspješno"
fi
```

Nizovi (liste):

```
niz=(element_1 element_2 element_3) # definiranje niza
echo "${niz[@]}" # ispis svih elemenata niza
echo "${niz[0]}" # ispis prvog elementa niza
echo "${niz[1]}" # ispis drugog elementa niza
echo "${#niz[@]}" # ispis broja elemenata niza
```

Petlje (iteracije):

```
# Iteracija kroz niz
for element in (1 2 3); do # prolazimo kroz svaki element niza
    echo "Element je: $element" # ispisujemo element
done

# ili

for element in "${niz[@]"; do # prolazimo kroz svaki element niza
    echo "Element je: $element" # ispisujemo element
done

# C-style for petlja

for (( i=0; i<10; i++ )); do # prolazimo kroz svaki broj od 0 do 9
    echo "Broj je: $i" # ispisujemo broj
done

# while petlja

brojac=10 # inicijaliziramo brojac

while [ $brojac -gt 0 ]; do # WHILE brojac is greater than 0
    echo "Brojač je: $brojac"
    brojac=$((brojac - 1)) # smanjujemo brojač za 1
done

# until petlja

until [ $brojac -eq 0 ]; do # UNTIL brojac is equal to 0
```

```

    echo "Brojač je: $brojac"
    brojac=$((brojac - 1)) # smanjujemo brojač za 1
done

# Iteracija kroz datoteke u direktoriju

radni_dir=$(pwd) # pohranjujemo trenutni radni direktorij u varijablu

for datoteka in "$radni_dir"/*; do # prolazimo kroz svaku datoteku u radnom direktoriju
    if [ -f "$datoteka" ]; then # primjer selekcije unutar petlje
        echo "Datoteka je: $(basename "$datoteka")" # ispisujemo naziv datoteke bez
korijenskih direktorija (apsolutne putanje)
    fi
done

```

Funkcije

```

function naziv_funkcije() { # definiranje funkcije
    # tijelo funkcije
}
# ili
naziv_funkcije() { # definiranje funkcije
    # tijelo funkcije
}

# pozivanje funkcije
naziv_funkcije # pozivamo funkciju bez operatora ()

```

```

function naziv_funkcije() { # definiranje funkcije
    arg_1=$1 # pohranjujemo prvi argument u varijablu
    varijabla="vrijednost globalne varijable" # globalna varijabla
    # tijelo funkcije
    local lokalna_varijabla="vrijednost lokalne varijable" # lokalna varijabla
    # tijelo funkcije
}
naziv_funkcije vrijednost_argumenta # pozivamo funkciju s argumentom
"vrijednost_argumenta"

```

Zadaci za Vježbu 3

Zadatak 1

Napišite bash skriptu koja će primiti 2 argumenta:

- prvi argument je apsolutna putanja do direktorija
- drugi argument je datotečni nastavak (npr. `.txt`, `.html`, `.sh`, itd.)

Skripta mora proći kroz sve datoteke u direktoriju i ispisati samo one datoteke koje imaju zadani nastavak. Ako ne postoji nijedna datoteka s tim nastavkom, skripta mora ispisati poruku da nema takvih datoteka.

U ispisu moraju biti uključene samo datoteke i to njihovi nazivi bez korijenskih direktorija (apsolutnih putanja) Ako korisnik ne proslijedi točno 2 argumenta, prekinite rad skripte i ispišite poruku da je potrebno proslijediti točno 2 argumenta.

Zadatak 2

Napišite bash skriptu koja će primiti 1 argument: broj od 1 do 10.

Skripta mora provjeriti je li argument unutar zadanog raspona, a ako nije, prekida s radom i ispisuje odgovarajuću poruku.

Ako je argument unutar zadanog raspona, skripta izrađuje novu datoteku `brojevi.txt` u koju će pohraniti niz koji sadrži sve brojeve od 1 do zadanog broja.

Zadatak 3

Napravite sljedeću strukturu direktorija:

```
.
├── OS3.md
├── OS3.pdf
└── screenshots
    ├── bash.png
    ├── nano.png
    └── vim.png
```

Napišite bash skriptu koja će proći kroz sve datoteke unutar direktorija `screenshots` i preimenovati ih na način da im doda prefix: `screenshot_N_` gdje je `N` redni broj datoteke (npr. `screenshot_1_bash.png`). Na kraju, skripta će ispisati sve preimenovane datoteke.

Zadatak 4

Napišite bash skriptu koja će primiti 1 argument: naziv direktorija. Skripta mora provjeriti nalazi li se direktorij u istom direktoriju kao i skripta. Ako se direktorij ne nalazi u istom direktoriju, prekida rad i ispisuje poruku da direktorij ne postoji.

Ako postoji, skripta mora proći kroz sve datoteke unutar direktorija i komprimirati ih naredbom `zip` u jednu zip datoteku. Ime zip datoteke neka bude `svi_zapisi.zip`. Ako korisnik proslijedi više od jednog argumenta, skripta mora prekinuti rad i ispisati poruku da je potrebno proslijediti samo jedan argument.