Uvod u skript programiranje

Nina Radojičić Matić

1. Pojam i uloga skript jezika

Skript jezici predstavljaju posebnu kategoriju programskih jezika namenjenih automatizaciji zadataka i upravljanju postojećim softverskim sistemima. Cilj je brz razvoj manjih programa sa ciljem automatizacije zadataka, obrade podataka i povezivanja različitih sistema. Dizajnirani su za automatizaciju rutinskih operacija i izvršavanje specifičnih zadataka, često unutar nekog okruženja ili aplikacije. Za razliku od tradicionalnih programskih jezika, skript jezici se najčešće izvršavaju direktno pomoću interpretera, bez potrebe za prethodnim kompajliranjem, što ih čini idealnim kada je potrebno brzo napisati i pokrenuti program bez složenog procesa razvoja.

Skript (engl. *script*) je niz komandi koje se izvršavaju u odgovarajućem izvršnom okruženju (engl. *run-time enviroment*).

Osnovna uloga skript jezika je da omoguće:

- automatizaciju rutinskih zadataka
- rad sa fajlovima i sistemom
- obradu tekstualnih podataka
- integraciju različitih softverskih komponenti

Skript jezici su posebno korisni u situacijama kada je potrebno brzo napisati i izvršiti program bez složenog procesa razvoja. Među najznačajnijim predstavnicima ove kategorije izdvajaju se Bash, Python, PowerShell i JavaScript - svaki sa svojim specifičnim domenom primene, od manipulacije fajlovima u operativnim sistemima pa sve do razvoja interaktivnih veb stranica.

2. Razlika između skriptnih i kompajliranih jezika

Programski jezici se mogu podeliti na skriptne (interpretirane) i kompajlirane.

Kod kompajliranih jezika (npr. C, C++):

- izvorni kod se prevodi (kompajlira) u izvršni program;

- izvršavanje je brže;
- razvojni proces je sporiji.

Kod skript jezika (npr. Python, Bash):

- kod se izvršava direktno pomoću interpretera;
- razvoj je brži i fleksibilniji;
- pogodni su za automatizaciju i prototipiranje.

Kratko rečeno, skript jezici omogućavaju brže pisanje i testiranje programa, dok kompajlirani jezici nude bolje performanse u složenim sistemima.

Primeri u dva jezika (Bash i Python) koja ćemo raditi u ovom semestru koji pokazuju kako se u skript jezicima često dosta može postići jednom komandom.

```
ls
```

```
print("Lista fajlova")
```

3. Karakteristike skript jezika

Glavne karakteristike skript jezika su:

- visok nivo apstrakcije;
- dinamičko tipiziranje;
- interpretacija (bez kompajliranja);
- jednostavna i čitljiva sintaksa;
- brz razvoj i kratko vreme implementacije.

Visok nivo apstrakcije

Skript jezici omogućavaju programeru da se fokusira na rešavanje problema, bez potrebe da upravlja detaljima kao što su memorija ili niskonivske operacije.

Na primer, čitanje fajla u skript jeziku može se svesti na nekoliko linija koda, dok bi u nižim jezicima zahtevalo znatno više rada.

Primer:

```
with open("data.txt") as f:
    print(f.read())
```

4. Pregled domena primene

Skript jezici se koriste u različitim oblastima, ali postoje i skript jezici opšte namene (primer Python).

4.1 Sistemska administracija

- upravljanje fajlovima i direktorijumima
- automatizacija sistemskih zadataka
- rad na serverima

```
mkdir test
cd test
```

4.2 Automatizacija

- automatizacija ponavljajućih zadataka
- obrada log fajlova
- pokretanje i povezivanje različitih alata

```
echo "Pokretanje"
```

4.3 Analiza podataka

- obrada CSV i JSON fajlova
- analiza velikih skupova podataka
- priprema podataka za dalje analize

```
import csv
```

5. Pregled najznačajnijih skript jezika

Bash

- koristi se u Linux/Unix okruženju
- namenjen radu u komandnoj liniji
- pogodan za rad sa fajlovima i sistemom

Primer:

```
echo "Zdravo"
```

Python

Python je interpretirani programski jezik visokog nivoa, popularan za analizu podataka i širok spektar primena

- Python je jezik opšte namene sa velikim brojem biblioteka.
- Koristi se za: - obradu podataka - automatizaciju - web razvoj - veštačku inteligenciju
- Prednosti: - jednostavna sintaksa - velika zajednica - platformska nezavisnost

  Primer:

  ```
  print("Hello")
  ```

PowerShell

- koristi se u Windows okruženju
- namenjen administraciji sistema
- omogućava rad sa procesima i servisima

  Primer:

  ```
  Get-Process
  ```

JavaScript

- koristi se u web razvoju
- izvršava se u browseru i na serveru
- omogućava interaktivnost web aplikacija

  Primer:

  ```
  console.log("Zdravo");
  ```

6. Kriterijumi za izbor skript jezika

Izbor jezika zavisi od više faktora:

Okruženje

- Linux → Bash
- Windows → PowerShell
- Web → JavaScript

Tip problema

Da li je cilj sistemska automatizacija, veb razvoj, analiza podataka…?

- rad sa sistemom → Bash
- obrada podataka → Python
- veb → JavaScript

Biblioteke i alati

Veći broj dostupnih biblioteka ubrzava razvoj.

Performanse

Za većinu skripti nisu ključne, ali su važne kod velikih sistema.

Održavanje i čitljivost

Čitljiv kod olakšava rad u timu.

Ne postoji najbolji jezik – izbor zavisi od problema, projektnih zahteva, ciljanog okruženja i postojećih resursa.

7. Primeri rada sa tekstualnim datotekama

Skript jezici se često koriste za obradu tekstualnih datoteka.

CSV (Comma-Separated Values)

- podaci su razdvojeni zarezima
- često se koriste u tabelarnim prikazima

Primer:

```
ime,prezime
Ana,Anic
```

TSV (Tab-Separated Values)

- podaci su razdvojeni tabulatorima

JSON (JavaScript Object Notation)

- strukturirani format podataka

- često se koristi u web aplikacijama

  Primer:

  ```
  {"ime": "Ana", "godine": 25}
  ```

XML (eXtensible Markup Language)

- hijerarhijski strukturiran format
- koristi se za razmenu podataka između sistema

  Primer:

  ```
  <osoba>
    <ime>Ana</ime>
  </osoba>
  ```

8. Zašto se Python koristi i za skriptovanje i za velike sisteme?

Python je jedinstven jer se koristi i za male skript programe i za velike softverske sisteme.

Razlozi:

- jednostavna i čitljiva sintaksa
- veliki broj biblioteka
- mogućnost skaliranja projekta
- nezavisnost od platforme

Program može početi kao jednostavna skripta, a kasnije prerasti u kompleksan sistem bez promene jezika.

9. Realna primena u industriji

Python se koristi u mnogim velikim sistemima kao što su:

- sistemi za preporuke sadržaja;
- veb aplikacije sa velikim brojem korisnika;
- analiza podataka i veštačka inteligencija;
- naučna istraživanja;

- cloud sistemi za skladištenje podataka.

Ovi primeri pokazuju da skript jezici nisu ograničeni na male zadatke, već imaju široku primenu u savremenim softverskim sistemima.

10. Kratak istorijski osvrt

Razvoj skript jezika započinje šezdesetih i sedamdesetih godina 20. veka, paralelno sa razvojem operativnih sistema, kada se javlja potreba za jednostavnijim načinima upravljanja komandnim okruženjem. Jedan od prvih značajnih primera jeste Unix shell, koji je omogućio korisnicima da kombinuju i automatizuju izvršavanje sistemskih komandi putem skripti.

Tokom osamdesetih i devedesetih godina dolazi do intenzivnijeg razvoja skript jezika, među kojima se izdvajaju Perl, Python i JavaScript. Ovi jezici uvode viši nivo apstrakcije, dinamičko tipiziranje i lakšu manipulaciju tekstom i podacima, čime postaju izuzetno pogodni za brzi razvoj aplikacija i obradu podataka.

Sa razvojem interneta, posebno krajem devedesetih godina, skript jezici dobijaju ključnu ulogu u razvoju veb aplikacija. JavaScript postaje standard za klijentsku stranu, dok jezici poput PHP omogućavaju generisanje dinamičkog sadržaja na serverskoj strani.

Danas skript jezici imaju široku primenu u različitim oblastima, uključujući automatizaciju sistemskih administrativnih zadataka, razvoj veb i mobilnih aplikacija, analizu podataka i veštačku inteligenciju. Njihova popularnost zasniva se na jednostavnosti upotrebe, fleksibilnosti i mogućnosti brzog razvoja softverskih rešenja.

Zaključak

Skript jezici predstavljaju važan alat u savremenom programiranju. Oni omogućavaju brz razvoj, jednostavnu automatizaciju i efikasan rad sa podacima. Razumevanje njihove uloge i pravilna primena omogućava rešavanje širokog spektra problema, od jednostavnih skripti do složenih softverskih sistema. Izbor jezika zavisi od problema koji se rešava, a ne obrnuto.

Skript jezici za rad u komandnoj liniji

Komandna linija i okruženje za rad

Komandna linija (engl. *Command Line Interface* - CLI) je tekstualni interfejs za interakciju sa operativnim sistemom. Umesto grafičkog korisničkog interfejsa (GUI), korisnici unose tekstualne komande za izvršavanje zadataka. Okruženje za rad u komandnoj liniji, poznato i kao terminal ili konzola, pruža pristup shell-u.

Uloga shell-a

Shell interpretira komande koje korisnik unese i izvršava ih. Shell jezici, kao što su Bash, csh, ksh i PowerShell, omogućavaju komunikaciju sa operativnim sistemom i manipulaciju dokumentima. Oni su ključni za automatizaciju rutinskih zadataka, upravljanje sistemom i izvršavanje programa.

Pregled alata i tipičnih scenarija upotrebe

- **Bash i PowerShell**: Ovi jezici se koriste za sistemsku automatizaciju, upravljanje fajlovima i direktorijumima, konfigurisanje sistema, izvršavanje programa i skripti, kao i za kreiranje složenijih skripti za automatizaciju zadataka na serverima.
- **Python**: Iako nije primarno shell jezik, Python se često koristi u komandnoj liniji za izvršavanje skripti za automatizaciju, analizu podataka, obradu teksta, mrežne operacije i interakciju sa API-jima. Njegova snaga leži u bogatoj biblioteci i čitljivoj sintaksi.

Regularni izrazi

Primer za razmišljanje: Zamislite da imate tekstualni fajl sa hiljadama redova podataka i zadatak da iz njega izvučete sve email adrese, telefonske brojeve ili datume - bez ikakvog alata osim sopstvenog koda. Upravo tu na scenu stupaju regularni izrazi.

U savremenom programiranju, obrada tekstualnih podataka predstavlja jedan od najčešćih i najzahtevnijih zadataka sa kojima se programeri svakodnevno susreću. Bez obzira na to da li je reč o validaciji korisničkog unosa, analizi log fajlova, ekstrakciji strukturiranih podataka iz nestrukturiranog teksta ili transformaciji velikih skupova podataka, potreba za preciznim i efikasnim mehanizmom pretrage i manipulacije tekstom javlja se u gotovo svakom domenu softverskog razvoja.

Regularni izrazi (*engl. regular expressions*, skraćeno *regex* ili *regexp*) predstavljaju formalni mehanizam za opisivanje obrazaca u tekstu, zasnovan na teoriji formalnih jezika i konačnih automata. Reč je o specifičnoj notaciji pomoću koje je moguće precizno definisati skup niski koje odgovaraju određenom uzorku, a potom ih pronalaziti, izdvajati, validirati ili zamenjivati unutar tekstualnih podataka.

Jedna od ključnih karakteristika regularnih izraza jeste njihova jezička neutralnost - jednom savladana sintaksa primenjiva je u velikom broju programskih jezika i okruženja, uključujući Python i Bash, koji su u fokusu ovog kursa.

Prošireni regularni izrazi (engl. *Extended Regular Expressions* - ERE) pojavili su se sa razvojem Unix alata za obradu teksta tokom 1970-ih godina, kao praktično proširenje prvobitnih regularnih izraza korišćenih u editorima poput `ed` i kasnije `grep`.

Istorijski razvoj ukratko:

- 1960. godine teorijsku osnovu regularnih izraza postavio je Stephen Cole Kleene kroz formalne jezike i automata teoriju.
- Rani Unix (1970-te) – alat `grep` koristio je jednostavnije osnovne regularne izraze (engl. *Basic Regular Expressions* - BRE).
- Kasne 1970-te / rane 1980-te – pojavljuju se proširenja sa operatorima `+`, `?`, `|`, `()` radi lakšeg rada u alatima kao što su `egrep` i `awk`.
- 1988. godina – POSIX standard formalno definiše dve varijante:
    - BRE (Basic Regular Expressions)
    - ERE (Extended Regular Expressions)

Kod običnih (engl. *Basic Regular Expressions* -BRE) neki specijalni operatori moraju da se pišu sa znakom `\` ispred (“escape”-uju) da bi imali posebno značenje. Kod ERE isti ti operatori se koriste direktno, bez `\`, pa je zapis pregledniji i jednostavniji. U nastavku ćemo podrazumevati ERE kada govorimo o regularnim izrazima.

1. Formiranje regularnih izraza

Regularni izraz predstavlja nisku karaktera koja opisuje obrazac pretrage. Njegova osnovna struktura zasniva se na kombinaciji literalnih karaktera, specijalnih karaktera i kvantifikatora, koji zajedno definišu uzorak koji se traži u tekstu.

1.1 Literalni karakteri

Najjednostavniji oblik regularnog izraza čine literalni karakteri - karakteri koji doslovno odgovaraju samima sebi u tekstu koji se pretražuje. Na primer, regularni izraz python pronalaziće svako pojavljivanje te reči u tekstu, uzimajući u obzir tačno zadati redosled karaktera.

1.2 Specijalni karakteri (metakarakteri)

Pored literalnih karaktera, regularni izrazi koriste skup specijalnih karaktera - metakaraktera - koji imaju posebno značenje unutar obrasca. Najvažniji metakarakteri prikazani su u sledećoj tabeli:

Metakarakter	Značenje	Primer
.	Bilo koji karakter osim novog reda	p.t odgovara pat, pot, pit
^	Početak niske	^Hello odgovara samo ako niska počinje sa Hello
$	Kraj niske	world$ odgovara samo ako niska završava sa world
\d	Cifra (0–9)	\d\d\d odgovara 123, 456

\w	Alfanumerički karakter ili podvlaka	\w+ odgovara ime, test_1
\s	Beli karakter (razmak, tabulator)	\s odgovara razmaku između reči
\	Da se metakaraket tretira kao literalni	\. odgovara doslovnoj tački
\b	Granica reči	\bda\b odgovara reči da, ali neće ga prepoznati u reči dan

1.3 Klase karaktera

Klase karaktera omogućavaju definisanje skupa karaktera od kojih jedan mora biti prisutan na određenoj poziciji u tekstu. Zapisuju se unutar uglastih zagrada [ ].

Na primer, regularni izraz [aeiou] odgovara bilo kom samoglasniku, dok izraz [a-z] odgovara bilo kom malom slovu engleske abecede. Negacija klase postiže se simbolom ^ unutar zagrada - [^0-9] odgovara bilo kom karakteru koji nije cifra.

1.4 Kvantifikatori

Kvantifikatori određuju koliko puta određeni karakter ili grupa karaktera može ili mora biti prisutna u obrascu. Osnovni kvantifikatori su sledeći:

Kvantifikator	Značenje	Primer
*	Nula ili više pojavljivanja	ab*c odgovara ac, abc, abbc
+	Jedno ili više pojavljivanja	ab+c odgovara abc, abbc, ali ne i ac
?	Nula ili jedno pojavljivanje	colou?r odgovara color i colour

{n}	Tačno n pojavljivanja	\d{4} odgovara tačno četiri cifre
{n,m}	Između n i m pojavljivanja	\d{2,4} odgovara od dve do četiri cifre

1.5 Grupisanje i alternacija

Grupisanje se postiže upotrebom okruglih zagrada ( ) i omogućava primenu kvantifikatora na čitavu grupu karaktera, kao i izdvajanje delova pronađenog obrasca. Na primer, izraz (ab)+ odgovara jednom ili više pojavljivanja niske ab.

Alternacija se postiže operatorom | i funkcioniše kao logički operator ILI. Izraz cat|dog pronalaziće svako pojavljivanje reči cat ili reči dog u tekstu.

1.6 Složeni primer - validacija email adrese

Kao ilustraciju kombinovanja navedenih elemenata, može se razmotriti regularni izraz za osnovnu validaciju email adrese:

^[\w.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z]{2,}$

Analiza ovog izraza otkriva sledeću strukturu: izraz ^[\w.+-]+ označava da niska mora počinjati jednim ili više alfanumeričkih karaktera, tački, znakova plus ili minus; simbol @ je literalni karakter koji mora biti prisutan; [a-zA-Z0-9-]+ opisuje naziv domena sastavljen od slova, cifara ili crtica; \. predstavlja doslovnu tačku koja razdvaja naziv domene od ekstenzije; dok [a-zA-Z]{2,}$ zahteva da niska završava ekstenzijom od najmanje dva slova.

Ovaj primer jasno pokazuje kako se kombinovanjem relativno malog broja gradivnih elemenata mogu formirati izrazi sposobni da precizno opišu složene tekstualne obrasce.

2. Praktična primena regularnih izraza u skript jezicima

Regularni izrazi su, zahvaljujući svojoj jezičkoj neutralnosti, integrisani u veliki broj programskih jezika. Ipak, svaki jezik poseduje specifičan način rada sa regularnim izrazima, pa je važno razumeti kako se opšta sintaksa primenjuje u konkretnom okruženju.

2.1 Regularni izrazi u Python-u

Python pruža podršku za regularne izraze kroz standardni modul re, koji se uvozi na početku skripte. Modul nudi nekoliko ključnih funkcija, od kojih su najčešće korišćene prikazane u sledećoj tabeli:

Funkcija	Opis
re.match()	Proverava da li obrazac odgovara početku niske
re.search()	Pretražuje celu nisku i vraća prvo podudaranje
re.findall()	Vraća listu svih podudaranja u niski
re.sub()	Zamenjuje sva podudaranja zadatim tekstom
re.compile()	Kompajlira obrazac radi višekratne upotrebe

Primer 1 - Pronalaženje svih cifara u tekstu:

```
import re


tekst = "Imamo 1 čas predavanja i 2 časa vežbi nedeljno, tokom narednih 13 nedelja."

rezultat = re.findall(r'\d+', tekst)

print(rezultat)  # ['1', '2', '13']
```

U ovom primeru, obrazac \d+ pronalazi sve nizove od jedne ili više cifara unutar zadatog teksta. Prefiks r ispred navodnika označava tzv. *raw string*, čime se sprečava da Python interpretira karakter \ pre nego što ga prosledi modulu re.

Primer 2 - Validacija email adrese:

```
import re

def validiraj_email(email):

    obrazac = r'^[\w.+-]+@([a-zA-Z0-9-]+\.)+[a-zA-Z]{2,}$'
```

```
    if re.match(obrazac, email):
        return "Validna adresa"
    return "Nevalidna adresa"

print(validiraj_email("nina@matf.bg.ac.rs"))  # Validna adresa
print(validiraj_email("nina@"))               # Nevalidna adresa
```

Primer 3 - Zamena teksta pomoću re.sub():

```
import re
tekst = "Datum isporuke je 28-03-2026."
novi_tekst = re.sub(r'(\d{2})-(\d{2})-(\d{4})', r'\3/\2/\1', tekst)
print(novi_tekst)  # Datum isporuke je 2026/03/28.
```

U ovom primeru demonstrirana je upotreba grupa - delovi obrasca unutar zagrada ( ) pamte se i mogu se koristiti u zameni putem referenci \1, \2, \3, koje odgovaraju prvoj, drugoj i trećoj grupi redom.

2.2 Regularni izrazi u Bash-u

U Bash okruženju, regularni izrazi se najčešće koriste u kombinaciji sa alatima poput grep, sed i awk, koji su standardni deo Unix/Linux sistema.

Primer 1 - Pretraga fajla pomoću grep:

```
grep -E '\d{3}-\d{4}' kontakti.txt
```

Opcija -E aktivira proširene regularne izraze (*Extended Regular Expressions -ERE*), dok navedeni obrazac pronalazi sve redove koji sadrže telefonske brojeve u formatu XXX-XXXX. Ranije se za `grep -E` koristio `egrep`.

Primer 2 - Zamena teksta pomoću sed:

```
sed -E 's/([0-9]{2})\.([0-9]{2})\.([0-9]{4})/\3-\2-\1/g' fajl.txt
```

Komanda sed sa oznakom s vrši zamenu - u ovom slučaju, datumi zapisani u formatu DD.MM.YYYY konvertuju se u format YYYY-MM-DD. Oznaka g na kraju označava globalnu zamenu, tj. zamenu svih podudaranja u svakom redu, a ne samo prvog.

Primer 3 - Filtriranje log fajla:

```
grep -E '^ERROR|^WARNING' aplikacija.log
```

Navedeni obrazac izdvaja sve redove iz log fajla koji počinju rečju ERROR ili rečju WARNING, što je čest zadatak u analizi sistemskih i aplikativnih dnevnika.

2.3 Regularni izrazi u JavaScript-u (za studente koji žele da saznaju više)

U JavaScript-u, regularni izrazi su ugrađeni tip podataka i mogu se definisati na dva načina - literalnom notacijom između kosih crtica, ili upotrebom konstruktora RegExp.

```
// Literalna notacija

const obrazac = /^\d{4}-\d{2}-\d{2}$/;


// Konstruktor

const obrazac2 = new RegExp('^\\d{4}-\\d{2}-\\d{2}$');
```

Metode koje se najčešće koriste za rad sa regularnim izrazima u JavaScript-u su test(), match(), replace() i split().

Primer 1 - Validacija formata datuma:

```
const obrazac = /^\d{4}-\d{2}-\d{2}$/;


console.log(obrazac.test("2026-03-28"));  // true
```

```
console.log(obrazac.test("28.03.2026"));  // false
```

Primer 2 - Izdvajanje svih URL adresa iz teksta:

```
const tekst = "Posetite https://skript-programiranje-i.matf.bg.ac.rs/ ili https://petlja.org/ za više informacija.";

const obrazac = /https?:\/\/[\w.-]+\.[a-z]{2,}/g;

console.log(tekst.match(obrazac));

// ['https://skript-programiranje-i.matf.bg.ac.rs/', 'https://petlja.org/']
```

U ovom primeru, https? označava da je karakter s opcion, čime obrazac obuhvata i http i https protokol. Kose crte u URL adresi moraju biti escapovane kao \/ jer kosa crta u JavaScript-u označava kraj literalnog regularnog izraza.

Primer 3 - Zamena teksta metodom replace():

```
const tekst = "Ime: Marko Marković, Tel: 064-123-4567";

const anonimizan = tekst.replace(/\d{3}-\d{3}-\d{4}/, "***-***-****");

console.log(anonimizan);  // Ime: Marko Marković, Tel: ***-***-****
```

2.4 Regularni izrazi u PowerShell-u (za studente koji žele da saznaju više)

PowerShell nudi ugrađenu podršku za regularne izraze kroz operator -match, -replace i -split, kao i kroz cmdlet Select-String, koji je funkcionalni ekvivalent alatu grep u Unix okruženju.

Primer 1 - Operator -match:

```
$email = "student@univerzitet.edu.rs"

if ($email -match '^[\w.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z]{2,}$') {

    Write-Output "Validna email adresa"

}
```

Primer 2 - Pretraga fajlova pomoću Select-String:

```
Select-String -Path "*.log" -Pattern "^ERROR" | Select-Object LineNumber, Line
```

Ovom komandom pretražuju se svi log fajlovi u trenutnom direktorijumu, a kao rezultat se ispisuju brojevi redova i sadržaj svakog reda koji počinje rečju ERROR.

Primer 3 - Operator -replace:

```
$tekst = "Telefon: 011-123-4567"

$rezultat = $tekst -replace '\d{3}-\d{3}-\d{4}', 'XXX-XXX-XXXX'

Write-Output $rezultat  # Telefon: XXX-XXX-XXXX
```

2.5 Komparativni pregled

Iako sintaksa regularnih izraza ostaje u velikoj meri konzistentna između različitih jezika, postoje određene razlike u načinu njihovog pozivanja i u naprednim mogućnostima koje svaki jezik nudi. U sledećoj tabeli dat je sažet pregled osnovnih operacija u obrađenim jezicima:

Operacija	Python	Bash	JavaScript	PowerShell
Pretraga	re.search()	grep -E	.match()	-match / Select-String
Zamena	re.sub()	sed -E 's/.../.../'	.replace()	-replace
Validacija	re.match()	[[=~]]	.test()	-match
Sve instance	re.findall()	grep -oE	.match(/g)	[regex]::Matches()

Poznavanje ovih razlika omogućava programeru da efikasno primeni stečeno znanje o regularnim izrazima bez obzira na radno okruženje ili jezik koji koristi.

1.Uvod u Bash

1. Uvod

U savremenom softverskom razvoju i sistemskoj administraciji, sposobnost efikasnog rada sa komandnom linijom predstavlja jednu od temeljnih kompetencija svakog programera i sistem administratora. Bash (*Bourne Again SHell)* zauzima centralno mesto u ovom domenu kao najrasprostranjeniji komandni jezik u Unix/Linux ekosistemu.

Razvoj komandnih interpretera (shell-ova) usko je povezan sa istorijom Unix operativnog sistema iz 1970-ih godina. Prvi značajan shell bio je Thompson shell, a njegov naslednik bio je Bourne Shell (*sh*), koji je Stephen Bourne razvio u okviru kompanije AT&T. Bourne Shell je postavio osnovu za mnoge koncepte koji se i danas koriste, uključujući skriptovanje, rad sa promenljivama i kontrolne strukture. Tokom vremena razvijeni su i drugi shell-ovi, poput C Shell i KornShell, koji su uvodili dodatne funkcionalnosti i unapređenja u radu sa komandnom linijom.

Bash je razvio Brian Fox 1989. godine pod okriljem GNU projekta, kao slobodan i prošireni naslednik originalnog Bourne Shell-a. Njegov razvoj je kasnije nastavljen u okviru Free Software Foundation, sa ciljem da se obezbedi potpuno funkcionalna i slobodna alternativa postojećim Unix alatima. Bash je zadržao kompatibilnost sa sh, ali je uveo i brojne napredne mogućnosti, kao što su komandna istorija, automatsko dovršavanje (tab completion), unapređeno upravljanje procesima i bogatiji skriptni jezik.

Od tada je Bash postao standardni interaktivni shell na gotovo svim Linux distribucijama, kao i na macOS sistemima do verzije Catalina, dok je na novijim verzijama podrazumevani shell Zsh. Ipak, Bash ostaje široko dostupan i aktivno korišćen, uključujući i Windows platformu putem Windows Subsystem for Linux. Zahvaljujući ovoj rasprostranjenosti, znanje Bash komandi direktno je prenosivo između različitih operativnih sistema i radnih okruženja.

Dakle, shell je program koji interpretira komande koje korisnik unosi i prosleđuje ih jezgru operativnog sistema na izvršavanje. Predstavlja sloj apstrakcije između korisnika i hardvera - korisniku pruža tekstualni interfejs kroz koji može pokretati programe, upravljati procesima i manipulisati fajl sistemom, bez potrebe za direktnom interakcijom sa jezgrom.

Postoji više vrsta shell programa, od kojih su najznačajniji:

- **sh** (Bourne Shell) - originalni Unix shell, osnova za sve moderne shell-ove;
- **bash** (Bourne Again Shell) - proširena verzija sh, standard na Linux sistemima;
- **zsh** (Z Shell) - popularna alternativa sa naprednim mogućnostima, podrazumevani shell na macOS od verzije Catalina;
- **fish** (Friendly Interactive Shell) - fokusiran na interaktivnost i upotrebljivost;
- **dash** - minimalistički shell kompatibilan sa POSIX standardom, često korišćen za izvršavanje skripti.

Terminal (ili emulator terminala) je aplikacija koja pruža grafički interfejs za interakciju sa shellom. Važno je razumeti razliku: terminal je prozor, a shell je program koji se u njemu izvršava.

---

2. Zašto učiti Bash?

Bash nije samo alat za unos komandi, već je reč o potpunom skript jeziku koji omogućava automatizaciju složenih zadataka. Njegova vrednost ogleda se u nekoliko ključnih oblasti.

Automatizacija rutinskih zadataka predstavlja primarnu snagu Bash-a. Zadaci poput pravljenja rezervnih kopija, periodičnog čišćenja privremenih fajlova, generisanja izveštaja ili pokretanja servisa mogu se opisati Bash skriptom i potom izvršavati automatski, bez intervencije korisnika.

Rad sa fajlovima i sistemom obuhvata navigaciju fajl sistemom, pretragu i filtriranje fajlova, kao i manipulaciju njihovim sadržajem - operacije koje su sastavni deo svakodnevnog rada u Linux okruženju.

Integracija alata jedan je od distinktivnih aspekata Bash-a: komande se mogu ulančavati pomoću pipe operatora (|), čime se izlaz jedne komande direktno prosleđuje kao ulaz sledeće, što omogućava izgradnju složenih tokova obrade podataka iz jednostavnih gradivnih blokova.

Serverska administracija nezamisliva je bez poznavanja Bash-a. Upravljanje procesima, konfigurisanje servisa, analiza log fajlova i nadgledanje sistema standardno se obavljaju putem komandne linije.

Konačno, poznavanje Bash-a predstavlja **osnovu za napredne tehnologije** poput DevOps praksi, CI/CD pipeline-ova, kontejnerizacije (Docker, Kubernetes) i cloud računarstva, gde su skripte nezaobilazan element infrastrukture.

---

3. Osnovna sintaksa Bash komandi

Osnovni koncepti rada u terminalu

Prompt je znak koji označava da je shell spreman za prijem komandi. Tipični oblik prompta:

korisnik@hostname:~/putanja$

Znak $ označava standardnog korisnika, dok # označava superkorisnika (root). Prompt je moguće konfigurisati kroz promenljivu PS1.

Svaka Bash komanda prati jedinstvenu strukturu koja se sastoji od tri elementa:

komanda [opcije] [argumenti]

Komanda definiše operaciju koja se izvršava.

Opcije (zastave) modifikuju ponašanje komande i najčešće počinju crticom (-), npr. -l za dugačak ispis ili -r za rekurzivnu operaciju. Kratke opcije pišu se sa jednom crticom (-l), duge sa dve (--long)

Argumenti su podaci na kojima komanda deluje, kao što su putanje do fajlova ili niske karaktera.

Primeri:

Komanda	Opcija	Argument	Efekat
ls	-l	/home	Detaljan listing direktorijuma
cp	-r	dir1 dir2	Rekurzivno kopiranje direktorijuma
grep	-i	'bash' fajl.txt	Pretraga bez razlikovanja veličine slova
rm	-f	fajl.txt	Brisanje bez zahteva za potvrdu

Opcije se mogu kombinovati: ls -la ekvivalentno je sa ls -l -a i prikazuje detaljan listing uključujući skrivene fajlove.

---

4. Navigacija fajl sistemom

Fajl sistem u Linux okruženju organizovan je kao hijerarhijsko stablo koje počinje od korenskog direktorijuma /. Poznavanje komandi za navigaciju preduslov je za svaki dalji rad u Bash okruženju.

pwd (*Print Working Directory*) ispisuje apsolutnu putanju trenutnog direktorijuma. Ova komanda je korisna svaki put kada je potrebno utvrditi gde se korisnik trenutno nalazi u hijerarhiji.

pwd

/home/student/projekti

ls prikazuje sadržaj direktorijuma. Najčešće korišćene kombinacije opcija su -l (detaljan prikaz sa pravima pristupa, vlasnikom i veličinom) i -a (prikazuje i skrivene fajlove čije ime počinje tačkom).

ls -la /home/korisnik

Opcija	Opis
-l	Detaljan prikaz (dozvole, vlasnik, veličina, datum)
-a	Prikazuje skrivene fajlove (čije ime počinje sa .)
-h	Čitljive veličine (KB, MB, GB) uz kombinaciju sa -l
-t	Sortiranje po datumu izmene (najnoviji prvi)
-r	Obrnuti redosled sortiranja
-R	Rekurzivno listanje poddirektorijuma

-S Sortiranje po veličini

--color Bojeno prikazivanje prema tipu fajla

ls -lah /etc # Detaljan prikaz sa skrivenim fajlovima i čitljivim veličinama

ls -lt ~/dokumenti # Sortirano po datumu, najnoviji prvi

cd (*Change Directory*) menja trenutni direktorijum. Posebne vrednosti argumenta su:

.. (roditeljski direktorijum),

~ (home direktorijum trenutnog korisnika) i

- (prethodni direktorijum).

cd /var/log # Apsolutna putanja

cd /etc # Apsolutna putanja

cd dokumenti # Relativna putanja

cd .. # Jedan nivo naviše

cd ~ # Home direktorijum

cd - # Prethodni direktorijum

find - Pretražuje fajl sistem prema zadatim kriterijumima:

Opcija	Opis	Primer
-name	Pretraga po nazivu (case-sensitive)	find . -name "*.txt"
-iname	Pretraga po nazivu (case-insensitive)	find . -iname "readme*"
-type	Pretraga po tipu (f=fajl, d=dir, l=link)	find / -type d -name "bin"
-size	Pretraga po veličini	find . -size +10M
-mtime	Pretraga po datumu izmene (u danima)	find . -mtime -7
-exec	Izvršava komandu nad pronađenim fajlom	find . -name "*.log" -exec rm {} \;
-maxdepth	Ograničava dubinu pretrage	find . -maxdepth 2 -name "*.sh"

find /home -name "*.py" -size +1k -mtime -30

Pronalazi Python fajlove veće od 1KB, izmenjene u poslednjih 30 dana

5. Rad sa fajlovima i direktorijumima

5.1 Kreiranje i upravljanje

mkdir kreira novi direktorijum, a sa opcijom -p kreira čitavu putanju rekurzivno:

touch novi_fajl.txt

mkdir projekat

mkdir -p projekat/src/utils # Kreira celu hijerarhiju

mkdir -p projekat/src/moduli # Kreira celu hijerarhiju (-p: parents)

mkdir -m 755 javni_dir # Sa zadatim dozvolama

touch - Kreira prazan fajl ili ažurira vremensku oznaku tj. vreme pristupa postojećeg fajla.

touch skript.sh

touch -t 202503280900 fajl.txt # Postavlja specifično vreme

cp kopira fajl ili direktorijum (sa -r),

Opcija	Opis
-r	Rekurzivno kopiranje direktorijuma
-p	Čuva atribute (dozvole, vlasnik, vreme)
-i	Pita pre prepisivanja

-u Kopira samo ako je izvor noviji od odredišta

-v Prikazuje šta se kopira (verbose)

cp -rp /etc/nginx /backup/nginx_$(date +%Y%m%d)

Ova komanda pravi backup (kopiju) celog Nginx konfiguracionog foldera, i to u novi folder koji u nazivu ima današnji datum.

Objašnjenje delova:

cp → komanda za kopiranje

-r → kopira direktorijum rekurzivno (sve fajlove i podfoldere)

-p → čuva originalne dozvole, vlasništvo i datume fajlova

/etc/nginx → izvor (gde se nalazi Nginx konfiguracija)

/backup/nginx_$(date +%Y%m%d) → destinacija (gde se kopira)

Šta znači $(date +%Y%m%d)?

Ubacuje današnji datum u ime foldera.

Format:

%Y → godina (npr. 2026)

%m → mesec (04)

%d → dan (06)

Na primer, ako je danas 6. april 2026. godine, rezultat će biti:

/backup/nginx_20260406

Konačan rezultat:

Dobijamo folder:

/backup/nginx_20260406/

u kome je kompletna kopija /etc/nginx

mv premešta ili preimeuje fajl.

mv staro_ime.txt novo_ime.txt

mv *.log /var/log/arhiva/

mv -i fajl.txt /home/student/ # Sa pitanjem pre prepisivanja

rm briše fajlove, a sa -r i čitave direktorijume - treba napomenuti da Linux nema kantu za otpatke u komandnoj liniji, pa ova operacija nije reverzibilna.

Opcija	Opis
-r	Rekurzivno brisanje direktorijuma
-f	Prisilno brisanje bez potvrde
-i	Pita pre svakog brisanja
-v	Prikazuje šta se briše

> **Upozorenje:** rm -rf je nepovratna operacija. Poseban oprez potreban je kada se koriste promenljive u putanjama jer greška može uzrokovati brisanje sistemskih fajlova.

ln komanda služi za kreiranje **linkova ka fajlovima** - možeš napraviti **tvrdi (hard)** ili **simbolički (soft) link**.

ln -s /putanja/do/originala naziv_linka # Simbolički link kao prečica ka fajlu (kao shortcut u Windows-u), ako obrišemo original link postaje nevažeći

ln original.txt tvrdi_link.txt # Tvrdi link (drugo ime za isti fajl, oba fajla dele isti sadržaj, nema razlike između “originala” i “kopije”

5.2 Pregled sadržaja

cat ispisuje čitav sadržaj fajla, pogodno za kraće fajlove. Za duže fajlove koriste se **less** (interaktivni preglednik) ili **more**. **head** i **tail** prikazuju prvih odnosno poslednjih n redova fajla - tail -f je posebno koristan za praćenje log fajlova u realnom vremenu.

cat konfiguracija.txt

less veliki_log.txt

head -n 20 izvestaj.txt

tail -f /var/log/syslog

wc broji redove (-l), reči (-w) ili karaktere (-c) u fajlu.

find pretražuje fajl sistem prema zadatim kriterijumima:

find /home -name "*.sh" -type f # Svi shell skripti u /home

find . -mtime -7 # Fajlovi izmenjeni u poslednjih 7 dana

6. Promenljive i specijalni parametri

6.1 Definisanje i upotreba promenljivih

Promenljive se u Bash-u definišu bez ključnih reči, a između naziva, znaka jednakosti i vrednosti ne sme biti razmaka:

ime="Marko"

broj=42

pi=3.14

U Bash-u nema striktnih tipova.

Vrednost promenljive dohvata se prefiksom $. Preporučuje se korišćenje vitičastih zagrada ${} kada je naziv promenljive praćen dodatnim karakterima:

echo "Zdravo, $ime!"

echo "Vrednost je: ${broj}kg"

Razlika između " " i ' ' u Bash-u

1. Dupli navodnici " " → dozvoljavaju interpretaciju

U njima Bash **obrađuje promenljive i komande**.

naziv="Skript programiranje"

echo "Predmet: $naziv"

Rezultat:

Predmet: Skript programiranje

Dakle, $naziv se zamenjuje svojom vrednošću

2. Jednostruki navodnici ' ' → ništa se ne menja

Sve unutar njih se tretira kao **običan tekst**.

naziv="Skript programiranje"

echo 'Predmet: $naziv'

Rezultat:

Predmet: $naziv

Komandna supstitucija omogućava da se rezultat izvršene komande dodeli promenljivoj. Moderna sintaksa koristi $(komanda), a starija `komanda`:

datum=$(date +"%d.%m.%Y")

broj_fajlova=$(ls | wc -l)

echo "Danas je $datum, a ima $broj_fajlova fajlova."

6.2 Specijalni parametri

Promenljive okruženja (engl. *environment variables*) čuvaju sistemske informacije i konfiguraciju.

Parametar	Značenje
$0	Ime skripte
$1, $2, ...	Argumenti prosleđeni skripti po redosledu
$#	Ukupan broj argumenata
$@	Svi argumenti kao zasebne niske
$?	Exit status prethodne komande (0 = uspeh)

$$	PID - identifikator trenutnog shell procesa tj. trenutne skripte
$USER	Korisničko ime
$HOME	Putanja home direktorijuma
$PATH	Lista direktorijuma za pretragu izvršnih programa
$SHELL	Putanja do aktivnog shella
$PWD	Putanja trenutnog radnog direktorijuma

Prikazivanje vrednosti promenljive:

```
echo $HOME        # /home/student
echo $PATH        # /usr/local/bin:/usr/bin:/bin
printenv          # Prikazuje sve promenljive okruženja
```

Istorija komandi - Bash čuva istoriju izvršenih komandi u fajlu ~/.bash_history:

```
history           # Prikazuje istoriju komandi
history 20        # Poslednjih 20 komandi
!!                # Ponovi poslednju komandu
!ls               # Ponovi poslednju komandu koja počinje sa "ls"
Ctrl+R            # Interaktivna pretraga istorije
```

6.3 Korisne prečice u terminalu

Dopunjavanje tabulatorom (engl. *tab completion*) - pritiskanjem Tab na tastaturi Bash automatski dopunjava nazive komandi, fajlova i direktorijuma, što ubrzava rad i smanjuje greške.

Korisni prečice u terminalu:

Prečica	Funkcija
Ctrl+C	Prekida trenutni proces
Ctrl+Z	Pauzira trenutni proces (šalje u pozadinu)
Ctrl+D	Zatvara terminal / signalizira kraj ulaza
Ctrl+L	Čisti ekran (ekvivalentno komandi clear)
Ctrl+A	Pomera kursor na početak linije
Ctrl+E	Pomera kursor na kraj linije
Ctrl+U	Briše sadržaj linije do kursora

6.4 Dozvole i vlasništvo

Svaki fajl ima skup dozvola za vlasnika, grupu i ostale korisnike. Struktura dozvola u ls -l prikazu:

-rwxr-xr-- 1 student studenti 1024 Mar 28 14:30 skript.sh

Prva tri karaktera (rwx) su dozvole vlasnika, naredna tri (r-x) dozvole grupe, poslednja tri (r--) dozvole ostalih.

chmod - Menja dozvole:

```
chmod +x skript.sh          # Dodaje dozvolu za izvršavanje svima

chmod u+w fajl.txt          # Dodaje pravo pisanja vlasniku

chmod 755 skript.sh         # Oktalni zapis: vlasnik=rwx, grupa=r-x, ostali=r-x

chmod 644 dokument.txt      # Vlasnik=rw-, grupa=r--, ostali=r--

chmod -R 755 direktorijum/  # Rekurzivno
```

chown - Menja vlasnika fajla:

Osnovna sintaksa:

```
chown [opcije] korisnik[:grupa] fajl
```

Primer

```
chown student fajl.txt
```

Sada fajl.txt pripada korisniku student

```
chown student:studenti fajl.txt
```

Vlasnik je student, a grupa studenti

```
chown -R www-data /var/www/html
```

`-R` = rekurzivno menja vlasnika za:

- folder
- sve fajlove
- sve podfoldere

7. Obrada podataka u komandnoj liniji

Bash se može vrlo efikasno koristiti za obradu podataka direktno u komandnoj liniji, bez potrebe za pisanjem složenih programa. Osnovna ideja je kombinovanje malih, specijalizovanih alata kroz tzv. *pipeline* (tok podataka između komandi), gde se izlaz jedne komande prosleđuje kao ulaz drugoj pomoću operatora `|`.

7.1 Komande za obradu teksta

cat - Prikazuje sadržaj fajla:

```
cat fajl.txt

cat -n fajl.txt            # Sa brojevima redova

cat -A fajl.txt            # Prikazuje specijalne karaktere

cat fajl1.txt fajl2.txt > spojen.txt  # Spajanje fajlova
```

head i **tail** - Prikazuju početak ili kraj fajla:

```
head -n 20 fajl.txt        # Prvih 20 redova (podrazumevano: 10)

tail -n 50 log.txt         # Poslednjih 50 redova

tail -f /var/log/syslog    # Praćenje fajla u realnom vremenu

tail -F aplikacija.log     # Kao -f, ali prati i rotaciju fajla
```

less - služi za pregled sadržaja fajla po stranicama, bez učitavanja celog fajla odjednom u ekran.

```
less dugacak_fajl.txt

# Navigacija: strelice, Space  sledeća stranica, b  prethodna stranica, PgUp/PgDn, /reč (pretraga), q (izlaz)
```

grep - Pretražuje tekst na osnovu obrasca:

Opcija	Opis
-i	Pretraga bez razlikovanja malih/velikih slova
-r	Rekurzivna pretraga po direktorijumu
-n	Prikazuje broj linije
-v	Invertuje rezultat (redovi koji NE odgovaraju)
-c	Broji podudaranja umesto da ih prikazuje
-l	Prikazuje samo nazive fajlova sa podudaranjem
-E	Prošireni regularni izrazi
-A n	Prikazuje n redova posle podudaranja
-B n	Prikazuje n redova pre podudaranja
-o	Prikazuje samo deo koji se poklapa, ne ceo red

grep -rn "TODO" ./src/ # Rekurzivna pretraga sa brojevima linija

grep -E "^ERROR|^WARNING" app.log # Prošireni regex - redovi koji počinju sa ERROR ili WARNING

```
grep -v "^#" /etc/hosts              # Redovi koji nisu komentari
```

sed - Stream editor za transformaciju teksta:

```
sed 's/staro/novo/g' fajl.txt        # Zamena svih pojavljivanja
sed -i 's/staro/novo/g' fajl.txt     # Izmena direktno u fajlu (-i: in-place)
sed '1,5d' fajl.txt                  # Brisanje redova 1-5
sed -n '10,20p' fajl.txt             # Prikazuje samo redove 10-20
sed '/^$/d' fajl.txt                 # Uklanja prazne redove
sed 's/[[:space:]]*$//' fajl.txt     # Uklanja trailing whitespace
```

awk - Moćan alat za obradu strukturiranih tekstualnih podataka:

```
awk '{print $1}' fajl.txt            # Prikazuje prvi kolonu
awk -F: '{print $1, $3}' /etc/passwd # Korisnik i UID iz /etc/passwd (separator: :)
awk '$3 > 1000' /etc/passwd          # Redovi gde je treće polje > 1000
awk 'NR==5' fajl.txt                 # Samo peti red
awk 'END{print NR}' fajl.txt         # Broji ukupan broj redova
awk '{sum+=$1} END{print sum}' br.txt # Sumira prve kolone
```

sort - Sortira redove teksta:

```
sort fajl.txt                        # Abecedno sortiranje
sort -r fajl.txt                     # Obrnuto sortiranje
sort -n brojevi.txt                  # Numeričko sortiranje
sort -k2 fajl.txt                    # Sortiranje po drugoj koloni
sort -t: -k3 -n /etc/passwd          # Po UID, sa separatorom :
```

```
sort -u fajl.txt                # Sortiranje i uklanjanje duplikata
```

uniq - Uklanja ili prikazuje duplikate. Zahteva sortirani ulaz da bi bio tačan rezultat jer `uniq` uklanja **samo uzastopne (susedne) duplikate**, ne sve duplikate u celom fajlu. Stoga, se često koristi kombinacija sort | uniq.

```
uniq fajl.txt                   # Uklanja uzastopne duplikate
uniq -c fajl.txt                # Broji pojavljivanja
uniq -d fajl.txt                # Prikazuje samo duplikate
sort fajl.txt | uniq -c | sort -rn   # Top lista po učestalosti
```

cut - Izdvaja kolone iz teksta. Komanda `cut` u Bash-u služi za **izdvajanje određenih kolona ili karaktera iz tekstualnog ulaza**, najčešće iz fajlova ili izlaza drugih komandi.

```
cut -d: -f1 /etc/passwd         # -f1 da bude prva kolona, -d je da bude separator :
cut -d, -f2,4 podaci.csv        # Druga i četvrta kolona CSV-a
cut -c1-10 fajl.txt             # Karakteri od 1 do 10
```

tr - Zamenjuje ili briše karaktere:

```
tr 'a-z' 'A-Z' < fajl.txt       # Konverzija u velika slova
tr -d '\n' < fajl.txt           # Uklanja nove redove
tr -s ' ' < fajl.txt            # Komprimuje višestruke razmake u jedan
echo "hello world" | tr ' ' '_'     # Zamena razmaka podvlakom
```

wc - Broji redove, reči i karaktere:

```
wc fajl.txt                     # Prikazuje redove, reči i bajtove
wc -l fajl.txt                  # Samo broj redova
wc -w fajl.txt                  # Samo broj reči
```

```
wc -c fajl.txt                  # Samo broj bajtova
ls | wc -l                      # Broj fajlova u direktorijumu
```

diff - Poredi sadržaj dva fajla:

```
diff fajl1.txt fajl2.txt
diff -u fajl1.txt fajl2.txt     # Unified format (prikazuje kontekst)
diff -r dir1/ dir2/             # Rekurzivno poređenje direktorijuma
```

tee - Čita sa stdin i piše i na stdout i u fajl:

```
ls -l | tee lista.txt           # Prikazuje i snima u fajl
komanda | tee -a log.txt        # Dodaje na kraj fajla (-a: append)
```

xargs - Prosleđuje rezultate komande kao argumente drugoj komandi:

```
find . -name "*.tmp" | xargs rm       # Briše sve .tmp fajlove
cat lista.txt | xargs ls -l           # Listanje fajlova iz liste
find . -name "*.txt" | xargs grep "greška"  # Pretraga u pronađenim fajlovima
```

7.2 Kompozicija komandi

Bash omogućava kombinovanje komandi na nekoliko načina:

Sekvencijalno izvršavanje:

```
komanda1 ; komanda2          # Izvršava bez obzira na rezultat
komanda1 && komanda2         # Izvršava komanda2 samo ako je komanda1 uspela
komanda1 || komanda2         # Izvršava komanda2 samo ako je komanda1 neuspela
```

Pajp operator | - Izlaz jedne komande postaje ulaz sledeće.

Primeri:

```
cat log.txt | grep "ERROR" | sort | uniq -c
```

Ova komanda:

1. čita sadržaj fajla log.txt
2. filtrira linije koje sadrže "ERROR"
3. sortira ih
4. prebrojava ponavljanja svake linije

```
ps aux | grep python | grep -v grep
```

Ova komanda, korak po korak:

```
ps aux
```

Prikazuje sve trenutno pokrenute procese u sistemu (za sve korisnike, sa detaljima).

```
| grep python
```

Filtrira izlaz i zadržava samo linije koje sadrže reč *python*.

```
| grep -v grep
```

Uklanja liniju koja sadrži samu grep python komandu (-v znači izbaci ono što se poklapa).

```
cat /var/log/auth.log | grep "Failed" | awk '{print $11}' | sort | uniq -c | sort -rn
```

Gornji primer: top lista IP adresa sa neuspelim prijavama. Korak po korak:

```
cat /var/log/auth.log
```

Učitava sadržaj log fajla koji beleži autentifikacione događaje (npr. SSH login pokušaje).

```
| grep "Failed"
```

Filtrira samo linije koje sadrže neuspešne pokušaje prijave (*Failed login attempts*).

```
| awk '{print $11}'
```

Izdvaja 11. kolonu iz svake linije - u ovom slučaju to je najčešće **IP adresa** sa koje je pokušaj izvršen.

`| sort`
Sortira IP adrese kako bi se identične grupisale jedna do druge.

`| uniq -c`
Broji koliko puta se svaka IP adresa pojavljuje.

`| sort -rn`
Sortira rezultat numerički (`-n`) i opadajuće (`-r`), tako da se na vrhu nalaze IP adrese sa najviše neuspešnih pokušaja.

Napomena: `cat` ovde nije neophodan - komanda se može pisati efikasnije:

```
grep "Failed" /var/log/auth.log | awk '{print $11}' | sort | uniq -c | sort -rn
```

Supstitucija komandi (command substitution) - **izlaz jedne komande koristi kao deo druge komande**. Drugim rečima, Bash prvo izvrši unutrašnju komandu, a zatim njen rezultat ubaci na mesto gde je pozvana.

```
datum=$(date +%Y%m%d)

echo "Danas je $(date)"   #Bash prvo izvrši date, a zatim njegov izlaz ubaci u echo.

backup_dir="/backup/$(hostname)_$(date +%Y%m%d)"
```

Upotreba zagrada (...) u Bash okruženju inicira pokretanje izdvojenog procesa (subshell-a), u okviru kojeg se izvršavaju navedene komande. Sve promene stanja unutar tog procesa - uključujući promenu radnog direktorijuma - ostaju lokalne za podljusku i ne utiču na okruženje roditeljskog (glavnog) shell-a.

`{ ... }`
Komande se izvršavaju u **istom shell-u** (nema kreiranja novog procesa)
`( ... )`
Komande se izvršavaju u **subshell-u** (izdvojen proces)

Primer:

```
(cd /tmp && ls)          # Izvršava u podshell-u; ne menja tekući direktorijum.
```

Objašnjenje: Komanda `cd /tmp` menja trenutni radni direktorijum unutar podljuske (engl. *subshell*) na `/tmp`. Operator `&&` obezbeđuje da se naredna komanda (`ls`) izvrši isključivo pod

uslovom da je prethodna komanda uspešno završena (tj. da je promena direktorijuma izvršena bez greške). Nakon toga, `ls` ispisuje sadržaj direktorijuma `/tmp`. Po završetku izvršavanja izraza, podljuska se zatvara, a kontrola se vraća glavnom shell-u, pri čemu njegov radni direktorijum ostaje nepromenjen. Ovakav pristup se često koristi u situacijama kada je potrebno izvršiti niz operacija u privremeno izmenjenom okruženju, bez trajnog uticaja na kontekst izvršavanja, čime se doprinosi većoj sigurnosti i predvidivosti skripti.

Ako želimo da grupišemo naredbe, a da izvršavanje bude u tekućem shell-u.

```
{ cd /tmp; ls; }          # Izvršava se u tekućem shell-u
```

Sintaksne napomene:

- mora postojati razmak posle `{` i pre `}`
- komande se razdvajaju sa `;` ili novim redom

7.3 Preusmeravanje ulaza i izlaza

Svaki proces u Linux sistemu ima tri standardna toka:

Tok	Deskriptor	Opis
stdin	0	Standardni ulaz
stdout	1	Standardni izlaz
stderr	2	Standardni izlaz za greške

Preusmeravanje izlaza:

```
ls -l > lista.txt          # stdout u fajl (prepisuje)
ls -l >> lista.txt         # stdout u fajl (dodaje)
```

ls nepostojeci 2> greske.txt # stderr u fajl

ls nepostojeci 2>&1 # stderr na stdout

ls -l > /dev/null 2>&1 # Odbacuje i stdout i stderr, tiho izvršavanje

`> /dev/null`

- Preusmerava **standardni izlaz (stdout)** u `/dev/null`
- `/dev/null` je specijalan “crna rupa” fajl - sve što se u njega pošalje se **odbacuje**

`2>&1`

- `2` označava **standardni error (stderr)**
- `>&1` znači: preusmeri stderr na isto mesto gde ide stdout, a pošto je stdout već preusmeren u `/dev/null`, i stderr ide tamo

Redosled je važan. U narednom primeru stderr ide na originalni stdout (terminal), pa se greške ipak vide:

ls -l 2>&1 > /dev/null

komanda &> izlaz.txt # stdout i stderr u isti fajl (Bash 4+)

komanda > izlaz.txt 2>&1 #ekvivalentno prethodnom (portabilniji)

Preusmeravanje ulaza:

sort < nesortiran.txt # stdin iz fajla

mysql -u root baza < dump.sql # Učitava SQL iz fajla

Here-document (here-doc) u Bash-u je mehanizam koji omogućava da se **višelinijski tekst direktno prosledi komandi kao standardni ulaz (stdin)**, bez potrebe za spoljnim fajlovima.

Primer:

```
cat << EOF > konfiguracija.txt

[server]

host = localhost

port = 8080

EOF
```

Objašnjenje:

1. `cat << EOF`

- Pokreće **here-doc**, gde sve linije do `EOF` predstavljaju ulaz za komandu `cat`
- `EOF` je delimiter (oznaka kraja)

2. `> konfiguracija.txt`

- Preusmerava izlaz komande `cat` u fajl konfiguracija.txt
- Ako fajl postoji → **biće prepisan**
- Ako ne postoji → **biće kreiran**

3. Sadržaj između

```
[server]

host = localhost

port = 8080
```

- Ovo je tekst koji se upisuje u fajl

Ovakav postupak se često koristi za generisanje konfiguracionih fajlova.

U nastavku je još nekoliko primera.

```
# Sa supstitucijom promenljivih:

cat << EOF

Hostname: $(hostname)
```

```
Datum: $(date)
Korisnik: $USER
EOF

# Bez supstitucije (EOF u navodnicima):
cat << 'EOF'
Ovo je $bukvalan tekst - bez supstitucije
EOF
```

/dev/null - Specijalni fajl koji odbacuje sve što se u njega upiše:

```
find / -name "*.conf" 2>/dev/null     # Potiskuje greške
nohup dugacak_zadatak.sh > /dev/null 2>&1 &  # Pokretanje u pozadini bez izlaza
```

8. Pisanje i izvršavanje shell skriptova

8.1 Promenljive i interpolacija stringova

Bash skripta počinje **shebang** linijom koja definiše interpreter. Shebang linija (`#!`) je prva linija u skriptama (npr. Bash, Python) na Unix-sličnim sistemima koja govori operativnom sistemu koji interpreter da koristi za izvršavanje fajla. Sastoji se od znaka tarabe i uzvičnika (`#!`), nakon kojih sledi putanja do interpretera (npr. `#!/bin/bash` ili `#!/usr/bin/env python3`), omogućavajući direktno pokretanje skripte kao izvršnog fajla.

#!/bin/bash

Deklaracija i upotreba promenljivih:

ime="Marko" # Bez razmaka oko =

broj=42

echo $ime # Osnovna upotreba

echo "${ime}_Marković" # Vitičaste zagrade za graničenje naziva

readonly KONSTANTA="vrednost" # Konstantna promenljiva

unset ime # Brisanje promenljive

Tipovi promenljivih:

declare -i broj=42 # Celobrojna promenljiva, `declare` se koristi za definisanje promenljivih sa specifičnim atributima, **`-i`** označava da je promenljiva integer (celo broj)

declare -r KONS="nep." # Read-only

declare -a niz=("a" "b" "c") # Niz

declare -A mapa # Asocijativni niz (Bash 4+)

mapa["kljuc"]="vrednost"

Nizovi u Bash-u predstavljaju kolekcije elemenata (najčešće stringova) kojima se pristupa pomoću indeksa. Koriste se za organizaciju i obradu više vrednosti unutar jedne promenljive.

```
voce=("jabuka" "kruška" "šljiva")

echo ${voce[0]}            # Pristup elementu: jabuka

echo ${voce[@]}            # Svi elementi

echo ${#voce[@]}           # Broj elemenata

voce+=("breskva")          # Dodavanje elementa
```

- Indeksiranje počinje od **0**
- Bash podržava samo **jednodimenzionalne nizove** (bez pravih matrica)
- Elementi mogu biti različitog tipa (prakticno se tretiraju kao stringovi)
- U praksi, nizovi su korisni za rad sa listama fajlova, korisnika ili rezultata komandi, i često se koriste u skriptama za automatizaciju.

Interpolacija stringova (efikasan način umetanja vrednosti promenljivih direktno unutar strigova, čineći kod čitljivijim od klasičnog spajanja - konkatenacije) prikazana je narednim primerima.

```
ime="Student"

echo "Zdravo, $ime!"               # Dvostruki navodnici - vrši supstituciju

echo 'Zdravo, $ime!'               # Jednostruki navodnici - bez supstitucije

echo "Ima ${#ime} karaktera"       # Dužina stringa

echo "${ime,,}"                    # Konverzija u mala slova

echo "${ime^^}"                    # Konverzija u velika slova

echo "${ime:0:3}"                  # Podstring: pozicija 0, dužina 3

echo "${ime/nt/nti}"               # Zamena prvog pojavljivanja


putanja="/home/nina/dokumenti/fajl.txt"

echo "${putanja##*/}"              # Samo naziv fajla (basename) jer ##*/ uklanja najduži prefiks koji se poklapa sa */, tj. sve do poslednjeg /
```

```
#Rezultat je fajl.txt
```

echo "${putanja%/*}" # Samo direktorijum (dirname) jer `%/*` uklanja **najkraći sufiks koji počinje sa `/`**, tj. deo posle poslednjeg `/`

```
#Rezultat /home/nina/dokumenti
```

Aritmetika se može vršiti na nekoliko načina. Bash po defaultu tretira sve vrednosti kao stringove, ali postoje specijalni mehanizmi za rad sa celobrojnim vrednostima i operacijama. U narednoj tabeli je spisak aritmetičkih operatora.

Operator	Značenje
`+`	sabiranje
`-`	oduzimanje
`*`	množenje
`/`	deljenje
`%`	ostatak pri deljenju
`**`	stepenovanje
`++`	inkrement
`--`	dekrement

1. **Korišćenje `$(( ... ))`** je najčešće korišćeni i preporučeni način.

```
a=10; b=3

echo $((a + b))          # 13

echo $((a % b))          # 1 (modulo)

echo $((a ** 2))         # 100 (stepenovanje)

((a++))                  # Inkrement

rez=$((a + b))           # Može se rezultat dodeliti promenljivoj
```

2. **Korišćenje `let`**

```
a=7; b=3

let "rezultat = a * b"

echo "$rezultat"         # 21
```

Primer:

```
rezultat=$(echo "scale=2; 22/7" | bc)  # Decimalna aritmetika
```

Objašnjenje korak po korak:

`echo "scale=2; 22/7"`

- `scale=2` postavlja broj decimala na 2
- `22/7` je deljenje koje želimo izračunati
- Rezultat: `3.14`

`| bc`

- `bc` je komandni kalkulator u Linux-u sa podrškom za decimalne brojeve i preciznost
- Prima tekstualni ulaz i vraća rezultat

`rezultat=$( ... )`

- Command substitution: izlaz komande `bc` se smešta u promenljivu `rezultat`

Napomena

- Za celobrojnu aritmetiku Bash može direktno koristiti $((...))
- Za decimalne vrednosti je obavezno koristiti bc ili awk.

8.2 Kontrola toka izvršavanja

Uslovna grananja

```
ime="Ana"
if [ "$ime" = "Ana" ]; then
   echo "Pozdrav, Ana!"
fi
```

- String operatori:
  - `=` ili `==` → poređenje
  - `!=` → različito
  - `-z` → prazna niska
  - `-n` → nije prazna

Osnovna uslovna struktura u Bash-u prati sintaksu if-elif-else-fi.

Primer:

```
broj=15

if [ $broj -gt 10 ]; then

   echo "Veće od 10"

elif [ $broj -eq 10 ]; then

   echo "Jednako 10"

else

   echo "Manje od 10"

fi
```

Operatori za numeričko poređenje unutar uglastih zagrada su: -eq (jednako), -ne (različito), -gt (veće), -lt (manje), -ge (veće ili jednako) i -le (manje ili jednako). Za poređenje niski koriste se = i !=, dok se provera postojanja fajla ili direktorijuma vrši operatorima -f i -d:

```
if [ -f "/etc/hosts" ]; then
    echo "Fajl postoji"
fi
```

Operatori za poređenje:

Numerički	Znakovni	Opis
-eq	= ili ==	Jednako
-ne	!=	Nije jednako
-lt	<	Manje od
-le	<=	Manje ili jednako
-gt	>	Veće od
-ge	>=	Veće ili jednako

Operatori za testiranje fajlova:

-e fajl # Postoji

-f fajl # Regularan fajl

-d fajl # Direktorijum

-r fajl   # Čitljiv

-w fajl   # Zapisiv

-x fajl   # Izvršiv

-s fajl   # Nije prazan (size > 0)

-z "$str" # String je prazan

-n "$str" # String nije prazan

Moderni način pisanja uslova sa [[ ]]:

```
if [[ "$ime" == "Marko" && $broj -gt 10 ]]; then
    echo "Uslov ispunjen"
fi
if [[ "$fajl" =~ \.txt$ ]]; then  # Regex poređenje
    echo "Tekstualni fajl"
fi
```

Petlje

for petlja iterira kroz listu vrednosti ili opseg:

```
# Iteracija kroz listu
for jezik in Bash Python JavaScript; do
    echo "Jezik: $jezik"
done

for voce in jabuka kruška šljiva; do
    echo "Voće: $voce"
```

```
done

# Iteracija po fajlovima
for fajl in *.txt; do
    echo "Obrađujem: $fajl"
done

# Iteracija kroz opseg
for i in {1..5}; do
    echo "Iteracija $i"
done

# C-style for petlja
for ((i=0; i<5; i++)); do
    echo "i = $i"
done

# Iteracija po nizu
for elem in "${niz[@]}"; do
    echo "$elem"
done
```

Napomena: `"${niz[@]}"` vraća **sve elemente niza kao odvojene vrednosti**, sa očuvanjem razmaka u elementima, pa je ispravan način korišćenja.

Na primer:

```
niz=("hello world" "foo bar")
for e in "${niz[@]}";

do

        echo "$e";

done
```

Rezultat:

```
hello world
foo bar
```

Dok `for e in ${niz[@]}` bez navodnika **razdvaja elemente po svim razmacima**, što može dovesti do grešaka.

while petlja izvršava blok komandi sve dok je uslov ispunjen:

```
brojac=1

while [ $brojac -le 5 ]; do

   echo "Brojac: $brojac"

   ((brojac++))

done
```

Primer korišćenja while petlje za čitanje iz datoteke red po red.

```
while IFS= read -r linija; do

   echo "Red: $linija"

done < fajl.txt
```

Pretpostavimo da postoji fajl `fajl.txt`.

1. **`while IFS= read -r linija`**
    - `read -r linija` čita **po jednu liniju** iz ulaza i smešta je u promenljivu `linija`.
    - `-r` onemogućava da `read` tumači backslash (`\`) kao escape karakter.
    - `IFS=` postavlja *Internal Field Separator* na prazan string, što sprečava automatsko uklanjanje vodećih i završnih razmaka.
2. **`do ... done`**
    - Obuhvata blok komandi koje se izvršavaju za svaku liniju fajla.
3. **`echo "Red: $linija"`**
    - Ispisuje trenutnu liniju sa prefiksom `"Red: "`.
4. **`done < fajl.txt`**
    - Preusmerava sadržaj fajla `fajl.txt` na standardni ulaz petlje `while`.

Zašto se koristi ova forma?

- Bezbedno čitanje linija sa razmacima i posebnim karakterima.
- Sprečava gubitak vodećih/zaštitnih razmaka.
- Efikasno za velike fajlove, jer ne učitava ceo fajl u memoriju,

Primer: beskonačna petlja

```
while true; do
   # naredbe
   sleep 5  #Pauzira izvršavanje na 5 sekundi između iteracija
done
```

until petlja - izvršava dok uslov nije ispunjen:

```
until [ $brojac -ge 10 ]; do
   ((brojac++))
done
```

case naredba se koristi za grupisanje više uslova i izvršavanje različitih blokova komandi na osnovu vrednosti promenljive. Slično je `switch` naredbi u drugim jezicima.

```
case $promenljiva in
  obrazac1)
    # komande za obrazac1
    ;;
  obrazac2|obrazac3)
    # komande za obrazac2 ili obrazac3
    ;;
  *)
    # komande za sve ostale slučajeve (default)
    ;;
esac
```

`;;` označava **kraj bloka komandi** za dati obrazac

`|` omogućava da više obrazaca izvrši isti blok

`*` se koristi kao **default**, pokriva sve ostale vrednosti

`esac` je `case` napisan unazad, završava konstrukciju

Primer sa case: jednostavna kalkulacija

```
echo "Unesi operaciju (sabiranje, oduzimanje, mnozenje):"
read op

case $op in
```

```
  sabiranje)
      echo "Unesite dva broja:"
      read a b
      echo "Rezultat: $((a + b))"
      ;;
  oduzimanje)
      echo "Unesite dva broja:"
      read a b
      echo "Rezultat: $((a - b))"
      ;;
  mnozenje)
      echo "Unesite dva broja:"
      read a b
      echo "Rezultat: $((a * b))"
      ;;
  *)
      echo "Nepoznata operacija!"
      ;;
esac
```

Primer: proveravanje tipa fajla

```
echo "Unesi putanju fajla ili direktorijuma:"
read putanja

case "$putanja" in
    */)
        # Ako putanja završava sa /, tretiramo je kao direktorijum
        echo "Ovo je direktorijum."
        echo "Sadržaj direktorijuma:"
        ls "$putanja"
        ;;
    *.txt)
        # Ako je fajl sa ekstenzijom .txt
        echo "Ovo je tekstualni fajl."
        wc -l "$putanja"   # prikazuje broj linija
        ;;
    *.sh)
        # Ako je Bash skripta
        echo "Ovo je Bash skripta."
        chmod +x "$putanja"
        echo "Skripta sada ima dozvolu izvršavanja."
        ;;
    *)
        # Svi ostali fajlovi
```

```
        if [ -f "$putanja" ]; then
            echo "Ovo je običan fajl."
            file "$putanja"  # prikazuje tip fajla
        else
            echo "Fajl ili direktorijum ne postoji."
        fi
        ;;
    esac
```

Kontrola toka petlje:

```
break       # Prekida petlju
continue    # Preskače na sledeću iteraciju
break 2     # Prekida dve ugneždene petlje
```

Praktičan primer - Bash skripta za bekap

```
#!/bin/bash
# backup.sh - Skripta za arhiviranje direktorijuma
IZVOR="/home/korisnik/dokumenti"
ODREDISTE="/backup"
DATUM=$(date +"%Y%m%d_%H%M%S")
IME_ARHIVE="backup_${DATUM}.tar.gz"

# Provera da li izvorni direktorijum postoji
```

```
if [ ! -d "$IZVOR" ]; then
    echo "Greska: direktorijum '$IZVOR' ne postoji."
    exit 1
fi

# Kreiranje odredišnog direktorijuma ako ne postoji
mkdir -p "$ODREDISTE"

# Kreiranje kompresovane arhive
tar -czf "${ODREDISTE}/${IME_ARHIVE}" "$IZVOR"

# Provera uspešnosti operacije
if [ $? -eq 0 ]; then
    echo "Bekap uspesno kreiran: ${ODREDISTE}/${IME_ARHIVE}"
else
    echo "Greska: bekap nije uspesno kreiran."
    exit 2
fi
```

Analiza ove skripte ilustruje nekoliko ključnih principa: *shebang* linija (#!/bin/bash) eksplicitno definiše interpreter, promenljive se koriste za centralizovano čuvanje putanja, komandna supstitucija generiše jedinstveno ime arhive na osnovu trenutnog datuma i vremena, uslovna provera verifikuje postojanje izvornog direktorijuma pre nastavka, a $? se koristi za obradu grešaka nakon kritične operacije.

8.3.Funkcije

U Bash-u, **funkcije** omogućavaju grupisanje blokova komandi koje se mogu višestruko koristiti u okviru istog skripta, što poboljšava čitljivost, održavanje i modularnost koda.

Osnovna sintaksa funkcije

```
ime_funkcije() {
  # komande funkcije
}
```

Alternativni oblik

```
function ime_funkcije {
  # komande funkcije
}
```

Nema razlike u ponašanju, samo se stil razlikuje.

Funkcije mogu vratiti **status (integer 0-255)** koristeći `return`

Primer:

```
# Definicija funkcije
pozdravi() {
    local ime="$1"        # Lokalna promenljiva
    echo "Zdravo, $ime!"
    return 0              # Povratni kod (0 = uspeh)
}

# Poziv funkcije
pozdravi "Marko"
```

Primer **uslovne provere sa povratnim statusom** (exit status), koji se koristi za signalizaciju uspeha ili neuspeha.

```
provera_broja() {
   if [ $1 -gt 10 ]; then
      return 0   # uspeh
   else
      return 1   # neuspeh
   fi
}

provera_broja 15
echo $?   # 0 (uspeh)
provera_broja 5
echo $?   # 1 (neuspeh)
```

`$?` čuva status prethodne komande ili funkcije

Preporučeno poboljšanje

Da bi skripta bila sigurnija (npr. ako argument nije prosleđen):

```
provera_broja() {
  if [ -z "${1:-}" ]; then
     echo "Nedostaje argument"
     return 2
  fi
```

```
    if [ "$1" -gt 10 ]; then
        return 0
    else
        return 1
    fi
}
```

Objašnjenje

1. `$1`

- Predstavlja **prvi argument funkcije**
- Ako nije prosleđen → promenljiva je *nepostavljena (unset)*

2. `${1:-}`

- Ovo je **parametarska ekspanzija sa podrazumevanom vrednošću**
- Znači:
    - ako `$1` postoji → koristi `$1`
    - ako ne postoji → koristi **prazan string**

Ključno: sprečava grešku kada je uključen `set -u`

3. `-z`

- Testira da li je string **prazan**
- True ako je:
    - argument izostavljen
    - ili prosleđen kao prazan string `""`

Važna napomena

- `return` **ne vraća vrednost kao u drugim jezicima**, već samo status (0–255)
- Ako želiš da vratiš stvarnu vrednost (npr. broj ili tekst), koristi:
    - `echo` + `$(...)`

Funkcija sa povratnom vrednošću

```
kvadrat() {
    echo $(($1 * $1))
}
rezultat=$(kvadrat 5)      # Hvata izlaz funkcije
```

Funkcija sa lokalnim promenljivama

```
racunaj() {
  local rezultat=$(( $1 + $2 ))
  echo "$rezultat"
}

rez=$((racunaj 5 3))  # ne preporučuje se aritmetika ovako, bolje:
rez=$(racunaj 5 3)
echo $rez  # 8
```

- `local` obezbeđuje da promenljiva bude vidljiva samo unutar funkcije

`Podsetnik:`

`$( ... )` izvršava **komandu ili funkciju** i uzima njen izlaz

`$(( ... ))` se koristi **za celobrojnu aritmetiku** unutar Bash-a

8.4 Pisanje, pokretanje i testiranje skriptova

Struktura dobro organizovane skripte:

```
#!/bin/bash

#

# naziv_skripta.sh - Opis svrhe skripta

# Autor: Ime Prezime

# Datum: 2025-03-28

# Verzija: 1.0

#

# Upotreba: ./naziv_skripta.sh [opcije] argument

#



# Strogi mod - preporučuje se uvek koristiti za pouzdane skripte

set -euo pipefail
```

Pojašnjenje:

`set -e` - izlazak pri grešci

- Skripta se **odmah prekida** ako bilo koja komanda vrati **status različit od 0**
- Pomaže da se greške **ne ignorišu** i da se ne nastavlja izvršavanje sa pogrešnim podacima

Primer:

```
set -e

false

echo "Ovo se neće prikazati"  # skripta izlazi odmah posle 'false'
```

`set -u` - upozorenje na neinicijalizovane promenljive

- Svaka promenljiva koja nije definisana i koristi se izaziva **grešku i prekid skripte**

- Sprečava greške poput `echo $NEPOZNATA`

Primer:

```
set -u

echo $NEPOZNATA  # greška: NEPOZNATA nije definisana
```

`set -o pipefail` - propagacija grešaka kroz pipe

- Standardno, Bash vraća status poslednje komande u pipe-u
- `pipefail` čini da ceo pipeline vrati grešku ako bilo koja komanda u pipe-u zakaže

set -o pipefail

false | true

echo $? # 1 → greška iz 'false' se propagira

- Bez `pipefail`, `$?` bi bio 0 (status poslednje komande `true`) → greška bi bila ignorisana

Konstante

readonly SCRIPT_DIR="$(cd "$(dirname "$0")" && pwd)"   #Ova linija u Bash-u predstavlja **bezbedan način da se dobije apsolutna putanja direktorijuma u kojem se nalazi skripta** i da se ona označi kao **read-only (nepromenljiva)**.

#Ovo je korisno za **referenciranje fajlova i resursa u odnosu na lokaciju skripte**, bez obzira odakle se pokreće.

readonly LOG_FILE="/var/log/moj_skript.log"

Funkcije

#Definiše funkciju pod nazivom `prikazi_pomoc`

#Sve komande unutar `{ ... }` se izvršavaju kada se funkcija pozove

prikazi_pomoc() {

#Omogućava **unos višelinijskog teksta** direktno u skriptu

$Tekst između `<< EOF` i završnog `EOF` se šalje na standardni izlaz (`stdout`) komande `cat`

Može se koristiti za **prikaz pomoći, uputstava ili template sadržaja**

```
    cat << EOF
Upotreba: $0 [opcije] argument

Opcije:
  -h, --help     Prikazuje ovu pomoć
  -v, --verbose  Detaljan ispis
  -o FAJL        Izlazni fajl

EOF
}

# Obrada argumenata
while getopts ":hvo:" opcija; do
    case $opcija in
        h) prikazi_pomoc; exit 0 ;;
        v) VERBOSE=true ;;
        o) IZLAZ="$OPTARG" ;;
        \?) echo "Nepoznata opcija: -$OPTARG"; exit 1 ;;
```

```
    esac
done

# Glavna logika
main() {
    echo "Skript pokrenut u $(date)"
    # ... logika ...
}

main "$@"
```

Pokretanje skripta:

```
bash skript.sh              # Direktno pokretanje interpreterom
chmod +x skript.sh          # Dodavanje dozvole za izvršavanje
./skript.sh                 # Pokretanje kao izvršni fajl
source skript.sh            # Izvršava u tekućem shellu (. skript.sh)
```

set opcije za pouzdanije skripte:

Opcija	Opis
set -e	Prekida skriptu pri prvoj grešci
set -u	Greška pri korišćenju nedefinisane promenljive

set -o pipefail	Greška pri neuspehu bilo koje komande u pajpu
set -x	Ispisuje svaku komandu pre izvršavanja (debug)
set -n	Parsuje skriptu bez izvršavanja (syntax check)

8.5 Osnove debagovanja

Debagovanje pri pokretanju *(Debug važi za celu skriptu od početka do kraja)***:**

```
bash -x skript.sh      # Izvršava u debug modu (ispisuje svaku komandu)
bash -n skript.sh      # Proverava sintaksu bez pokretanja
bash -v skript.sh      # Ispisuje svaku liniju pre izvršavanja
```

- **Debagovanje pri pokretanju** je brže i jednostavnije za globalni pregled

Inline debagovanje: Kada je set -x uključen, Bash će za svaku naredbu:ispisati komandu pre njenog izvršavanja, zajedno sa proširenim vrednostima promenljivih.

Prednosti u odnosu na debagovanje pri pokretanju**:**

- Precizno debagovanje samo određenog dela koda
- Manje „šuma“ u izlazu
- Pogodno za kompleksne skripte

```
set -x                 # Uključuje debug mod
# ... sumnjivi deo koda za koji je ukljucen debug mod...
set +x                 # Isključuje debug mod
```

Primer:

```
#!/bin/bash

set -x   # uključi debug mod

# deklaracija integer promenljive

declare -i suma=0

# petlja od 1 do 5

for i in {1..5}

do

   suma=suma+i

   echo "Trenutna suma: $suma"

done

set +x   # isključi debug mod

echo "Konačna suma: $suma"
```

Izgledalo bi ovako u terminalu kad pokrenemo. Linije koje počinju sa + su debug informacije. Prikazuje se kako Bash vidi komandu **nakon ekspanzije**

```
+ declare -i suma=0

+ for i in '{1..5}'

+ suma=0+1

+ echo 'Trenutna suma: 1'

Trenutna suma: 1

+ for i in '{1..5}'

+ suma=1+2

+ echo 'Trenutna suma: 3'

Trenutna suma: 3

+ for i in '{1..5}'
```

```
+ suma=3+3
+ echo 'Trenutna suma: 6'
Trenutna suma: 6
+ for i in '{1..5}'
+ suma=6+4
+ echo 'Trenutna suma: 10'
Trenutna suma: 10
+ for i in '{1..5}'
+ suma=10+5
+ echo 'Trenutna suma: 15'
Trenutna suma: 15
+ set +x
Konačna suma: 15
```

Logovanje: Ova funkcija u Bash-u predstavlja jednostavan i veoma koristan mehanizam za logovanje poruka u skriptama, uz istovremeni prikaz na ekranu i upis u log fajl. Napomena: `$LOG_FILE` mora biti definisan pre poziva funkcije:

```
LOG_FILE="/var/log/mojaskripta.log"

log() {

    echo "[$(date '+%Y-%m-%d %H:%M:%S')] $*" | tee -a "$LOG_FILE"

}


log "INFO: Skript pokrenut"  #Pozivanje funkcije log

log "ERROR: Fajl nije pronađen: $fajl"
```

`$(date '+%Y-%m-%d %H:%M:%S')`

- Ubacuje **trenutni datum i vreme**
- Format:

  2026-04-14 12:34:56

`$*`

- Predstavlja **sve argumente funkcije kao jedan string**
- Primer:

  log "Greška u modulu X"

  `$*` = `Greška u modulu X`

`echo "[timestamp] message"`

- Formira log liniju u standardnom formatu:

[2026-04-14 12:34:56] Greška u modulu X

```
tee -a "$LOG_FILE"
```

- `tee`:
  - šalje izlaz **na ekran (stdout)**
  - i istovremeno u fajl
- `-a`:
  - append (dodavanje na kraj fajla, ne prepisivanje)

Rezultat:

- poruka se vidi u terminalu
- i upisuje se u log fajl

Hvatanje grešaka:

Trap za hvatanje grešaka i čišćenje pri izlasku

trap 'echo "Greška u liniji $LINENO"; exit 1' ERR

Objašnjenje

`trap`

- Bash komanda koja **hvata signale ili događaje**
- Omogućava da definišemo šta da se desi kada dođe do:
  - greške;
  - prekida (Ctrl+C);
  - izlaska iz skripte;
  - itd.

`ERR`

- Specijalni signal u Bash-u.
- Aktivira se kada **neka komanda vrati nenulti status (grešku).**
- Radi samo ako nije onemogućeno (`set -e` može uticati na ponašanje).

`'echo "Greška u liniji $LINENO"; exit 1'`

Ovo je akcija koja se izvršava kada dođe do greške:

- `echo` → ispis poruke
- `$LINENO` → broj linije u skripti gde je greška nastala
- `exit 1` → prekida skriptu sa kodom greške

trap 'rm -f /tmp/privremeni_fajl.$$' EXIT

Naredni obrazac u Bash-u predstavlja **standardan način eksplicitnog rukovanja greškama pomoću negacije (`!`) i uslova (`if`)**.

```
# Provera povratnog koda

if ! komanda_koja_moze_da_padne; then

    echo "Komanda nije uspela" >&2

    exit 1

fi
```

`>&2` šalje poruku na **standardni error (stderr)**

Tipične greške i rešenja:

Greška	Uzrok	Rešenje
Permission denied	Nema dozvole za izvršavanje	chmod +x skript.sh
command not found	Pogrešna putanja ili naziv	Proveriti $PATH i pravopis
unbound variable	Nedefinisana promenljiva uz set -u	Koristiti ${var:-podrazumevano}
syntax error near unexpected token	Sintaksna greška	bash -n skript.sh za proveru
Beskonačna petlja	Uslov nikad nije ispunjen	Dodati set -x i pratiti tok

Prilog: Brzi referentni pregled najvažnijih komandi

Komanda	Najvažnije opcije	Opis
ls	-l -a -h -t -r -R	Listanje direktorijuma
cd	- ~ ..	Promena direktorijuma
pwd		Trenutni direktorijum
find	-name -type -size -mtime -exec	Pretraga fajl sistema
mkdir	-p -m	Kreiranje direktorijuma
cp	-r -p -i -u -v	Kopiranje
mv	-i -v	Premeštanje/preimenovanje
rm	-r -f -i -v	Brisanje
chmod	-R	Promena dozvola
chown	-R	Promena vlasnika
cat	-n -A	Prikaz sadržaja

head	-n	Početak fajla
tail	-n -f -F	Kraj fajla / praćenje
less		Straničarski pregled
grep	-i -r -n -v -c -l -E -A -B	Pretraga teksta
sed	-i -n -e	Transformacija teksta
awk	-F -v	Obrada strukturiranog teksta
sort	-r -n -k -t -u	Sortiranje
uniq	-c -d	Uklanjanje duplikata
cut	-d -f -c	Izdvajanje kolona
tr	-d -s	Zamena karaktera
wc	-l -w -c	Brojanje
diff	-u -r	Poređenje fajlova

tee	-a	Pisanje i na stdout i u fajl
xargs	-n -I -P	Prosleđivanje argumenata
ps	aux -ef	Prikaz procesa
kill	-9 -15	Završavanje procesa
chmod	+x 755 644	Dozvole
echo	-e -n	Ispis teksta
printf		Formatirani ispis
date	+%Y%m%d	Datum i vreme
history	-c	Istorija komandi
man		Priručnik (manual pages)
which		Putanja do izvršnog fajla
type		Tip komande

Literatura:

- „Learning the Bash shell”, C. Newham
- „The Linux Command Line”, W. Shotts
- „Learning Python”, M. Lutz
- Materijali za kurs Programske paradigme, prof. Dr Milena Vujošević Janičić