# Usporedba naredbenih ljusaka u operacijskim sustavima

**Jelačić, Emilia**

**SVEUČILIŠTE U ZAGREBU**

**FAKULTET ORGANIZACIJE I INFORMATIKE**

**V A R A Ž D I N**

**Emilia Jelačić**

# USPOREDBA NAREDBENIH LJUSAKA U OPERACIJSKIM SUSTAVIMA

**ZAVRŠNI RAD**

**Varaždin, 2021.**

**SVEUČILIŠTE U ZAGREBU**

**FAKULTET ORGANIZACIJE I INFORMATIKE**

**VARAŽDIN**

**Emilia Jelačić**

**JMBAG: 00161344814**

**Studij: Primjena informacijske tehnologije u poslovanju**

# USPOREDBA NAREDBENIH LJUSAKA U OPERACIJSKIM SUSTAVIMA

**ZAVRŠNI RAD**

**Mentor:**

Mag. ing. comp. Luka Milić

**Varaždin, rujan 2021.**

*Emilia Jelačić*

**Izjava o izvornosti**

Izjavljujem da je ovaj moj završni rad izvorni rezultat mojega rada te da se u izradbi istoga nisam koristio drugim izvorima osim onima koji su u njem navedeni. Za izradbu rada su rabljene etički prikladne i prihvatljive metode i tehnike rada.

*Autorica potvrdila prihvaćanjem odredaba u sustavu FOI-radovi:*

_____

## Sažetak

Ovaj rad opisuje razvoj i primjenu naredbenih ljusaka u operacijskim sustavima. Opisuje rad znamenitih naredbenih ljusaka kroz skriptne primjere i na temelju njih nudi zaključak o mogućnosti pojedine ljuske. Prvi dio je posvećen pregledu naredbene ljuske – njezine uloge unutar operacijskoga sustava i općim pregledom mogućnosti te opisom nekih znamenitih ljusaka po naredbenim sustavima. Drugi dio rada je namijenjen usporedbi najznamenitijih naredbenih ljusaka na skriptnim primjerima.



**Ključne riječi**: ljuska, Linux, skripta, Bash, PowerShell

# Sadržaj

# 1. Uvod

Početkom 1970-ih javljaju se operacijski sustavi s interpretatorom naredbenoga retka, odnosno naredbenom ljuskom, a dobivaju na važnosti razvojem sustava Unix. Razvoj naredbene ljuske može se smatrati začetkom suvremenih operacijskih sustava, koje obilježavaju članovi poput hijerarhijskoga datotečnoga sustava, pojave naredbene ljuske i jezgre (eng. *kernela*) – glavnoga upravljačkoga programa, zaduženoga za pokretanje i zaustavljanje drugih programa, međudjelovanje sklopovlja i programa te naredbene ljuske u svrhu učinkovitog rada svih članova operacijskoga sustava (Robins, Beebe, 2005.) (“Red Hat Inc”., bez dat.).

Tema ovoga završnoga rada je pregled znamenitih naredbenih ljusaka u današnjim operacijskim sustavima i njihova usporedba na skriptnim primjerima. Autora ove teme je potaknulo zanimanje za boljim razumijevanjem naredbene ljuske, posebno u sustavima Linux. Rad je podijeljen na tri poglavlja. U prvom poglavlju opisana je arhitektura operacijskoga sustava i uloga ljuske te građa same ljuske i navedene su glavne razlike između ljusaka u sustavima Linux i Microsoft Windows. Nabrojene su i u kratkim crtama opisane ljuske kojima se opširnije bave sljedeća poglavlja. Drugo poglavlje je podijeljeno na dva velika dijela, prvi dio za opis i usporedbu ljusaka unutar operacijskih sustava Linux, drugi za usporedbu unutar operacijskoga sustava MS Windows i treći za usporedbu ljusaka između tih dvaju sustava. Usporedbe su temeljene na analizi određenih skripata različitih naredbenih ljusaka. U tu svrhu su rabljeni operacijski sustavi Windows 10 i Linux OpenSUSE 15.1 u sustavu za virtualizaciju VirtualBox.

# 2. Operacijski sustavi

Operacijski sustav se može promatrati kao program sastavljen od raznih članova, poput ljuske, raznih primjenskih i upravljačkih programa, koji služe kao posrednici između sklopovlja i operacijskoga sustava, programa za *backup*, protuvirusnih programa i slično. Neki od tih elemenata su pohranjeni u memoriji, dok se drugi učitavaju iz odvojenih programa. Operacijski sustav olakšava iskorištavanje svih sklopovskih sastavnica i omogućuje višeprogramski rad. Jezgra je izravno povezana sa sklopovljem, a prima naredbe kroz ljusku (Fox, 2014.) (Budin, 2010.).

## 2.1. Jezgra

Operacijski sustav je skup programa koji služi povezivanju sklopovlja i računalne potpore računala u cilju pružanja usluga korisniku. Najvažniji programi većine suvremenih operacijskih sustava su jezgra i ljuska. Jezgra se učitava u radnu memoriju prigodom uključivanja računala i ostaje u njoj do ponovnoga isključivanja te čini osnovicu operacijskoga sustava, zbog toga jer je odgovorna za odnos između programa i upravljanja podatcima na razini sklopovlja. Jezgra određuje koji će program rabiti koje sredstvo, dakle čini raspodjelu poslova između programa (“Fox”, 2014.).

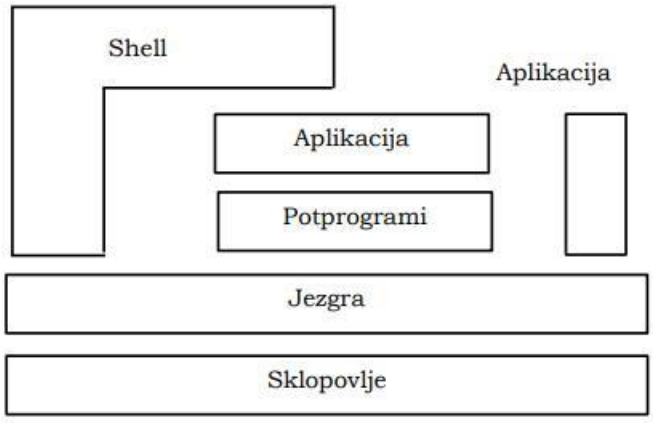
Općenito govoreći, jezgra je zaslužna za (“Fox”, 2014.):

- određivanje i dodjeljivanje procesa i sredstava primjenskih programa
- upravljanje memorijom
- upravljanje ulazno – izlaznim uređajima
- kontrolu upravljačkih programa
- sustavske pozive – upravljanje adresnim prostorom, dretvama i međuprocesnom komunikacijom

## 2.2. Ljuska

Ljuska je posrednik između jezgre i korisnika, služi kao sučelje operacijskog sustava. Ono može biti slikovno (GUI) i u obliku naredbenoga retka (CLI). Ljuska je dio operacijskoga sustava preko kojega korisnik komunicira s jezgrom. Na to upozorava i sam naziv, dakle jezgra je obavijena propusnom ljuskom, računalnim programom koji prima naredbe od korisnika i šalje zahtjev za njihovim izvođenjem. Zadaća joj je manipulacija datotečnim sustavom i ostalim dijelovima operacijskoga sustava (Fox, 2014.). Grafičke ljuske karakterizira jednostavnija, intuitivnija navigacija po sustavu, mogućnost obavljanja više radnjâ istodobno, zbog čega su široko rabljene. Međutim, naredbeni redak osigurava veću kontrolu nad kapacitetima operacijskoga sustava u smislu prilagođavanja naredaba određenim parametrima, što omogućuje i bržu pretragu, a u većini sustava su naredbe kratke i smislene što čini navigaciju još bržom i jednostavnijom (Fox, 2014.) (Gilliis A. S., 2018.).

Pored svega navedenoga, naredbene ljuske pomažu kod obavljanja rutinskih i administrativnih radnjâ na računalu kroz pisanje i pokretanje naredaba u odvojenim skriptama, koje su sastavljene su od blokova naredaba u skriptnom jeziku napisanom i prilagođenom za određenu ljusku. Neke od tih radnjâ uključuju automatizaciju određenih poslova, kao što je evidencija o stanju memorije, preimenovanje datoteka i slično. Te skripte može rabiti bilo tko, bez potrebe za poznavanjem što piše u njima (Robins, Beebe, 2005.). Naredbena ljuska služi kao sučelje prema operacijskom sustavu, pisanju skripata, a rabi se i za razvoj novih naredaba preko skriptnih jezika (Burk, Horvath, 1997.).

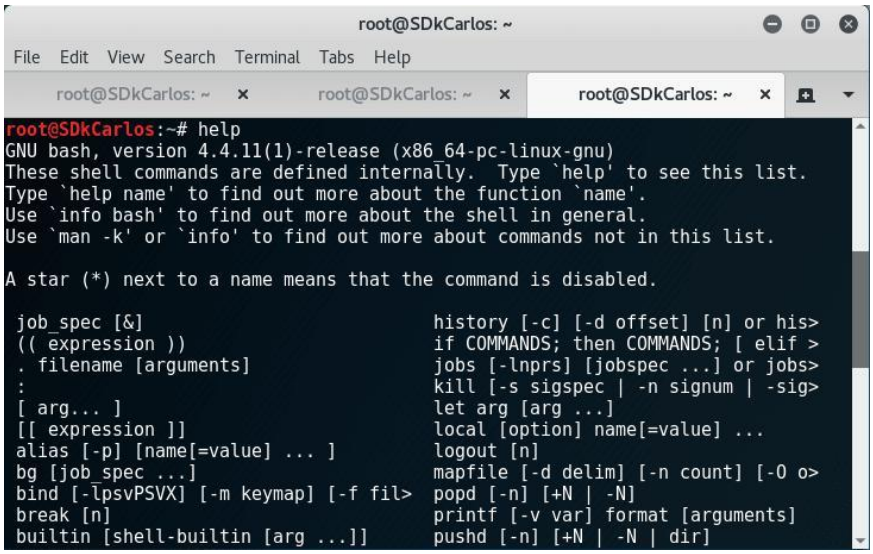



Slika 1. Jezgra i ljuska u operacijskom sustavu (Izvor: Kirasić, bez dat.)

### 2.2.1. Naredbena ljuska

Naredbena ljuska služi kao interpretator naredaba i programski jezik. Izvršavanje naredbe se postiže njezinim unosom u naredbeni redak. Naredba je često neka riječ, niz znakova ili kratica (*alias*). Ljuska može izvršavati naredbe sinkrono i asinkrono. Kod sinkronoga izvršavanja, svaku naredbu izvrši posebno i tek nakon toga dopušta novi ulaz, dok se kod asinkronoga izvršavanja više naredaba se može izvršavati istodobno (gnu.org, bez dat.). Današnje naredbene ljuske, uz sposobnost pokretanja naredaba imaju ugrađen programski jezik, sa svim njegovim članovima, poput sposobnosti definiranja i brisanja varijabala, funkcija, grananja i ostalo. Ovi članovi prvenstveno služe za skriptiranje i čine jezik visoke razine pomoću kojega je lakše manipulirati objektima u operacijskom sustavu.

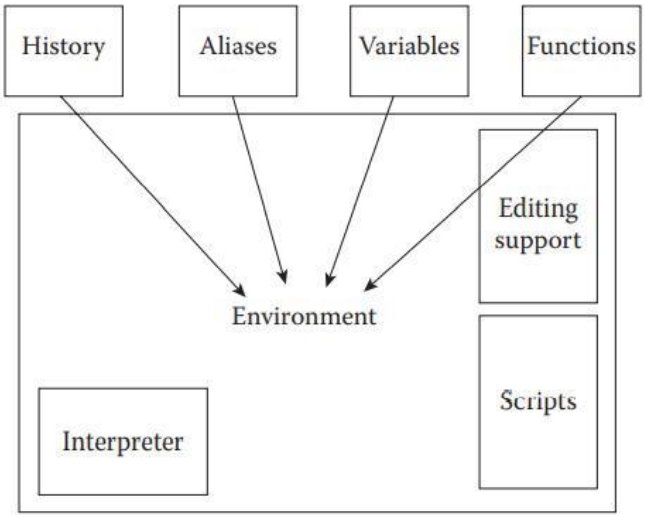
Ljuske omogućuju pregled prošlih naredaba, uređivanje naredaba, stvaranje *aliasa* i puno drugih mogućnosti kontrole nad sredstvima operacijskoga sustava (Ramey, Western, etBot al., 2020.).




Slika 2. Naredbeni redak u sustavu Linux (Izvor: Delgado, 2017.)

### 2.2.1.1. Dijelovi naredbene ljuske

Naredbena ljuska se sastoji od ugrađenoga interpretatora koji prevodi naredbu u kod za izvršavanje. Dio ljuske čine skripte koje se nalaze u posebnim datotekama. One se pokreću unosom naziva skripte koja se želi pokrenuti bez argumenata. To je neinteraktivni način pokretanja skripte jer izvršava skriptu i iziđe iz nje, a korisnik ne određuje tijek i ponašanje ljuske za to vrijeme. U Linux sustavima skripta mora započeti s znakom *shebang*, tj. `#!` i lokacijom interpretatora za jezik ljuske u kojem je napisana, dok u sustavima Windows to nije potrebno. U skripti se komentari pišu iza znaka `#`. Skripta se može pokrenuti nekim argumentima koji se pišu nakon njezina naziva, a odnose se na način izvođenja trenutačne skripte i željeno ponašanje ljuske (Ramey, Western, et al.., 2020.) (Fox, 2014.).




Slika 3. Ljuska u Linuxu (Izvor: Fox, 2014.)

U ljusci je moguće definirati i rabiti varijable. Često se rabe u skriptama kao i ostali programski konstrukti, poput petalja, selektora, ugrađenih naredaba. Ugrađene se naredbe pokreću kao dio ljuske, a ne iz vanjskoga programa. U većini suvremenih ljusaka se mogu definirati i pokretati funkcije. Unutar ljuske postoje razne opcije, na primjer, ljuska pruža korisniku mogućnost da načini *alias* za duge i složenije naredbe koje češće rabi. Vrlo važna značajka znamenitih ljusaka je pristup popisu prije unesenih naredaba, način na koji se one dohvaćaju i koji su sve načini vraćanja prošlih naredaba. Naredbene ljuske se mogu podijeliti na interaktivne ljuske, neinteraktivne ljuske i interaktivne ljuske za *login*. Interaktivne ljuske su one koje primaju neki unos. Ljuska postaje neinteraktivna kada se pokreće skripta, odnosno, kada je pokrenut neki automatizirani proces na čije izvršavanje

korisnik nema i utjecaj. Interaktivna ljuska *login* služi unosu podataka potrebnih za prijavu korisnika u sustav (Ramey, Western, et al., 2020.).

### 2.2.2. Skriptni programski jezici

Naredbene ljuske sadržavaju skriptne programske jezike, koji se sastoje od programskih konstrukata kao i ostali jezici visoke razine – funkcijama, varijablama, nizovima, kontrolom tijeka, itd. Kao što je prije navedeno, ljuska je interpretator programskoga koda, što znači da je sav kod napisan u jeziku ljuske, tumačen redak po redak dok je program pokrenut. Interpretatorski jezici, u odnosu na kompilatorske lakše manipuliraju objektima u memoriji poput datoteka i kazala, što pojednostavljuje i ubrzava proces pisanja skripte te omogućuje jednostavniju promjenu koda, što neznatno usporava proces izvršavanja skripata (Robbins, Beebe, 2005.). Skriptne jezike karakterizira jednostavnost izvođenja, brzina i jednostavnost pisanja skripata i prenosivost na više sustava, čemu u prilog ide činjenica da je većina sustava Unix kompatibilna s ljuskom Bourne ili njezinim nasljednicama (Fox, 2014.).

Uz jezike naredbene ljuske, neki od znamenitijih interpretatora su Javascript, PHP, Python, jezici koji se najčešće rabe za manipulaciju objektima prigodom *web*-programiranja, ali i u administraciji, multimediji, itd. S obzirom na to da se usporedno razvijaju skriptni jezici za naredbenu ljusku, oni postaju najjednostavnije oruđe za ovu vrstu programiranja. Većina tih jezika prati standard POSIX, što ga čini kompatibilnim s većinom sustava temeljenim na Unixu. U sustavima Microsoft Windows su razvijeni skriptni jezici koji se puno razlikuju u skladnji od onih raširenih po sustavima Linux i Mac, ali svrha im je ista. U prilog tomu ide i činjenica da je ljuska PowerShell, najpoznatija ljuska za MS Windows, standardizirana standardom POSIX IEEE POSIX 1003.2. Najvažniji zadaci skriptnih jezika su: automatizacija poslova, izvlačenje podataka iz baze i kontrola nad operacijskim sustavom (Robbins, Beebe, 2005.).

### 2.2.3. Automatizacija poslova kroz skripte

Skriptiranje se najčešće rabi u svrhu automatizacije rutinskih poslova, kako isti niz naredaba ne bi se trebao pisati više puta. Skripte se pišu u skriptnim jezicima, a sastoje se od naredaba, varijabala, petalja, selektora i ostalih elemenata programskih jezika. Skriptni jezici omogućuju automatiziranje određenih ulaza, stvaranje vlastitih jednostavnih programa što čini administraciju jednostavnijom i bržom. Automatizacija administratoru omogućuje da rutinske radnje obavi u kraćem roku uz manje posla, s obzirom na to da skriptu piše samo jednom te ju rabi po potrebi. Česti primjeri skriptiranja u svrhu automatizacije uključuju stvaranje korisnika, izradbu sigurnosnih preslika, praćenje unutarnje memorije i sl. Skripte su jednostavne za pozivanje, pri čem štede vrijeme, a lako je pronaći i ispraviti pogrješke u kodu. Negativni vidovi skriptiranja su, između ostaloga, manjak kompatibilnosti između operacijskih sustava, sporost u izvršavanju skripata i ostalo (Bhardwaj, P. K. 2006.).

## 2.3. Skupina standarada POSIX

Portable Operating System Interface (POSIX) je standard objavljen od instituta IEEE, prvi put u 1980-ima. Ovaj standard je definiran za potrebe lakše kompatibilnosti između različitih operacijskih sustava. Većina naredbenih ljusaka odgovara standardu POSIX 1003.2., definiranomu za naredbene ljuske. Nastao je iz potrebe za lakšim prijenosom operacijskoga sustava na različite modele računala, bez potrebe za ponovnim razvojem istih programa. Neka važna načela dizajna POSIX su: što jednostavniji prijenos primjenskih programa, što se ne odnosi samo sustave Unix, standardizacija izgradnje računalnih sustava kroz odnos između primjenskih programa i operacijskih sustava, a ne kroz određivanje načina njihove implementacije (Linux Hint, bez dat.) (Robbins, Rosenblatt, 2002.). Po standardu se očekuje mogućnost prijenosa izvornoga koda po svim platformama. Sustavi koji su najprilagođeniji standardu POSIX su MacOS, Solaris i dr., dok su Linux, Android i dr. većim dijelom prilagođeni, ali nisu certificirani (Robbins, Rosenblatt, 2002.).

# 3. Naredbene ljuske

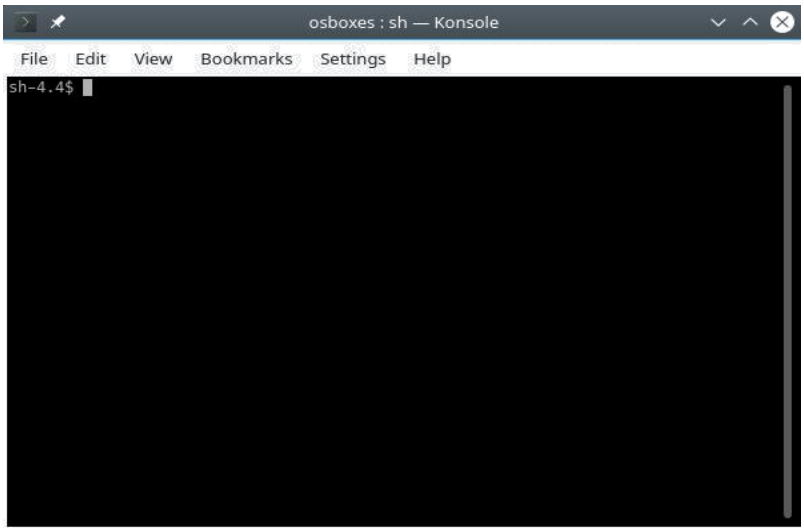
## 3.1. Sustavi Linux

Unatoč tomu što današnje distribucije Linux posjeduju slikovno sučelje, velik dio korisnika Linux rabi naredbeni redak. Razlozi toga su, između ostaloga, brža pretraga, bolja kontrola nad sredstvima. Sustavi Linux su otvorenoga koda, što omogućuje pregled izvornoga koda i pojednostavljuje distribuciju sustava Linux (Fox, 2014.).

U ljusci postoje varijable koje mogu biti predefinirane ili postavljene od korisnika. Predefiniranim varijablama ljuska automatski zadaje vrijednost prigodom prijave u sustav. Njima se ne može promijeniti sadržaj i često su napisane velikim slovima, a služe za promjenu postavaka ljuske i za pružanje informacija o određenim datotekama (npr. `HISTFILE` za datoteku `.history`) i slično. Varijable se postavljaju u naredbenom retku i u ljusci. Varijabla se u praksi piše malim slovom, a poziva se znakom `$`. Predefinirane varijable su i pozicijski parametri. One u sebe pohranjuju argumente naredbe, tako da se prva riječ pohranjuje u varijablu `$0`, druga u `$1`, do `$9`. Predefinirana varijabla `$LINENO` se rabi kod skriptiranja i prikazuje koji se redak koda u skripti trenutno izvršava. Pomoću ove varijable je moguće odrediti koji redak skripte da ljuska izvrši (Ramey et al., 2020.).

### 3.1.1. Bourne

Ljuska Bourne (`sh`) je nastala u AT&T Bell Laboratoriesu u SAD-u 1977. godine. Njezin razvoj pripisuje se inženjeru Stephenu Bourneu. Napisana je za tadašnji operacijski sustav Unix 7. Na temelju arhitekture i mogućnosti ljuske Bourne se razvija ljuska POSIX.2 koja služi kao standard za izgradnju naredbene ljuske (Joy, M, 2002.).



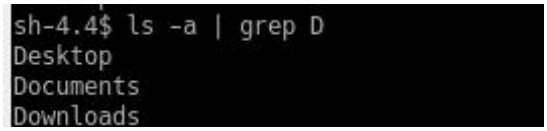

Slika 4. Terminal Bourne u OpenSUSE 15.1

### 3.1.1.1. Naredbe i preusmjeravanje izlaza

Ljuska Bourne je nastala zbog brojnih tehničkih nedostataka dotadašnjih ljusaka. Uvodi razne inovacije u kontekstu naredbenoga retka i skriptiranja, neke od kojih su: prorjeđivanje rezultata naredbe pomoću biljega *pipe* (`|`), dodani su selektori i petlje za kontrolu tijeka, omogućeno je definiranje varijabala, uvid u upravljanje signalima u skriptama i ostalo (Burk, Horvath, 1997.). Vrlo važna inovacija ove ljuske je mogućnost korisnika da kroz ljusku pošalje određen signal nekomu pokrenutomu procesu, kako bi ga promijenio ili zaustavio. Razvojem ljuske Bourne pojavljuje se i mogućnost kontrole nad opisnicima datoteka. Opsinik datoteke je identifikacijski broj koji posjeduje svaka datoteka u operacijskom sustavu, a opisuje to sredstvo i način na koji mu se može pristupiti (Keel, 2021.).

Osnovne naredbe ljuske Bourne rabe se u terminalu i u skriptama, a imaju razne funkcije. Većina osnovnih naredaba Bourne je ista kao u drugim ljuskama za Unix/Linux. Neke od njih su: `break` – izlazak iz petlje, `cd` – pozicionira se u navedenoj podmapi, `echo` – ispisuje vrijednost varijable, `exit` – zatvara *terminal* ili zaustavlja pokrenuti proces, `set` – mijenja opcije izvršavanja naredaba, `find` – pronalazi datoteku ili kazala, `mkdir` – stvara kazalo, `rmdir` – briše kazalo, `touch` – stvara datoteku, `cat` – ispisuje sadržaj datoteke, `ln` – stvara poveznicu na dokument. Navedene naredbe su osnova poznavanja rada s naredbenim retkom, a rabe se i u svim drugim ljuskama u sustavima Linux i šire. Naredbu u ljusci Bourne čini neki niz slova koji tvore riječi ili kratice izraza. Unos naredbe se sastoji od unosa naredbe, nakon čega dolazi razmak i nakon toga često argumenti na tu unesenu

naredbu. Na primjer, naredba `ls -a` prikazuje sve datoteke u trenutačnom kazalu, a argument `a` služi za prikaz i skrivenih datoteka (Arensburger, A., 2005.) (Pfaff, 1999.). Biljeg *pipe* (`|`), razdvaja različite naredbe, od kojih se svaka izvodi kao odvojen proces, ali je izlaz prve naredbe ulaz u drugu, a to čini preko tog znaka (Pfaff, B., 1999.).




Slika 5. Rezultat naredbe `ls -a` u Bourneu

Ljuska Bourne omogućuje preusmjeravanje rezultata ili ulaza naredbe u određeni dokument, odnosno iz njega. To se radi operatorima preusmjeravanja. Operator `> neki_doc.txt` preusmjerava rezultat naredbe u tu tekstualnu datoteku. Na primjer, `ls -a | grep d > neki_doc.txt`, ispisuje rezultat ove naredbe u `neki_doc.txt`. Ako se želi ispisati još jedan ili više rezultata u istu datoteku, rabi se `>>`, kako se ne bi obrisao prvi unos. Operatorom manje, `<` stvara se ulaz iz datoteke u kojoj je pohranjen. Na primjer, `< datoteka.txt`. Ako su dvije (ili više) naredbe razdvojene znakovima `&&`, prvo se izvodi jedna i ako je ona uspješna, izvodi se i druga. Znakovi `||` se pišu između naredaba, ako se želi da se druga naredba pokrene ako prva nije bila uspješna (Pfaff, B., 1999.).

### 3.1.1.2. Osnovni programski elementi

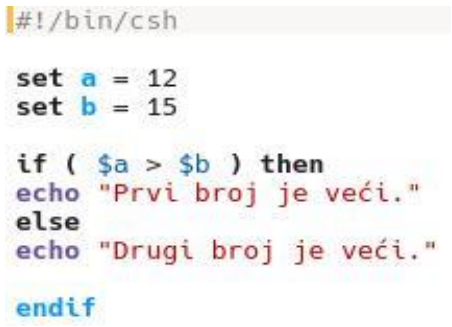
Ljuska Bourne je ujedno i interpretativni skriptni jezik, što je korisnicima omogućilo pisanje skripata na temelju naredaba i programskih konstrukata, koji su poslužili obradbi podataka, primjerice, varijabala, naredaba, logičkih izraza, petalja za njihovu manipulaciju (Pfaff, B., 1999.). Skladnja postavljanja varijable unutar ljuske Bourne glasi: `varijabla=vrijednost`, pri čem nema razmaka. Varijabla se poziva znakom `$`, u ovom slučaju bi to bilo `$varijabla` ili `${varijabla}`. Naziv može sadržavati brojeve i podvlake, ali ne smije započeti brojkom. Neke varijable unutar ljuske Bourne su predefinirane, poput `$*` koja pohranjuje svaku riječ unesene naredbe, svaki pozicijski parametar kao niz. `$@` pohranjuje sve pozicijske parametere, ali isključuje prvu riječ, tj. pozicijski parametar `$0`. Svaki sljedeći je pohranjen u posebnu varijablu, od `$1` do `$9` (IMB, bez dat.).

Skripta ljuske Bourne započinje izrazom *shebang* `#!/bin/sh`, gdje `#!` upućuje na to koji interpretator se rabi u skripti, dok je `/bin/sh`, mjesto na kojem je pohranjen interpretator ljuske Bourne (Foster-Johnson E., et al., 2005.). Usprkos tomu, u ovoj ljusci nije moguće prikazati popis proših naredaba ili se vratiti na prethodnu naredbu, kao ni vratiti se na određeni dio trenutačnoga teksta naredbe. Navigacija naredbenoga retka ograničena je na brisanje trenutačnoga unosa znak po znak i na pokretanje naredbe, što se tradicionalno čini tipkom `Enter` (Foster-Johnson E., et al., 2005.). Naredbeni redak nije moguće preurediti ili urediti po želji, ne posjeduje mogućnost uređivanja naredaba, kao što ni ne prikazuje stazu trenutnoga kazala. Za većinu operacija su potrebni vanjski programi o kojima je ljuska ovisna, a što dodatno usporava izvođenje (Pfaff, 1999.). Bourne se danas rijetko rabi, no unatoč tome je vrlo važna u povijesnom razvoju zbog toga što je njezina arhitektura uzeta kao temelj razvoja suvremenih ljusaka, poput ljusaka `bash`, `zsh` i `ksh`, a većina elemenata programiranja je slična ili jednaka u većini ljusaka za Linux (Jones, 2011.).

### 3.1.2. Ljuska C

Ljuska C (`csh`) je nastala 1978. godine na sveučilištu Berkeley u SAD-u. Tvorac ljuske je Bill Joy, tadašnji student na spomenutom sveučilištu i američki inženjer računarstva. Naziv ljuske potječe od programskoga jezika C s kojim dijeli sličnost u nekim dijelovima koda. Glavni razlog zbog kojega je temeljena na jeziku C je kompatibilnost s Unix-om, koji je isto napisan u jeziku C (Jones, 2011.).

Iako je ljuska C nastala samo godinu dana nakon ljuske Bourne, sadržava više mogućnosti, dok neke od njih uključuju sposobnost vraćanja na prethodnu naredbu, što se ostvaruje tipkama sa strjelicama *gore/dolje* ili upisom biljega `!!` i pritiskom tipke `Enter`. Biljeg `!n` se rabi za izvršavanje prošle naredbe sa svim argumentima s kojima se bila izvršena i u početku. Pri tom `n` obilježava redni broj te naredbe na popisu prošlih naredaba, a do popisa se dolazi unosom naredbe `history` u ljusku. Biljeg `!$` unosi i pokreće ljusku za zadnjim argumentom prethodne naredbe, a `!-n` izvršava naredbu koja je izvršena prije n prošlih naredaba, a `!:n` obilježava `n`-ti argument prošle naredbe (University of Hawaii, 2001.). Sposobnost izravnoga procjenjivanja izraza, bez potrebe za pokretanjem vanjskih programa, je pojednostavila skladnju i ubrzalo izvođenje programa. Zbog razvijenijih mogućnosti je ubrzo postala široko rabljena i ugrađivana u sustave Unix, posebice onih otvorenog koda, razvijena s licencom sveučilišta Berkeley (Joy, bez dat.) (Foster-Johnson, et al., 2005.).




Slika 6. `If`-grananje u ljusci C

Ljuska C uvodi sposobnost definiranja tzv. *aliasa*, koji omogućuje da se nekoj naredbi promijeni naziv u drugu, za potrebe kraćenja češćih naredaba i slično. Primjer postavljanja *aliasa* je: `$ alias md = 'mkdir -p'`. Za poništavanje *aliasa* se rabi naredba `unalias`. Ljuska C posjeduje mogućnost pokretanja programa u podlozi i po potrebi njihovo pozivanje u prednji plan (kontrola poslova) (Joy, bez dat.).

### 3.1.3. Ljuska Tenex C

Pet godina poslije Ken Greer na sveučilištu Carnegie Mellon u SAD-u razvija ljusku `tcsh`, koja zove i ljuska Tenex C. Nastala je s namjerom da se ljusci C prošire mogućnosti, stoga se dobiva ljusaka koja je u potpunosti kompatibilna sa ljuskom C, što znači da može pokretati i čitati skripte napisane u ljusci C (Jones, 2011.) (Parker, 2011.).

#### 3.1.3.1. Dovršavanje ispravljanje unosa

Tnex C sadržava sve mogućnosti ljuske C koje proširuje automatskim dovršavanjem imena datoteka (`Tab`-tipka) i varijabala, proširivanjem teksta na bilo kojem mjestu u retku, a ne samo na kraju, kako bi ljuska mogla razlikovati što se želi dovršiti. Ako se naredba dovršava, nakon prve riječi se često pišu znakovi kao `|, &, &&` i slično, što ljuska tumači kao naredbu. Ispred varijable dolazi znak `$,` što znači da se automatsko završavanje koje počinje tim znakom odnosi i na varijable, a sve ostalo se tumači kao datoteka. Za ovu svrhu postoje i ugrađene varijable, poput `autolist` koju treba postaviti, a koja nudi popis izbora mogućih naredaba, ako se ne uspije automatski dovršiti upis. Kod pretrage datoteka u varijablu `fignore` može se staviti popis sufikasa koji se žele zanemariti kod automatskoga

nadopunjanja. Još jedna predefinirana varijabla koja služi lakšoj pretrazi je `correct` – postavlja se na `all` za ispravak pogrješno napisanih redaka, ili na `cmd` za ispravak pogrješno napisane riječi (Joy, et al., 2001.).

### 3.1.3.2. Pregled prošlih naredaba i kazala

U ljusci C je uvedena mogućnost izradbe tzv. popisa ili stoga kazala, koja je proširena u ljusci TC. To je popis kazala koji su nedavno otvarani. Naredba `pushd` ih dodaje na taj popis, a `popd` ih briše te prelazi u to kazalo, dok naredba `dirs` ispisuje popis svih kazala u stogu. Varijable koje se često spominju u ovom kontekstu su `savedirs`, koja ima mogućnost automatskoga pohranjivanja popisa kazala kod odjavljivanja sa sustava i `dirstack` za dodavanje određenih kazala na popis kazala (Jones, 2011.). Navigacija i pretraga prošlim naredbama se izvodi kao u ljusci C, bez znatnih promjena (Joy, et al., 2001.).

### 3.1.3.3. Planiranje događaja

Ljuska TC ima sposobnost planiranja događaja, automatskih radnjâ koje se pokreću nakon nekoga vremenskoga razdoblja. To se ostvaruje preko posebnih naredaba i varijabala. Na primjer, naredba `sched` stavlja druge naredbe na popis za izvršavanje u određenom vremenu, varijabla `autologout` se postavlja ako se želi zaključati ljuska u određenom razdoblju.

Ljuska `tcsh` tumači kod redak po redak, za razliku od Bournea ili jezika `bash` koji tumače po logičkim blokovima, isto tako C raspoznaje razmak između operatora u varijabli, što pojednostavljuje skriptiranje. Ljuska TC je jedna od najrabljenijih naredbenih ljusaka u Linux sustavima uz `bash`, koliko zbog naprjednih sposobnosti za vrijeme u kojem je nastala, a koliko zbog toga što je od samih početaka je otvorenog koda i često izlaze nove inačice (Joy, et al., 2001.).

### 3.1.4. Ljuska Korn

KornShell (`ksh`) je naredbena ljuska razvijena za sustave Unix 1983. godine. Nastala je u razvojnoj tvrtki Bell Labs u SAD-u, tada u vlasništvu korporacije AT&T. Nastala je u cilju unaprjeđivanja mogućnosti ljuske Bourne. Kompatibilna je s ljuskom Bourne, na kojoj se i temelji, a odražava sposobnosti ljuske C. Uvodi nove mogućnosti poput pregleda povijesti naredaba, pseudonima naredaba (*aliases*), odabira promjene izgleda naredbenoga retka, asocijativnih nizova, naredaba i metoda za kontrolu poslova (`kill, set` i sl.) (Parker, 2011.) (IBM, bez dat.).

Ljuska Korn je nastala zbog potrebe za funkcijskom nadogradnjom ljuske Bourne makar do razine ljuske C, koja je otvorenoga koda, zbog prijetnje od sve veće popularnostii programske potpore otvorenoga koda sustave Unix, u ovom slučaju, FreeBSD-a. Stoga ne čudi što je u početnim inačicama odražavala mogućnosti ljuske C, osim jedne velike razlike, a to je kompatibilnost s ljuskom Bourne i sposobnost pokretanja skripata Bourne (Foster-Johnson, Welch, 2005.). Razvijena je pod licencijom kompanije AT&T, a s vremenom postaje programska potpora otvorenog koda. Ljuska Korn može se promatrati i kao ljuska Bourne s funkcijskim nastavcima koji se aktiviraju po potrebi. Ona predstavlja niz naprjednijih mogućnosti u odnosu na Bourne, posebno oko izradbe kratica, navigacije po ljusci i upravljanja prošlim naredbama (Parker, 2011.).

#### 3.1.4.1. Sposobnosti naredbenoga retka

Korn se ne razlikuje u pristupu i skladnji od ljuske Bourne kod preusmjeravanja naredaba, pisanja više naredaba u jednom retku, koje se ostvaruje točkom sa zarezom (`;`) i razmakom između dviju naredaba. Rabi iste znakove kod zamjene naziva naredaba, imena datoteka i varijabala. Prigodom navođenja i izbjegavanja navodnika, kao i u ljusci Bourne, rabi dvostruke, jednostruke navodnike i *backtick* (`` ` ``). Jednostruki i dvostruki navodnici drugačije gledaju posebne znakove, poput `\` ili `&`. Jednostruki navodnici doslovno navode tekst između njih, a dvostruki navodnici obraćaju pozornost na posebne znakove, varijable i naredbe (Olczak, 2000.) (Burk, Horvath, 1999.).

Popis prošlih naredaba je dostupan unosom naredbe `history` u terminal. Naredba `history` dolazi s nizom opcija, na primjer, broj napisan iza nje obilježava koji prošli redak po redu se želi ponovno ispisati. Nalazi se u kazalu *home*, a varijabla `HISTFILE` sadržava njegov naziv i lokaciju (Rosenblatt, Robbins, 2002.) (Burk, Horvath, 1999.). Naredba `fc` ispisuje i uređuje naredbe s popisa prošlih naredaba te nudi uređivanje popisa s prošlim naredbama. Ponovno pokretanje određene naredbe s popisa prošlih naredaba se izvršava pomoću *aliasa* `r` i rednoga broja naredbe s popisa prošlih naredaba. Naredba `r` se ne dodaje na popis prošlih naredaba, ali se dodaje ona naredba koja je pokrenuta (Burk, Horvath, 1999.).

Za uređivanje naredaba se u ljusci Korn rabe načini uređivanja `emacs` i `vi`, a pokreću se unosom `set –o vi` ili `set –o emacs`. Način uređivanja `vi` rabi uređivač teksta `vi`, dok `emacs` rabi uređivač teksta `emacs`. Nije ih moguće rabiti istodobno. S obzirom na to da naredbe u sustavima Unix mogu postati složenije i duge, a uređivanje redaka nema mogućnosti poput uređivača teksta, ljuska Korn uvodi načine uređivanja teksta po uzoru na poznate uređivače. Jedna od pogodnosti ovakva uređivanja naredaba je jednostavnije uređivanje trenutačne i prošlih naredaba. `Emacs` način uređivanja pomaže uređivanju naredaba kombinacijom tipaka za lakše pisanje naredaba te dozivanje prošlih naredaba. U tablicama ispod su opisane glavne naredbe za navigaciju naredbenim retkom u načinu rada `emacs` (Rosenblatt, Robbins, 2002.).

Tablica 1. `emacs` – glavne tipke za uređivanje naredaba (Izvor: Rosenblatt, Robbins, 2002.)

| | | | |
|---|---|---|---|
| `ESC b` | Vraćanje unatrag za jednu riječ | `CTRL-D` | Brisanje znaka desno |
| `ESC f` | Pozicioniranje ispred trenutne riječi | `CTRL-Y` | Vraćanje zadnje izbrisanog |
| `ESC h` | Briše prvu riječ slijeva | `CTRL-P` | Vraća na prethodni redak |
| `ESC d` | Briše prvu riječ desno | `CTRL-N` | Prebacuje na sljedeći redak |
| `CRTL-B` | Vraćanje jedan znak unatrag | `ESC >` | Prebacuje na zadnji redak popisa prošlih naredaba |
| `CRTL-F` | Pomak unaprijed za jedan znak | `ESC <` | Vraća na prvi redak prošlih naredaba |

U ljusci Korn postoji mogućnost spremanja *aliasa* koji se definiraju kao zamjene za određene naredbe, kako bi se pojednostavio izraz neke složenije naredbe. Definiranje *aliasa* se, kako je spomenuto u ranijem poglavlju, rabi za kraćenje naredaba u svrhu bržega i jednostavnijega komuniciranja s ljuskom. Nakon što je *alias* definiran, ljuska ga tumači kao niz naredaba koji predstavlja. U ovoj ljusci se *aliasi* definiraju i rabe na sličan način kao i u ljusci C. *Alias* može pohraniti naredbu, argumente i opcije za određenu naredbu. Riječ *alias* je ugrađena naredba koja se pokreće svaki put kada korisnik želi definirati novi *alias* ili pribaviti popis trenutno definiranih *aliasa* (Burk, Horvath, 1999.).

```
$  alias update='sudo apt-get upgrade'
$ alias
=
autoload='\\builtin typeset -fu'
functions='\\builtin typeset -f'
hash='\\builtin alias -t'
history='\\builtin fc -l'
integer='\\builtin typeset -i'
local='\\builtin typeset'
login='\\builtin exec login'
nameref='\\builtin typeset -n'
nohup='nohup '
r='\\builtin fc -e -'
type='\\builtin whence -v'
update='sudo apt-get upgrade'
```

Slika 7. Definiranje *aliasa* i ispis postojećih u ljusci Korn

Ljuska Korn pruža razne opcije za postizanje željenih postavaka. Opcije su dostupne unosom naredbe `set` ili `ksh`. Opcije se najčešće definiraju slovom kao argumentom na unesenu naredbu. Naredba `ksh` služi pokretanju ljuske Korn u naredbenom retku, a dostupna je uz niz argumenata (Burk, Horvath, 1999.).

#### 3.1.4.2. Kontrola poslova

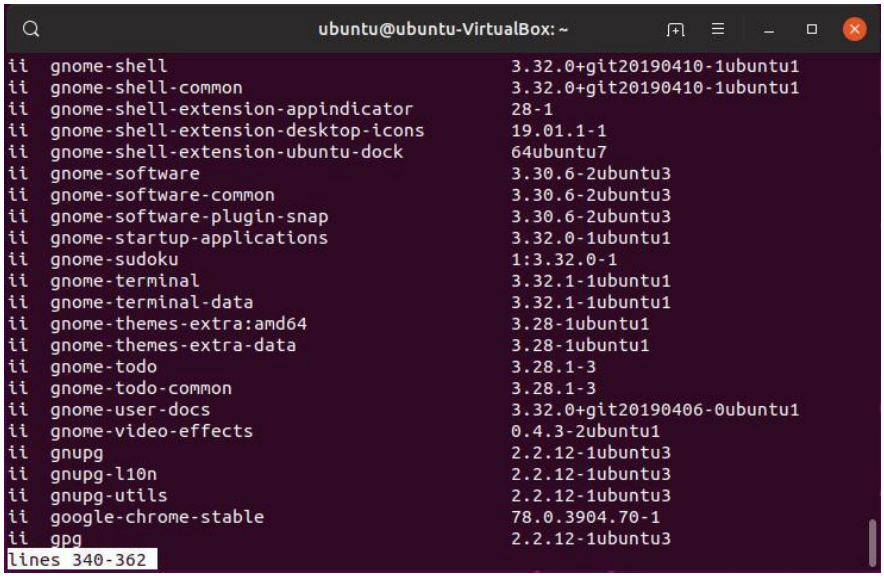
Kontrola poslova omogućuje korisniku izvršavanje više radnjâ istodobno. Moguće je jednostavnu ili složeniju naredbu, tj. asocirani pokrenuti proces prebaciti u podlogu. Tako se naredbeni redak oslobađa za sljedeće naredbe. Proces prebačen u podlogu nije nigdje prikazan. Naredba se prebacuje u podlogu ako se nakon njezina unosa stavi razmak i znak `&`. Naredba `fg` vraća pokrenuti proces iz podloge, a ako je pokrenuto više procesa, ljuska vraća onaj koji je pokrenut zadnji. Ako se želi vratiti određeni proces, nakon `fg` potrebno je napisati znak `%` i naredbu koja je pokrenuta ili broj procesa s popisa pokrenutih procesa. Naredba `jobs` nudi popis svih procesa koji se izvršavaju u podlozi, koja ispisuje trenutno pokrenute procese. Proces iz podloge je moguće prekinuti naredbom `disown` i određivanjem procesa kao u prethodnom primjeru (Robbins, Rosenblatt, 2002.).

### 3.1.5. `bash`

Bourne Again Shell, odnosno ljuska `bash` je nastala kao besplatna programska potpora, a razvijena je za Unix sustave. Sam naziv upućuje na to da se temelji na ljusci Bourne, a napisao ju je Brian Fox 1988. za Free Software Foundation. Free Software Foundation je neprofitna organizacija nastala 1985. godine, s ciljem razvoja te besplatne i slobodne distribucije i modifikacije programske potpore dostupne svima. Proizvedena je za sve sustave GNU pod licencom GNU Public License *(GPL*). `bash` je razvijen po uzoru na znamenite ljuske (`ksh`, C i druge). Nastala je kao zamjena za zastarjelu ljusku Bourne i može pokretati njezine skripte bez većih modifikacija (Jones, 2011.).

### 3.1.5.1. Značajke ljuske `bash`

Integracija mogućnosti ljuske C i `ksh` vidljive su u pogledu nekih obilježja, poput: neograničene povijesti naredaba, funkcije ljuske i *aliasa*, na primjer, ‘`ls`’ zamjenjuje *string* ‘`ls -F --color=auto --show-control-chars,`’ itd., kratice se pozivaju naredbom `alias`, a brišu se naredbom `unalias`. Omogućuje beskonačne indeksirane nizove i drugo (Robinson, 2019.). Neka od važnih obilježja ove ljuske kao i skriptnoga jezika su: Datoteke koje se čitaju kad se ljuska pokrene, na primjer, pri samom pokretanju se čita datoteka .`bashrc`, što znači da je interaktivna, a ne ljuska za prijavu, a ljuska za prijavu se pokreće unosom naredbe `-- login` (Ramey et al., 2020.). Ljuska `bash` može imitirati ponašanje ljuske Bourne, a u tom slučaju `bash` čita `.profile` datoteku u kazalu `/etc`. `bash` omogućuje jednodimenzijska polja bez ograničenja na veličinu, za čije se pozivanje rabi naredba `declare`. Neke od naredaba poput `pushd` dodaje kazala na popis posljednje posjećenih datoteka, dok ih naredba `popd` briše s popisa. Ove naredbe služe lakšoj navigaciji između kazala. Sam popis je dostupan preko naredbe `dirs` (Ramey et al., 2020.).




Slika 8. Ljuska `bash` u Ubuntu Linuxu (Izvor: Jha S., bez dat.)

`bash` se danas rabi kao zadana ljuska na većini sustava Linux i MacOS. Kako su sustavi Unix/Linux vrlo rašireni i dobro prilagođeni raznim potrebama, ljuska `bash` je dostupna na sustavima Microsoft Windows u sklopu okružja Cygwin (Parker, 2011.). `bash` je razvijen u skladu sa standardima POSIX (IEEE Standard 1003.) i kompatibilan je s ljuskom Bourne, što znači da se skripte i naredbe napisane u ljusci Bourne mogu pokretati unutar ljuske `bash`. Navigacija po popisu prošlih naredaba je slična ljusci C. Unos naredbe `history`, izbacuje popis svih unesenih naredaba, koji je najčešće ograničen na 30 (Ramey, et al., 2020.). Pozivanje varijabla u ljusci `bash` se provodi na isti način kao i u ljusci Korn. Unutar ljuske `bash` znamenit je i izraz grananja: `if/else`. Primjer skladnje izraza `if/else` je:

```
if [[ $VAR -gt 15 ]]
then
  echo "Varijabla je veća od 15."
else
  echo "Varijabla je manja ili jednaka 15."
fi
```

Uz `if/else`, u `bash`-u se rabi i logički izraz `case`, ovisno o potrebi. Od osnovnih programskih elemenata se često rabe i petlje, a to su `for`, `until i while` (Ramey, Western, et al., 2020.).

### 3.1.6. ZSH

Razvoj prve inačice ljuske Z pripisuje se Paulu Falstandu u 1990. godine, tadašnjem studentu na sveučilištu Princeton u SAD-u. Svrha razvoja ove ljuske je bila želja da se najnaprjednije sposobnosti ljuske Korn i C spoje u jednu. Ljuska C je imala razvijene interaktivne sposobnosti, dok je Korn je bila bolja za skriptiranje. Nastojanjem da se na temelju mogućnosti tih dviju ljusaka razvije ljuska koja ima naprjedne metode skriptiranja i intuitivnu, brzu i efektivnu navigaciju, nastaje ljuska `zsh`. Usprkos težnji da bude kompatibilna s ljuskama temeljenima na Bourneu, ne ostvaruje potpunu kompatibilnost ni sa jednom ranijom ljuskom, kako bi omogućila veću fleksibilnost i mogućnost nadogradnje. U velikoj mjeri je kompatibilna s ljuskom Korn i `bash`, iako nije u potpunosti prilagođena standardu POSIX (Parker, 2011., str 256.).

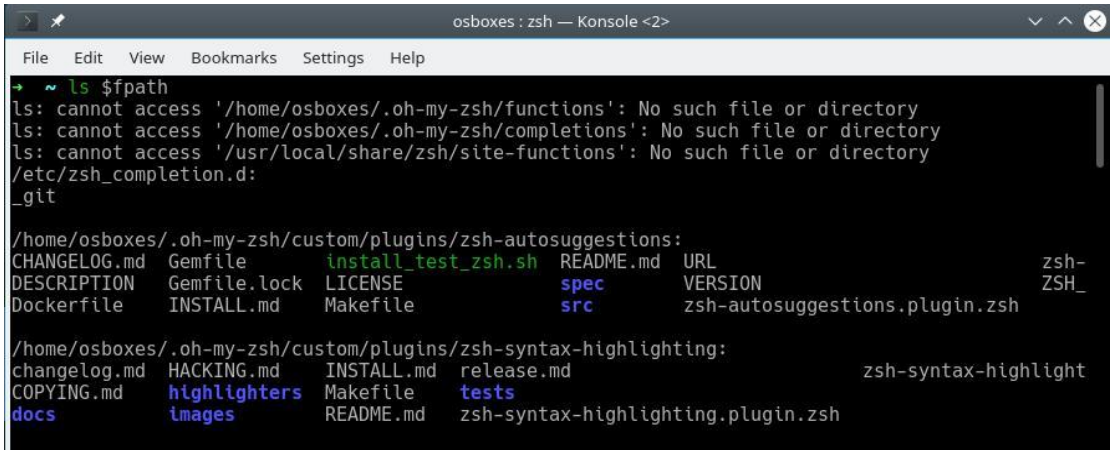
### 3.1.6.1. Značajke ljuske Z

U interaktivnom načinu rada, ljuska Z čita naredbe iz datoteke `.zshrc` koja se nalazi u korijenskoj mapi. U njoj se, uz naredbe, nalaze predefinirane funkcije, kratice i ostali članovi naredbenoga retka (Falstad, Backer, 1995.). Ljuska Z je u mogućnosti je promijeniti postavke kako bi se ponašala sličnije nekim drugim ljuskama, primjerice, naredbom `emulate bash` unos prilagođava sposobnostima te ljuske (Parker, 2011., str 256.). Značajna obilježja ove ljuske su: mogućnost postavljanja mnogobrojnih opcija koje se tiču navigacije, upravljanje datotečnim sustavom, pregledom prošlih naredaba, emulacijom drugih ljusaka i druge. Opcije se postavljaju naredbom `setopt`. Na primjer, generiranje naziva datoteka s određenim proširkama, opcija `EXTENDEDGLOB`, znakom `^` omogućuje zanemarivanje ostatka naredbe. Omogućuje postavljanje raspona cijelih brojeva, na načelu `<x-y>`, što znatno olakšava pretragu. Česte opcije su `autocd` za mijenjanje kazala bez pisanja naredbe `cd` i `correct` za ispravljanje pogrješno napisanih naredaba (Falstad, de Baker., 1995.).

Ljuska Z omogućuje preusmjeravanje unosa, *outputa* na više mjesta i rabi *pipe* (`|`) za filtriranje prikaza, što ju čini vrlo moćnom u radu s datotekama. Važnu ulogu u funkcioniranju ljuske Z imaju moduli, dijelovi `zsh`-a koji se nalaze u modulima nisu obvezni. Neki su početno povezani s ljuskom, dok je druge moguće naknadno instalirati. Ugrađeni modul `zsh/datetime` omogućuje izvođenje naredbe `strftime` koja služi dohvaćanju opširnih podataka o vremenu. Učitavanjem modula `zsh/mathfunc`, mogu se rabiti naprjednije matematičke funkcije, poput naredbe `sqrt` za izračun drugoga korijena nekoga broja, bez potrebe za pokretanjem drugih programa i usporavanjem sustava. Ljuska Z omogućuje učitavanje dodatnih funkcija koje proširuju mogućnosti ljuske pomoću novih naredaba. Kazala s tim funkcijama su definirana u varijabli `$fpath`, a moguće je i instalirati dodatne funkcije. Uz funkcije postoje i razni dodatci (kojih je oko 250) i mogućnosti promjene teme, koji pružaju neke dodatne mogućnosti, poput prikazivanje vremena pomoću znakova Unicode i slično (Eckert, 2021.).

Ljuska Z je poznata po „globiranju", generiranju naziva datoteka. Sposobnosti „globiranja“ je moguće proširiti opcijom `EXTENDEDGLOB`. Na primjer, ova opcija dopušta znaku `^` da niječe dio naredbe koji se nalazi poslije nje. Opcija `NUMERICGLOBSORT`, sortira datoteke po broju, ako ima broj u nazivu. U ljusci Z je moguće pronaći više različitih datoteka istodobno: `~ ls (backup|slike).*`. Rezultat su sve datoteke koje se zovu `backup` ili `slike`, neovisno o nastavku. Znak `~` je moguće rabiti u ljusci Z prije naziva datoteke, kako

bi se isključio iz pretrage. Opcije se postavljaju unosom naredbe `setopt` prije naziva, a prekidaju se naredbom `unset` (Falstad, Backer, 1995.).




Slika 9. Ispis kazala definiranih u varijabli `$fpath`

### 3.1.7. `fish`

*Friendly and interactive shell* ili `fish` je interaktivna ljuska nastala 2005. godine, a njezin tvorac je Axel Liljencrantz. Nastala je pod GNU-licencom i otvorenoga je koda. Ova ljuska se razlikuje od ostalih jer stavlja određena načela razvoja iznad standarda POSIX, zbog čega ga u velikoj mjeri i zaobilazi (fish-shell, bez dat.). `fish` je često zanemarena prigodom navođenja važnih naredbenih ljusaka, unatoč razvijenim mogućnostima, kako u dinamici rabljenja same ljuske, tako i kod skriptiranja. To bi moglo biti zbog odstupanja od standarda POSIX, dok su mogućnosti otprilike jednake starijim ljuskama s više dostupne dokumentacije i širom populacijom korisnika. Autor smatra da je za malo jasniji uvid u naredbene ljuske Unix sustava korisno spomenuti i one koje odstupaju od preporučenih standarda, koje se rjeđe rabe i prate drugačiji razvojni tijek (Menéndez, 2015.).

#### 3.1.7.1. Značajke ljuske `fish`

Neke od značajaka po kojima se ova ljuska izdvaja je bogato korisničko sučelje koje uključuje intuitivno naglašavanje različitih vrsta datoteka, trenutačnoga kazala, dovršavanje i filtriranje naredaba. Instalacijom ljuske `fish` uključene su neke postavke, poput obilježavanja teksta ili dovršavanja naredaba, dostupne u drugim ljuskama tek nakon konfiguracije. Povijest naredaba briše dvostruke naredbe, a postoje razne pogodne opcije vezane za povijest naredaba. Ljuska `fish` ima sposobnost promjene opcija. Isto tako, razlog izdvajanja ove ljuske je jednostavna i intuitivna skladnja što `fish` čine vrlo jednostavnim skriptnim jezikom. Posebne predefinirane varijable omogućuju brzu promjenu nekih postavaka, na primjer, za promjenu dopusnica datoteka (fish-shell, bez dat.).

Ljuska `fish` prati razvojna načela koja su nastala s ciljem promjene propusta dotadašnjih ljusaka. Ona uključuju ideju da ljuska mora imati što manje elemenata koji imaju što opću ulogu i nastojati izbjeći slične članove što čini i skriptni jezik kompliciranijim. Na primjer, suvremene ljuske rabe kratice (*alias*) i funkcije što čini dodatni problem, s obzirom a to da jedne i druge imaju svoje probleme, a u kombinaciji zbunjuju korisnika, dok ljuska `fish` rabi samo funkcije što ne smanjuje opseg mogućnosti ljuske. Načelo responzivnosti nalaže da ljuska mora biti responzivna neovisno o stanju sustava, potrebna je pravilna raspodjela memorije za sve poslove i konkretan odgovor (fish-shell, bez dat.).

Ljuska `fish` nastoji pojednostaviti pretragu vlastitih elemenata kako bi se olakšalo učenje ljuske. To čini tako da svaka pogrješka dolazi s porukom o pogrješci, najčešće obojene crvenom bojom. Pomoćna dokumentacija je brzo dostupna i potkrijepljena primjerima. Učitava vlastitu instaliranu dokumentaciju i na taj način nudi automatsko dopunjanje naredaba, koje se rabi tipkom `Tab`. S obzirom da je namjestivost raznih postavaka podložna teškoćama, pogješnim postavljanjima, `fish` je postavljena tako da izbjegava mogućnosti promjena opcija. Zato ona ne dopušta korisniku postavljanje čestih postavaka, poput definiranja broja naredaba koje se unose na popis prošlih naredaba. Ovo možda i nije najbolja vijest za upućene korisnike, međutim, tvorci ove ljuske smatraju da su na ovaj način riješeni problemi koje se često nalaze u ljuskama, poput ljusaka `bash` i `zsh` (fish-shell, Tutorial, bez dat.).

## 3.2. Windows

U sustavima Microsoft Windows rabe se dvije ljuske, Command Prompt i PowerShell. Command Prompt se razvija na temelju ljuske `COMMAND.COM`. S obzirom na to da je Microsoft Windows usmjeren na razvoj slikovnoga sučelja, prije pojave PowerShella nije bilo ljuske u sustavima MS koje bi po mogućnosti skriptiranja i opsegu mogućnosti parirale ljuskama Linux. Nakon pojave PowerShella, sustavi Windows postaju češće korišteni sustavskim administratorima, što nije umanjilo važnost i znamenitost ljusaka napisanih za sustave Linux (Stanek, 2004.) (Holmes, L., 2013.).

### 3.2.1. `Cmd.exe`

Command Prompt ili `cmd.exe` je ljuska i interpretatorski jezik u operacijskom sustavu Microsoft Windows. Ovaj program nije prenosiv. Napisan je isključivo za sustave Windows. Prvo izdanje je izašlo 1987. godine, a nasljeđuje `COMMAND.COM`, zadanu ljusku u sustavima MS-DOS, prethodnikom sustava MS Windowsa. S obzirom na razvoj slikovnoga sučelja, znamenitost ove naredbene ljuske u MS Windowsu je drastično pala kroz desetljeća postojanja, a razvijena je i ljuska PowerShell za 64-bitne sustave, naprjednijih sposobnosti. Command Prompt je zadana ljuska na 32-bitnim distribucijama Windows od 2000. godine. Nalazi se u kazalu `%SystemRoot%\System32` (Stanek, 2004.).

#### 3.2.1.1. Značajke

Kao i na operacijskom sustavu Linux, uz slikovno sučelje poželjno je imati i naredbeni redak zbog lakšega i bržega obavljanja ponavljajućih zadataka i veće kontrole nad datotečnim sustavom. Ova ljuska je nije osjetljiva na veliko i malo slovo kod pisanja naredaba. Naredbe u Command Promptu može se podijeliti na unutarnje i vanjske. Unutarnje su dio naredbene ljuske i za njihovo izvršavanje se ne pokreću vanjske datoteke, dok su vanjske one koje se najčešće izvršavaju iz vanjskih programa (Stanek, 2004.) (Asati, bez dat.).

Tablica 2. Osnovne naredbe u Command Promptu (Izvor: Stanek, 2004.)

| | |
|---|---|
| `cd` ili `chdir` | mijenja lokaciju trenutačnoga kazala |
| `cls` | briše prošle naredbe sa zaslona i iz memorije |
| `assoc` | mijenja proširku dokumenta, a bez parametara ispisuje sve proširke i tip dokumenta na koji se odnose |
| `start` | pokreće navedeni program, naredbu u drugom prozoru |
| `date` | ispisuje današnji nadnevak |
| `del` | briše datoteku, kazalo |
| `pushd` | pohranjuje trenutno kazalo i prebacuje u određenu mapu |
| `goto` | usmjerava interpretator na određen redak u skripti |
| `verify` | provjerava datoteke nakon pohranjivanja na disk |
| `Taskill -f /pid` | ubija određeni proces ili zadatak |
| `Shutdown` | gasi računalo |
| `Ipconfig /all` | ispisuje informacije sve informacije o adresi iP |
| `Systeminfo` | ispisuje konfiguraciju računala |
| `Tasklist` | ispisuje sve trenutačno pokrenute programe |
| `Copy/ Xcopy` | kopira datoteku/kopira mapu i sve datoteke u njoj |

Ljuska pamti povijest naredaba, zadani broj prošlih naredaba pohranjenih u memoriju je 50, a taj se broj može promijeniti. Za pretragu povijesti se rabe tipke sa strjelicama gore i dolje te `Enter` za unos. Tipkom `F7` se dolazi do popisa prošlih naredaba, koje se mogu odabrati na isti način kao u samom naredbenom retku. Command Prompt nudi mogućnost završavanja naredaba ako su pohranjene u memoriji. To je ostvarivo upisom prvih par slova naredbe i pritiskom tipke `F8` za završavanje (Chandio, 2020.).

Preusmjeravanje naredaba unutar naredbene ljuske je jedna od vrlo korisnih mogućnosti, a u Command Promptu skladnja je preuzeta iz Linux sustava. Biljezi > i >> preusmjeravaju izlaz naredbe u navedeni dokument. Biljeg | se rabi za povezivanje više naredaba, to jest, kada je izlaz jedne ulaz u drugu. Na primjer, `dir | find ".txt"`, ovdje se traži datoteka s proširkom `.txt` u trenutnom kazalu, a naredba `find` se odnosi na izlaz naredbe `dir`. Ako se želi porabiti neka vanjska datoteka kao ulaz u ljusku, to se čini rabeći < znak između naredbe i datoteke (Stanek, 2004.).

Definiranje varijabala u kontekstu skriptiranja je moguće u samoj konzoli, ali to se češće događa u nekom uređivaču teksta. Skladnja definiranja nove varijable počinje naredbom `set`. Na primjer, `set neki_string="Neki string."` Stvorena je varijabla `neki_string`. Neke varijable su predefinirane, a bave se praćenjem ili dokumentiranjem važnih procesa na operacijskom sustavu. Jedna takva je `errorlevel`, koja prati izlazni kod zadnje rabljenih naredaba u nizove kako bi se izvršile istodobno (Stanek, 2004.).

| Symbol | Syntax | Description |
|---|---|---|
| & | Command1 & Command2 | Execute Command1 and then execute Command2. |
| && | Command1 && Command2 | Execute Command2 if Command1 is completed successfully. |
| \|\| | Command1 \|\| Command2 | Execute Command2 only when Command1 doesn't complete successfully. |
| ( ) | (Command1 & Command2) && (Command3) | Use parentheses to group sets of commands for conditional execution based on success. |
| | (Command1 & Command2) \|\| (Command3) | Use parentheses to group sets of commands for conditional execution based on failure. |

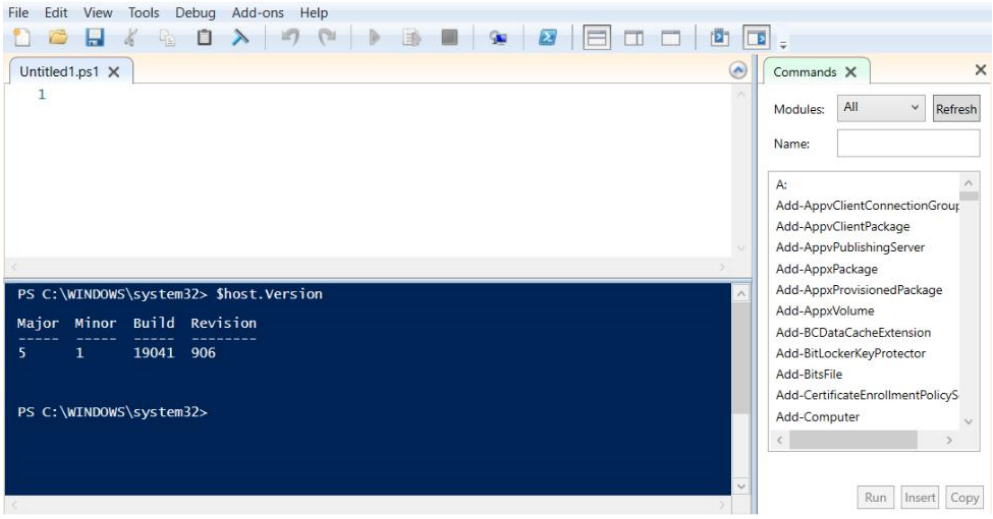
Slika 10. Povezivanje naredaba – Command Prompt

Moguće je i povezati više naredaba u slijed, gdje bi se one izvršavale slijedno, kao i neke druge kombinacije, ali treba paziti na pisanje zagrada. Dvije naredbe se izvršavaju zajedno kada između njih stoji znak `&`. Ako između dviju naredaba stoje dva znaka ampersand (`&&`), onda se druga naredba izvršava, ako je prva uspješno izvršena. Ako između dviju naredaba stoje dva znaka *pipe* (`||`), onda se druga izvršava, ako prva nije bila uspješna. Command Prompt posjeduje i sve ostale programske konstrukte, poput grananja preko `if` i `switch`, `for` i `while` petlje, a u varijable se mogu pohraniti nizovi i ima mogućnost pisanja funkcija. `Cmd.exe` je dostupna na svim MS Windowsima od 2000. godine i rabljena je kao zadana ljuska do razvoja PowerShella (Stanek, 2004.).

### 3.2.2. PowerShell

PowerShell je naredbena ljuska razvijena od MS Windowsa. Prvo izdanje izlazi 2006. godine. Sastoji se od naredbenoga retka i kombinacije objektno usmjerenoga i funkcionalnoga jezika (Microsoft, 2006.). Prema službenoj dokumentaciji (Microsoft, 2021.), PowerShell nastoji objediniti najbolje sposobnosti drugih ljusaka. Najvažnije sastavnice PowerShella uključuju prikaz prošlih naredaba i njihovo predviđanje, njihovo povezivanje znakom *pipe* (`|`), uvođenje tzv. *cmdletsa* (`command-lets`), dostupnost *aliasa* iz Command Prompta i drugo. Uz naredbeni redak, PowerShell nudi bogat skriptni jezik. Svaki izlaz u ljusci PowerShell je vrsta dokumenta objekt .NET. To znači da je vrsta podataka svakog izlaza u PowerShellu, objekt preuzet iz programskoga okvira .NET, što olakšava rad s ljuskom, za razliku od rada s čistim tekstom (Microsoft, 2021.).

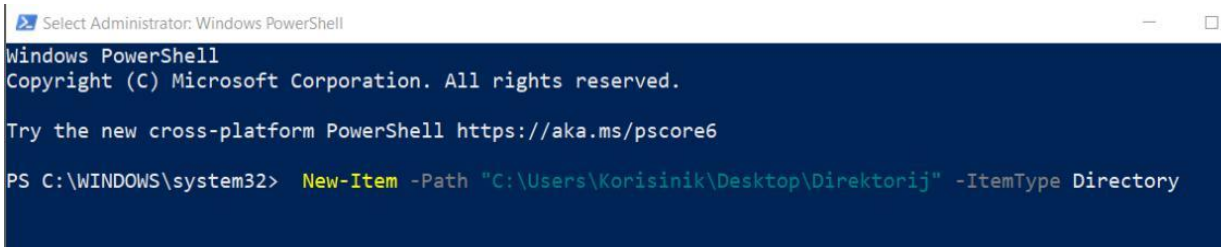
Do 2021. je izašlo sedam inačica PowerShella, od kojih je važno spomenti inačicu 5. 1, koja je izašla 2016., a napisana je za osobna računala i temelji se na .NET Frameworku. Usporedno je razvijena za Microsoft Server, temeljena na .NET Core platformi. Velika razlika između .NET Frameworka ili .NET Corea, između ostaloga je u tome što je .NET Core otvorenoga koda i dostupna na više operacijskih sustava, dok Framework ostaje razvojna okolina nativna operacijskom sustavu Windows (Pedamkar, P., bez dat.). 6. izdanje PowerShella, koji ga je učinio prenosivim na druge platforme. posebno se je primilo na Linuxu, ali nije zamijenilo inačicu 5. 1 na većini distribucija Windows. PowerShell dolazi s vlastitom okolinom za skriptiranje – PowerShell ISE (Integrated Scripting Environment) (Dent, 2019.).




Slika 11. Izgled PowerShell ISE-a na sustavu Windows 10 **–** ljuska i razvojna okolina

Slika iznad prikazuje PowerShell ISE. Zaslon je podijeljen na prostor za skriptiranje, naredbeni redak i popis naredaba s desne strane. S obzirom je inačica 5.1 u velikoj mjeri zastupljena na osobnim računalima, prigodom opisa ljuske ili članova skriptnoga jezika, autor će se referirati na ovu inačicu. *Cmdlets* su glavne naredbe u ljusci PowerShell, a kao zamjena za njih se mogu rabiti i kratice, tj. *alises*, od kojih se većina rabe kao naredba u Command Promptu. *Command-lets* se sastoje od parova glagola i imenica, povezanih spojnicom.

Razlog ovakvu postavljanju naredaba je intuitivno prisjećanje naredaba, lakše pogađanje, što ubrzava proces učenja. Popis svih glagola je ispisan unosom naredbe `Get-Verb`. Glagoli se dijele po svrsi naredbe koju čine. To su `data`, `life-cycle`, `security` i slično. Imenice predstavljaju objekte kojima bi naredba trebala manipulirati, a mogu se sastojati od više imenica, na primjer, naredba: `Send-MailMessage` (Dent, 2019.).




Slika 12. Primjer naredbe u PowerShellu 5.1

Naredbe je moguće dovršiti pritiskom na tipku `Tab` koja otvara izbornik za odabir određene naredbe. U PowerShellu postoji popis svih *aliasa*, a najlakše su dostupni naredbom `Get-Alias`. *Alias* često zamjenjuje jednostavne naredbe kako bi se skratilo pisanje, npr. `cd` umjesto `Set-Location`, `grep` umjesto `Select-String`, i slično (Holmes, 2013.).

Ljuska PowerShell ima razvijene sposobnosti skriptiranja i bogat programski jezik, koji je obogaćen objektima .NET-a, koji se mogu stvoriti ili referencirati postojeće knjižnice. One se odnose na knjižnice metoda i razreda, koje su otporne na bilo kakvu promjenu ili brisanje pa se zovu statične. PowerShell dolazi s integriranim sastavnicama .NET-a za olakšavanje skriptiranja. Najpoznatije od njih su metode i obilježja (*properties*). Pozivanje metode na neki razred se piše tako da se u uglatim zagradama pozove spomenuti razred, koji je odvojen dvjema dvotočkama od naziva metode koja se poziva.

Na primjer, to je `[io.compression.zipfile]::CreateFromDirectory($izvor, $staza)`. U ovom primjeru razred za sažimanje dokumenata zove metodu `CreateFromDirectory` i kao parametre uzima vrijednosti izvornoga i arhiviranoga dokumenta. Za pozivanje metode na objektu, skladnja je malo drugačija: "`$objekt.NazivMetode(popis parametara)`" (Holmes, 2013.).

# 4. Usporedba ljusaka na skriptnim primjerima

## 4.1. Metode i tehnike rada

U svrhu razradbe teme je rabljen sustav za virtualizaciju VirtualBox i na njem instalirani Linux OpenSUSE 15.1 i operacijski sustav Windows 10, s inačicom PowerShella 5.1. Za potrebe skriptiranja je rabljen uređivač teksta Visual Studio Code i običan uređivač teksta u Linuxu. U svrhu istraživanja je rabljena službena dokumentacija, objavljene knjige i internetski članci napisani na ovu temu.

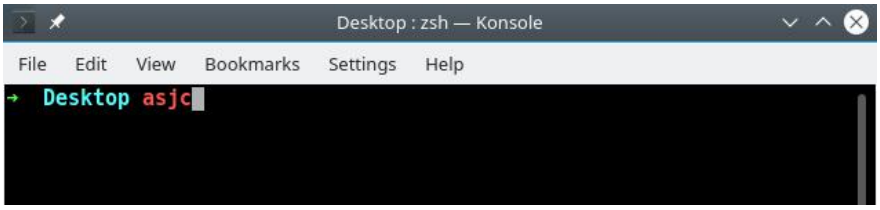
## 4.2. Usporedba `bash` i ljuske Z

U sustavu Linux se uspoređuju ljuska `bash` i Z, s obzirom na znamenitost obiju ljusaka. Usporedba će biti podijeljena na dva dijela. Prvi dio će se odnositi na usporedbu rada naredbenoga retka, tj. sposobnost navigacije datotečnim sustavom, unos jednostavnih naredaba, opcija prigodom unosa naredaba, dovršavanja naziva, ispravljanja pogrješaka kod unosa i slično. Drugi dio će se sastojati od skriptnih primjera kroz koje će se objasniti i razlika u efektivnosti koda, izvođenja naredaba, razlika u programskim konstruktima, opcijama, definiranju i pozivanju varijabala, proširivanja i slično.

### 4.2.1. Naredbeni redak

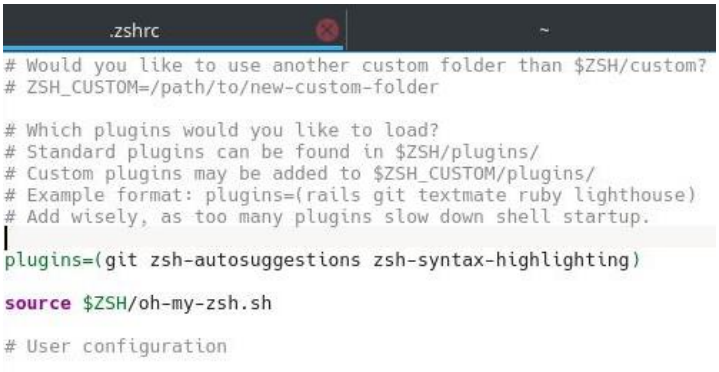
`bash` kao interaktivna ljuska čita naredbe iz konfiguracijske datoteke `.bashrc`, a ljuska Z iz `.zshrc`, koji se nalaze u kazalu *home*. Za promjenu postavaka ljuske, potrebno je promijeniti postavke navedene u toj datoteci (Falstadt, bez dat.). Prigodom instalacije ljuske Z, većina korisnika instalira i okvir *Oh-my-zsh* za upravljanje konfiguracijama, dostupan na *https://ohmyz.sh*. Preko tog okvira su dostupne brojne proširke koje povećavaju mogućnosti naredbenoga retka, a postoji čitava zajednica tvoraca koja se brine o njem (Oh-my-zsh, bez.dat). Nakon instalacije ovoga okvira i dodatka za naglašivanje skladnje i

automatsko predlaganje naredaba, naredbeni redak je promijenio izgled, sustav automatski predlaže naredbe na temelju prošlih naredaba i slično. Nakon instalacije često rabljenih proširaka, pogrješno upisane naredbe su pobojene u crveno, dok postojeće naredbe poprimaju zelenu boju.

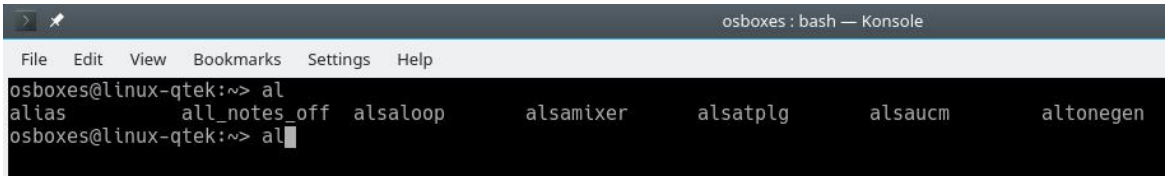



Slika 13. Unos nepostojeće naredbe u ljusku Z

Ljuska Z je kod unosa naredaba `cd`, `ls` ili `find` neosjetljiva na veliko ili malo slovo, za razliku od ljuske `bash`. U ljusci `bash` je tu opciju potrebno naknadno aktivirati tako da se promijene postavke u datoteci `.bashrc`. U ljusci `bash` je navigacija datotečnim sustavom intuitivna i dolazi s čitavim nizom ugrađenih načina za predviđanje naredaba i datoteka, primjerice, tipka `Tab` dovršava započeti niz znakova, ako zna što se želi napisati. To se odnosi na sve nazive datoteka, naredbe i nizove naredaba. Ako postoji više opcija koje počinju već unesenim nizom znakova, pritiskom na `Tab` tipku dobiva se popis svih mogućih izraza koji počinju tim nizom znakova.

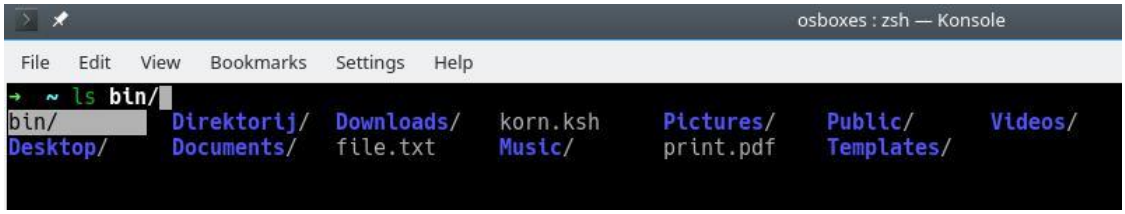



Slika 14. Datoteka `.zshrc` – `plugins`



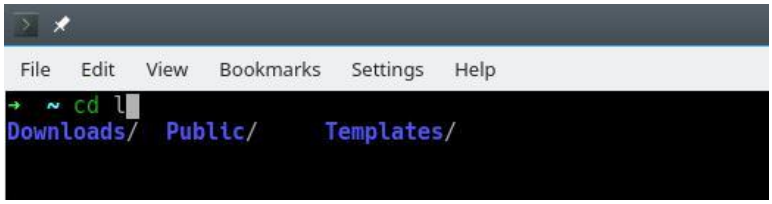

Slika 15. Predviđanje naredaba u ljusci `bash`

U ljusci Z tipka `Tab` ima sličnu funkciju, ali nakon drugog pritiska se otvara izbornik koji počinje navedenim znakovima, a ako nije unesen ni početni niz znakova, ljuska otvara Izbornik sa svim dostupnim dokumentima/naredbama, koje se može listati i pritiskom na tipku `Enter`, upisati u redak. Na Sl. 16 je prikazana pretraga korijenskoga kazala u ljusci Z:




Slika 16. Pretraga kazala u ljusci Z

Ljuska Z ima mogućnost predviđanja po bilo kojem slovu završne naredbe ili kazala, što je prikazano na slici ispod:




Slika 17. Predviđanje u ljusci Z

### 4.2.2. Usporedba `bash.sh` i `datoteka.sh`

```
#!/bin/bash

cd "/home/osboxes/Desktop/Resursi"

niz=("file" "tekst" "dokument" "skripta")

for dat in "${niz[@]}"; do

    if [ $dat == "${niz[0]}" ] || [ $dat ==  "${niz[1]}" ]

    || [ $dat == "${niz[2]}" ] || [ $dat == "${niz[3]}" ]; then

        echo $dat
        echo $dat
    fi

   mkdir -p "$dat"/podmapa_{1..5}

    done

    tekst="${niz[1]}"

    echo $tekst

    printf '%b\n' "${tekst^^}"

    shopt -s expand_aliases

    alias lh="ls -lhA" | lh
    unalias lh  | lh
```

Slika 18. Skripta `bash.sh`

Na Sl. 18 je prikazana skripta `bash.sh`. Njezinim pokretanjem stvoren je niz od četiriju članova, vrste `string` u kazalu `Resursi` na radnoj površini. Petlja `for` prolazi kroz taj niz i ispisuje svaki član niza i stvara kazala koji imaju imena kao svaki član niza. Svaki od tih kazala se sastoji od pet podmapa, koje se zovu `podmapa_1` …, tako do `podmape_5`. Nakon toga se ispisuju sva kazala u trenutnom kazalu i član niza na indeksu `1`. To je tekst u ljusci `bash`, i `file` u ljusci Z, jer indeksirani niz u njoj počinje brojem `1`. Za ispis drugog člana niza velikim slovima, u ljusci `bash` se rabi naredba za formatiranje unesene vrijednosti, s opcijom `%b\n`, koja služi za prelazak u novi red (Ramey et al., 2020.). U zagradi pored formatiranoga unosa stoji biljeg `^^`, kojim se daje ljusci `bash` do znanja kako se želi da rezultat naredbe bude ispisan velikim slovima. Mogućnosti ljuske `bash` dolaze do izražaja prigodom aktivacije posebnih opcija. Jedna od prvih radnjâ koju ljuska čini prigodom unosa naredbe je pregled svake riječi u svrhu pronalaska *aliasa* i zamjena te riječi izvornom naredbom. U ovoj skripti je definiranje i pozivanje *aliasa* omogućeno opcijom `shopt -s`

`expand_aliases`. Opcija `expand_aliases` je automatski pokrenuta unutar ljuske, no potrebno ju je pozvati prigodom skriptiranja.

U ljusci Z je moguće unijeti riječ `repeat` umjesto kontrole tijeka, kako bi se svaki član niza ispisao dva puta. Nakon `repeat` se piše broj, koji obilježava koliko ljuska treba ponoviti određenu radnju. Ovaj sustav pojednostavljuje proces skriptiranja. Ispisom članova niza na indeksu `1` se dobiva prvi član niza `$niz`, jer indeksiranje nizova u ljusci Z počinje brojem `1`. U ljusci Z ne rabi se naredba `printf` za ispis i promjenu *fonta*, nego `print`, naredba koja ne postoji u ljusci `bash` (može se rabiti i `echo`), praćena vitičastom zagradom s varijablom u koju je pohranjen tekst za ispis te dvotočka i slovo `u`, koje služi promijeni formata teksta u velika slova.

```
#! /bin/zsh

cd "/home/osboxes/Desktop/Resursi"

niz=("file" "tekst" "dokument" "skripta")


for dat in "${niz[@]}"; do

  repeat 2 echo "$dat"

   mkdir -p "$dat"/podmapa_{1..5}

done


 tekst="${niz[1]}"

 echo $tekst

 print ${tekst:u} # promijeni sub

#shopt -s expand_aliases

alias lh="ls -lhA"

lh
unalias lh

lh
```

Slika 19. Skripta `datoteka.sh` u ljusci Z

## 4.3. Usporedba ostalih ljusaka za sustave Linux

Ovo poglavlje će biti posvećeno usporedbi ostalih ljuskama za sustave Unix/Linux na nekoliko skriptnih primjera.

```
#! /bin/sh
kazalo="/home/osboxes/Desktop/Resursi"
cd $kazalo
set 'Šujičko' 'Lonjsko' 'Sinjsko'
for i do
   touch "$i.txt"
done
if [ -f Lonjsko.txt ]; then
echo "Datoteke su stvorene."
fi
```

Slika 20. `Bourne.sh`

Na Sl. 20 je prikazana jednostavna skripta u ljusci Bourne. Nakon znaka *shebang* se je potrebno pozicionirati u kazalo `Resursi` na radnoj površini. Indeksirani niz se definira kao varijabla te se nakon naredbe `set` definiraju članovi niza. Za svaki član niza se u kazalu u kojem je ljuska pozicionirana stvara tekstualna datoteka, koja se zove po svakom članu niza. `if/else` kontrola tijeka provjerava se postoji li datoteka `Lonjsko.txt`, nakon čega ljuska ispisuje poruku da su datoteke stvorene.

```
#!/bin/ksh
set kazalo = "/home/osboxes/Desktop/Resursi"
cd $kazalo
set -A polje "Šujičko" "Lonjsko" "Sinjsko"
for i in ${polje[@]}
do touch $i.txt
done
if [ -f Lonjsko.txt ]; then
print "Datoteke su stvorene."
fi
```

Slika 21. Skripta `korn.sh`

Na Sl. 21 je prikazana ista skripta u ljusci Korn. Naredbom `set` se definira nova varijabla. Definiranje polja se razlikuje od definiranja ostalih varijabala, dodavanjem opcije `A`, po čem se razlikuje od ljuske Bourne. Za svaki član polja stvara tekstualnu datoteku u mapi `Resursi`. Ako postoji datoteka `Lonjsko.txt`, naredba `print` ispisuje da su datoteke stvorene.

```
#! /bin/tcsh
set kazalo = "/home/osboxes/Desktop/Resursi"
cd $kazalo
set polje=("Lonjsko" "Sinjsko" "Šujičko")
foreach i ( $polje )
touch $i.txt
end

if ( -f Lonjsko.txt ) then
 echo "Datoteke su stvorene."
endif
```

Slika 22. `Tsch.sh`

U ljusci Tenex C varijabla tipa niz se definira davanjem nizu naziva i znakom jednakosti. Unutar zagrada su članovi niza. U ljuskama `csh` i `tcsh` je moguće rabiti petlju `foreach` za prolazak kroz članove niza, koji se referencira iza oblih zagrada. Stvaranje datoteka je jednako kao u ljuskama temeljenim na Bourneu. Niz naredaba koji se provjerava u selektoru `if` se isti piše unutar oblih zagrada, a završava riječju `endif`. Ista skripta u ljusci `fish` ne rabi zagrade ni znakove jednakosti, a varijable se postavljaju naredbom `set`. Petlja `for` ne završava riječju `done`, nego `end`. Kako bi se provjerilo postoji li datoteka `Ličko.txt`, rabi se naredba `test` i opcija `e`. `If`-izraz se ne prekida riječju `endif` ili `fi`, već isto s `end`.

```
#! /usr/bin/fish
set kazalo "/home/osboxes/Desktop/Resursi"
cd $kazalo
set polje "Šujičko" "Ličko" "Krbavsko"
for x in $polje
  touch $x.txt
end
if test -e Ličko.txt
   echo "Datoteke su stvorene."
else
  echo "Datoteka ne postoji."
end
```

Slika 23. `Fish.sh`

## 4.4. Usporedba PowerShella i CMD-a

Command Prompt je naredbena ljuska razvijena 1981. godine za starije inačice operacijskoga sustava MS Windows. Prva inačica PowerShella nastaje tek 2006. godine, sa znatnim poboljšanjima u odnosu na Command Prompt u vidu skriptiranja, postavljanja i izvršavanja naredaba i lakšom navigacijom po datotečnom sustavu (Microsoft, 2006.).

PowerShell je stvoren kao okvir za automatizaciju poslova na MS-sustavima. Vrlo je pogodan za skriptiranje, skladnja se razlikuje od skladnje Command Prompta i mogućnosti su proširene. Sadržava okvir .NET, koji u kombinaciji s već integriranim programskim jezikom čini vrlo moćno oruđe za potrebe automatizacije. Naredno poglavlje je posvećeno usporedbi Command Prompta i PowerShella, u svrhu boljega uvida nad članovima u kojima se ove dvije ljuske razlikuju te koje su im sličnosti, sa svrhom lakšega razumijevanja mogućnosti svake ljuske.

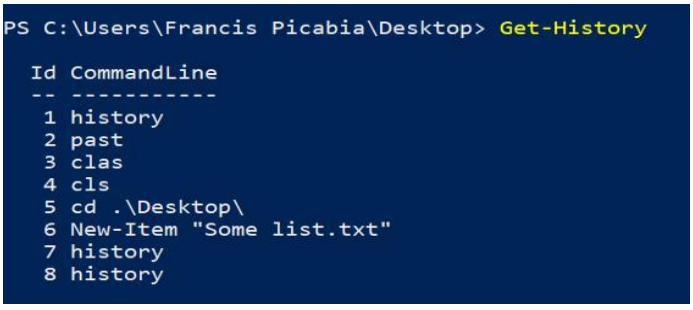
### 4.4.1. Naredbeni redak

Prvi dio se odnosi na osnovne značajke naredbenoga retka. Sposobnost ljuske da predvidi unos, ispravi pogrješke kod pisanja, kao i dostupnost datoteke s popisom prošlih naredaba, promijene datoteke prošlih naredaba, učinkovitost uređivanja retka prigodom unosa su sve sposobnosti naredbenoga retka. PowerShell i CMD se, između ostaloga, razlikuju i u nekim manjim dijelovima, poput pozivanja skripata i njihovih proširaka. CMD skripte imaju proširke `.bat` ili `.cmd`, dok PowerShell skripte imaju proširku `.ps1`.

Command Prompt poziva skripte unosom njihova naziva, a PowerShell ih pokreće kao ljuske za Linux, s točkom i kosom crtom prije naziva skripte, na način na koji se preko Linuxova *terminala* pokreću programi. U ljusci CMD je popis prošlih naredaba dostupan pritiskom na tipku `F7` i listanje tipkama gore, dolje. Prethodno unesene naredbe su dostupne unutar samoga retka u naredbenoj ljusci pritiskom na tipku gore, odnosno dolje. Popis prošlih naredaba dostupan je i unutar naredbenoga retka, unosom naredbe `doskey /history`. naredba `Doskey` pokreće vanjski program `Doskey.exe` koji služi pozivanju prethodno pokrenutih naredaba, njihovom uređivanju i stvaranju makronaredaba (Microsoft, bez dat.). Povijest prošlih naredaba je u Command Promptu dostupna samo za trenutačnu sjednicu. To znači da se ona briše nakon što se zatvori *terminal*, međutim, moguće ju je pohraniti kao zasebnu tekstualnu datoteku na računalo.


Slika 24. Popis prošlih naredaba u programu `cmd.exe`

U Ljusci PowerShell je popis prošlih naredaba dostupan unosom `Get-History` ili *aliasa* `history` u naredbeni redak (Holmes, 2013.). U svrhu jednostavnije navigacije, Command Promptu je omogućeno predviđanje imena datoteka pritiskom tipke `Tab`, kojom je omogućena pretraga svih datoteka koje su na raspolaganju unutar zadanoga kazala. Takvo dovršavanje se odnosi i na naredbe te na kazala. U PowerShellu je ta sposobnost proširena na sve naredbe, *aliase*, datoteke i kazala (Holmes, 2008.).




Slika 25. PowerShell – `Get-History`

### 4.4.2. Skriptni primjeri

Skripta `skripta1.bat` služi ispisu datoteka u trenutačnom kazalu, promjeni naziva tog kazala u `novi_zapis.txt`, provođenju provjere nad diskom i vraćanjem podataka, popravkom neke pogrješke na disku. U Command Promptu je to moguće učiniti naredbom `chkdsk`. Ova naredba vraća obavijest o tome koliko je dijelova diska pregledano, imali li pogrješaka te ih ispravlja. Bila koja obavijest koju generira ova naredba dopisuje se u `novi_zapis.txt`. Naredba `copy` služi za kopiranje datoteka, u ovom slučaju iz trenutačnoga kazala na radnu površinu. U istom retku skripta izlazi iz kazala naredbom *`cd.. .`*

Operator `&` služi za povezivanje više različitih naredaba bez potrebe za novim retkom. Češće se rabi unutar naredbenoga retka nego skripte jer je svejedno ako je sljedeća naredba u istom retku ili ne. Smještanjem znaka `&` između različitih naredaba izvršavaju se obje, jedna za drugom. Skripta će na radnoj površini stvoriti kazalo `Resursi` naredbom `mkdir`. Ponovno će se pozicionirati u kazalo `skripte` i ispisati sav sadržaj zapisa `novi_zapis.txt`, naredbom `type`. U petlji `for` se nalazi brojač koji prolazi kroz kazalo `skripte` i ispisuje koliko se datoteka trenutno nalazi u tom kazalu.

```
Get-ChildItem > zapis.txt

Rename-Item "zapis.txt" -NewName "novi_zapis.txt"


Repair-Volume >> novi_zapis.txt

Copy-Item "novi_zapis.txt"  -Destination C:\Users\Francis Picabia\Desktop
 -Recurse

copy-Item  -Path "C:\Users\Korisinik\Desktop\skripte\novi_zapis.txt"
           -Destination "C:\Users\Korisinik\Desktop" -Recurse;  cd ..


 New-Item -Type directory -Name "Nove skripte"


  Set-Location skripte; Get-Content novi_zapis.txt

 echo  (get-childitem).count
```

Slika 26. `Skripta1.bat`

Ljuska PowerShell rabi nazive koji se sastoje od glagola i imenice, a povezani su spojnicom. Velik dio tih naziva moguće je zamijeniti predefiniranim kraticama, tj. *aliasima*. Kratice u ljusci PowerShell su naredbe u CMD-u. Primjerice, naredbe poput `dir` ili `chkdsk` iz ljuske Command Prompt se mogu rabiti i u ljusci PowerShell, a nalaze se na popisu *aliasa*, odnosno kratica. *Aliasi* su dostupni unosom naredbe `Get-Alias` u naredbeni redak.

Na Sl. 27 je prikazano preimenovanje postojeće datoteke, što se izvršava naredbom `Rename-Item` i unosom `NewName` opcije za specifikaciju toga novoga imena. *Cmdlet* za kopiranje datoteke `Copy-Item` je praćen parametrima, poput `Path` i `Destination`. Za ispis sadržaja neke datoteke postoji naredba `Get-Content`, a moguće je rabiti i `type`. Parametar `Recurse` rabi se za obuhvaćanje svih potkazala prigodom izvršavanja neke radnje. U skripti `skripta1.ps1` je napravljeno novo kazalo naziva `Nove skripte`. Naredbom `Set-Location` skripta se pozicionira natrag u potkazalo skripte i ispisuje broj datoteka u njoj, što je vidljivo u zadnjem retku i što se izvršava funkcijom `count`. Kako bi se olakšalo skriptiranje u ljusci PowerShell, dostupan je PowerShell ISE, okolina za skriptiranje s bazom svih naredaba, predefiniranih varijabala i programskih konstrukata potrebnih za pisanje skripata.

```
@echo off

cd  "C:\Users\Korisinik\Desktop\skripte"

dir > zapis.txt

rename zapis.txt novi_zapis.txt

chkdsk >> novi_zapis.txt


copy novi_zapis.txt ../Desktop & cd ..

mkdir "Resursi"

cd skripte & type novi_zapis.txt


for /f %%A in ('dir ^| find "File(s)"') do set broj=%%A


echo Broj datoteka = %broj%
```

Slika 27. `Skripta1.ps1`

Skripta na Sl. 28 je napisana skripta u ljusci Command Prompt. Ona ispisuje sve datoteke unutar mape `Direktorij` te za svaku datoteku ispisuje nazive i sadržaj svih tekstualnih datoteka koje se nalaze u kazalu `Direktorij`. Opcija `/d` poreda rezultat naredbe u stupce, a opcija `/ON` ih poreda prema imenu, abecednim redoslijedom. Za obrnuti redoslijed se piše `/O-N`. Ova skripta zbraja koliko se datoteka nalazi u trenutnom kazalu i ispisuje taj broj. Isto tako, ova skripta ispisuje stazu svih datoteka u mapi `Direktorij` i njezin sadržaj. Ako ne postoji datoteka naziva `datoteke.zip`, pokreće se vanjski program `7z` za arhiviranje mape. Ako mapa pod tim imenom već postoji, ova ju skripta briše. U PowerShell skripti su pokrenute iste radnje, ali rabeći jezik i skladnju ljuske PowerShell. Ljuska rabi opciju `Filter` za filtriranje dijela naziva koji se želi pronaći. Za istodobno filtriranje i ispis datoteka, naredbe se povezuju biljegom *pipe* (`|`).

```
@echo off

   dir "C:\Users\Korisinik\Desktop\Direktorij\*.txt" /d  /ON


   for /f %%A in ('dir ^| find "File(s)"') do echo U direktoriju ima %%A datoteka.

   for %%f in ("C:\Users\Korisinik\Desktop\Direktorij.\*") do @echo %%f & type %%f



IF EXIST "C:\Users\Korisinik\Desktop\Direktorij\datoteke.zip"  (


   DEL "C:\Users\Korisinik\Desktop\Direktorij\datoteke.zip" ) ELSE  ( "C:\Program Files\7-Zip\7zFM.exe"

   a -tzip {datoteke.zip} {C:\Users\Korisinik\Desktop\Direktorij\Datoteke} )
```

Slika 28. `Skripta2.bat`

Unutar skriptnoga jezika PowerShell je dostupna i petlja `foreach`. *Cmdlet* `Foreach-Object` iz `skripte2.ps1` je često rabljen u ljusci PowerShell. Varijable se definiraju znakom `$` i nekim nizom znakova poslije njega, a pozivaju se na isti način. Kako bi kod bio što pregledniji, definirane su dvije varijable koje predstavljaju stazu i naziv dokumenta za arhiviranje. U PowerShellu je omogućeno arhiviranje datoteka preko unutarnjega sustava za arhiviranje pa nije potrebno rabiti vanjski program. To čini na način da poziva statička svojstva, preko `::` operatora metodu `CreateFromDirectory`, u zagradi je na prvom mjestu staze kazala koji se želi arhivirati, a na drugom naziv i staza arhivirane datoteke. Biljeg *pipe* je, kako u ljuskama Linux, tako i u ljusci Powershell, vrlo važan programski konstrukt, zbog toga što omogućuje da rezultat jednoga niza naredaba bude ulaz u drugu. U PowerShellu to mogu biti cijeli objekti, a ne samo nizovi naredaba, što čini ovu ljusku iznimno moćnom za skriptiranje.

Često se rabe skripte koje automatski obavljaju pojedine rutinske radnje. Primjerice, za brisanje datoteke nakon određenoga vremena od zadnjega otvaranja. To je moguće, ali malo kompliciranije od prethodnoga koda jer se rabe svojstva, poput `$_PsIsContainer` i `$_LastWriteTime`. Parametar `EA` je *alias* za `ErrorAction`, koji određuje kako ljuska postupa u slučaju da pronađe pogrješku prigodom izvršavanja trenutačne naredbe. Postavljen je na `0`, što znači da se obavijest o pogrješci neće generirati, a da će ljuska nastaviti s izvršavanjem koda. Ako varijabla koju se želi obrisati nije kazalo, što se provjerava svojstvom `$_PsIsContainer` te ako datoteka nije mijenjana 14 dana, potrebno ju je obrisati.

```
Get-ChildItem "C:\Users\Korisinik\Desktop\Direktorij" -Filter *.txt | Foreach-Object {
      echo $_.FullName
      $sadrzaj = Get-Content $_.FullName
        echo $sadrzaj
    }


$izvor = " C:\Users\Korisinik\Desktop\Direktorij\Datoteke"
$putanja = "C:\Users\Korisinik\Desktop\Direktorij\datoteke.zip"


If(Test-path $putanja) {Remove-item $putanja}
Add-Type -assembly "system.io.compression.filesystem"
[io.compression.zipfile]::CreateFromDirectory($izvor, $putanja)

 $Folder = "C:\Users\Korisinik\Desktop\Direktorij\datoteke.zip"
   Get-ChildItem $Folder -Recurse -Force -EA 0 |
     Where-Object {!$_.PsIsContainer -and $_.LastWriteTime -lt (Get-Date).AddDays(-14)}   | ForEach-Object {
     $_ | del
```

Slika 29. `Skripta2.ps1`

### 4.4.2.1. Matematičke funkcije

```powershell
$prvi_broj = Read-Host Unesi prvi broj
$drugi_broj = Read-Host Unesi drugi broj

 $broj1 = [int]::Parse($prvi_broj)
 $broj2 = [int]::Parse($drugi_broj)

function Izracunaj {

$zbroj = $broj1+$broj2
$razlika=$broj-$broj2
$kolicnik = $broj1/$broj2
$umnozak = $broj1*$broj2
$korijen = [Math]::Sqrt($zbroj)
$kvadrat = [Math]::Pow($zbroj,2)

echo "Zbroj je $zbroj"
echo "Razlika je $razlika"
echo "Umnozak je $umnozak"
echo "Kolicnik je $kolicnik"

echo "Korijen zbroja je $korijen"
echo  "Kvadrat zbroja je $kvadrat"

}

Izracunaj
```

Slika 30. Skripta `Izracunaj.ps1`

U skripti `Izracunaj.ps1`, unosom dvaju brojeva u naredbeni redak, korisnik dobiva ispisan njihov zbroj, razliku, umnožak i količnik te drugi korijen i kvadrat zbroja tih brojeva. Potrebno je promijeniti unos iz *stringa* u *integer*, što se može učiniti tako da se razred `Int` pridruži metodi `Parse`, nakon čega se u zagradi piše unos kojega je potrebno promijeniti u *integer*. Na primjeru `Izracunaj.ps1`, to je unos korisnika. Unutar funkcije `Izracunaj` su definirane varijable s aritmetičkim operacijama između unesenih vrijednosti. Za izračun korijena i kvadrata se rabe razred `Math` i metoda `Sqrt` za izračun drugoga korijena zbroja i `Pow` za izračun potencije. U ovom primjeru, to je kvadrat zbroja unosa. Nakon što se skripta pokrene, potrebno je upisati dva broja nad kojima se u funkciji `Izracunaj` vrše matematičke operacije. Funkciju je potrebno pozvati unosom naziva funkcije. Na Sl. 31 ispisan je rezultat pokrenute skripte.

```
Unesi prvi broj: 12
Unesi drugi broj: 34
Zbroj je 46
Razlika je -34
Umnozak je 408
Kolicnik je 0.352941176470588
Korijen zbroja je 6.78232998312527
Kvadrat zbroja je 2116
```



Slika 31. Rezultat skripte `Izracunaj.ps1`

Funkcija u Command Prompt ljusci se definira stavljanjem dvotočke ispred naziva funkcije. Unosom opcije `setlocal`, funkcija naslijeđuje sve trenutne varijable iz skripte. Varijable se pozivaju stavljanjem naziva varijable između dvaju znakova `%`. Kvadrat zbroja je izračunan tako da je zbroj pomnožen sa samim sobom jer ne postoji predefinirana funkcija za računanje potencija, kao ni za korjenovanje. Unosom `cmd` u naredbeni redak PowerShell, pokreće se Command Prompt i skripta, `Izracunaj.bat`.

```
@echo off

set /p _num1=Prvi broj:
set /p _num2=Drugi broj:

if "%_num1%" == "" goto :eof
if "%_num2%" == "" goto :eof

:izracunaj

setlocal

set /a zbroj=%_num1%+%_num2%
set /a razlika=%_num1%-%_num2%
set /a umnozak=%_num1%/%_num2%
set /a kolicnik=%_num1%*%_num2%

echo Zbroj je %zbroj%  Razlika je %razlika%
echo Umnozak je %umnozak%  Kolicnik je %kolicnik%

set /a kvadrat=%zbroj%*%zbroj%
echo Kvadrat je %kvadrat%

endlocal
```

Slika 32. Aritmetičke operacije u CMD-u

```
PS C:\Users\Korisinik\Desktop> cmd
Microsoft Windows [Version 10.0.19042.1110]
(c) Microsoft Corporation. Sva prava pridržana.

C:\Users\Korisinik\Desktop>Izracunaj.bat
Prvi broj:13
Drugi broj:23
Zbroj je 36
Razlika je -10
Umnozak je 299
Kolicnik je 0
Kvadrat je 1296
C:\Users\Korisinik\Desktop>
```

Slika 33. Rezultat skripte `Izracunaj.bat`

## 4.5. Usporedba ljusaka sustava Windows i Linux

U ljusci Bourne je napisana jednostavna skripta koja provjerava je li trenutačni korisnik *root*, odnosno ima li administratorske ovlasti. UID korisnika *root* je `0`. Ako je UID `0`, ljuska ispisuje poruku `Jeste root`, a ako nije, ispisuje poruku `Niste root`.

```
#! /bin/sh

ROOT_UID=0

if [ "$UID" -eq "$ROOT_UID" ]
then
echo "Jeste root."
else
echo "Niste root"
fi
exit 0
```

Slika 34. `Bourne root.sh`

U ljusci PowerShell je provjereno ima li trenutni korisnik administratorske ovlasti. Za to je potrebno definirati trenutačnoga korisnika preko sustavskih varijabala `COMPUTERNAME` i `USERNAME`. Cmdletom `Get-LocalGroupMember` i unosom `Administrators` se obilježava objekt, a to je lokalna skupina `Administrators`. `Name` nakon zagrade obilježava samo nazive profila koji imaju administratorske ovlasti, dok opcija `contains` obilježava trenutno aktivnog korisnika, što je pohranjeno u varijabli `$korisnik`. Ako je istina da `$user` pripada skupini `Administrators`, tada ljuska ispisuje `Imate administratorska prava`.

```
$korisnik = "$env:COMPUTERNAME\$env:USERNAME"

$user = (Get-LocalGroupMember 'Administrators').Name -contains $korisnik

   if ($user = "True") {
      Write-Host "Imate administratorska prava"
   }

   else {
      Write-Host "Nemate administratorska prava"
   }
```

Slika 35. Funkcija `admin` – PowerShell

U ljusci `bash` je napisana skripta u kojoj se prelazi u kazalo `Resursi` na radnoj površini. Nakon upita korisnik u nosi korisničko ime, koje postaje novo ime za profil na računalu. To se radi naredbom `useradd`, a nova lozinka se stvara unosom naredbe `passwd` i korisničkoga imena računa za koji se stvara lozinka. Nakon toga korisnik unosi neki broj manji od `20` te se petljom `for` ispisuju sve brojeve od unesenoga do broja `19`.

```
#! /bin/bash
 cd /home/osboxes/Desktop/Resursi
  read -p "Korisničko ime : " username
          sudo useradd $username
          sudo passwd $username
read -p "Unesite broj manji od 20: " broj

for   ((x = $broj; x < 20; x++)); do
  echo $x
done
```

Slika 36. Stvaranje korisnika u ljusci `bash`

U Command Promptu se unos korisnika traži naredbom `set` i opcijom `/p` te definiranjem varijable u koju se pohranjuje unos korisnika. Naredba `net` i opcija `user` stvaraju novog korisnka. Novi korisnik je korisničko ime koje je unio korisnik i nakon toga se piše lozinka te opcija `/add`. Za ispis brojeva do `20` se unos korisnika pohranjuje u varijablu broj, na isti način kao i pri unosu korisničkoga imena. Za ispis svih brojeva u zadanom rasponu rabi se petlja `for`, a članovi petlje se poziva bilo kakvom biljegom, ispred koje moraju stajati znakovi `%%`.

```
@echo off

 set /p korisnik="Unesite novog korisnika "
net user %korisnik% lozinka456 /ADD

set /p broj="Unesite broj manji od 20: "

FOR /L %%i IN (%broj%,1,19) DO (
ECHO %%i
)
```

Slika 37. Stvaranje korisnika u Command Promptu

Ljuska PowerShell rabi *cmdlet* `Read-Host`, za unos podataka od korisnika. Potrebno prigodom definiranja lozinke, potrebno ju je promijeniti u siguran *string*. Ljuska PowerShell to čini naredbom `ConvertTo-SecureString`. Nakon toga se, naredbom `New-LocalUser` i opcijom `Name` i `Password` stvara nova lozinka.

```
$novi_korisnik = Read-Host "Unesite novo korisničko ime"
$nova_lozinka = Read-Host "Unesite novu lozinku"
 $pass
$pass= ConvertTo-SecureString -AsPlainText -Force -String $nova_lozinka

New-LocalUser -Name $novi_korisnik -Password $pass
```

Slika 38. PowerShell – stvaranje korisnika

U ljusci `tcsh` se unose i zbrajaju dva broja. Unos podataka se piše preko `echo` i opcije `n`, koja omogućuje korisniku da ostane u istom redu, a `set` <naziv varijable> i znak `$<` služi pohrani unosa u varijablu `$x`, odnosno `$y`. U varijablu `$odg` se pohranjuje operacija zbrajanja ranije definiranih varijabla. Kako bi se definirala, potrebno je ispred nje staviti znak `@` i razmak.

```
#! /bin/tcsh

#tenex - varijable, datumi, aritmetika

echo -n "Unesi prvi broj:"
set x = $<
echo -n "Unesite drugi broj:"
set y = $<
@ odg = $x + $y
echo "$odg"
```

Slika 39. Zbroj dviju varijabala u ljusci `tcsh`

Command Prompt prima unos naredbom `set` i opcijom `/p` dok se u varijablu c pohranjuje zbroj unesenih brojeva, koji je definiran naredbom `set` i opcijom `/A`, za pohranjivanje aritmetičkih operacija.

```
set /p br1="unesite prvi broj: "
set /p br2="unesite drugi broj: "


SET /A c = %br1%+%br2%
echo Zbroj %br1% i %br2% je %c%.
```

Slika 40. Zbroj dvije varijable u ljusci CMD

U ljusci PowerShell je unos korisnika uvijek *string* pa je potrebno promijeniti vrstu unosa, u ovom slučaju u *integer*, naredbom `[int]`. Definirana je varijabla `$zbroj`, uz koju nije potrebna opcija za pohranjivanje aritmetičkih operacija.

```
[int]$broj1= Read-Host -Prompt "Unesite prvi broj"

[int]$broj2= Read-Host -Prompt "Unesite drugi broj"

$zbroj = $broj1 + $broj2

Write-Host "Zbroj je $zbroj"
```

Slika 41. Zbroj dvije varijable u PowerShellu

U skripti `fish` je definirana funkcija `nekaFunkcija` koja provjerava je li `$varijabla` definirana. To je provjereno narebom `test` i opcijom `z`. Ako nije, ispisuje da nije prazna, a u protivnom ispisuje svaki broj od `12` do `17` kroz petlju `for`.

```
#! /usr/bin/fish

function nekaFunkcija
if test -z $varijabla
echo "Varijabla je prazna.";
 else;
 echo "Varijabla nije prazna.";
end
for i in (seq 12 17); echo "$i"; end
 end

nekaFunkcija
```

Slika 42. Funkcija `nekaFunkcija` u ljusci `fish`

Ista skripta je napisana u ljusci Windows PowerShell. Definirana je funkcija koja provjerava je li varijabla jedanka `$null`, što znači da je prazna. Funkcija vraća ispis `Varijabla ni/je prazna`, ovisno o tome je li navedena varijabla prazna, što znači da joj nije dodijeljena vrijednosti ili nije.

```
function funkcija {
   if ($varijabla -eq $null) {
   return Write-Host "Varijabla je prazna."
   }
   else {
     return Write-Host "Varijabla nije prazna"
    }
   }
   funkcija
```

Slika 43. Primjer funkcije u ljusci PowerShell

Napisana je skripta u ljusci Korn koja traži korisnika da upiše neki broj. Za upis se rabi naredba `read` i pohranjuje primljeni odgovor u varijablu `$answer`. Dokle god `$a` nije veći od unesenoga broja ljuska ispisuje rezultat operacije `$a + 1`.

```
#! /bin/ksh

a=10
print -n "Unesite neki broj veći of 10: "
   read answer

until [[ $a > $answer ]];do

  a=$(expr $a + 1)
   echo $a

done
```

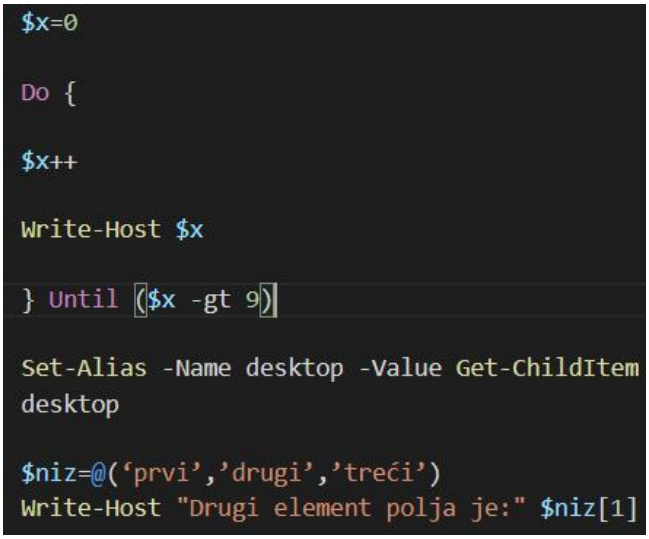
Slika 44. petlja `until` u ljusci Korn

U ljusci Z je postavljen *alias* za ispis svih datoteka koje imaju nastavak `.sh` te ispis trećega člana niza.

```
#!  /bin/zsh

alias trazi="find . -name \*.sh -print"

trazi

niz=("file" "tekst" "dokument" "skripta")

echo $niz[2]
```

Slika 45. `zsh` – *alias*, ispis člana niza

Unutar ljuske PowerShell postoji petlja `Do Until`, kojom se ispisuje i varijabla `$x`, počevši od `0`, sve dok `$x` nije veći od `9`. Postavljanje *aliasa* se postiže naredbom `Set-Alias`. Niz se postavlja definiranjem naziva niza i pisanja elemenata između oblih zagrada, ispred kojih stoji znak `@`.




Slika 46. PS – petlja `Until`, postavljanje *aliasa*

# 5. Preporuke za izbor ljuske

## 5.1. Preporuke za Linux

U operacijskom sustavu Linux OpenSUSE 15.1 na virtualnom stroju VirtualBox pregledane su najznamenitije ljuske u sustavima Unix/Linux na temelju skriptnih primjera, u kojima su uspoređeni razni programski konstrukti, poput petalja, pozivanja i postavljanja varijabala i naredaba. Mogućnosti ljuske prigodom navigacije, komprimiranja datoteka, obavljanja jednostavnih aritmetičkih operacija, ispisa teksta i slično. Ljuske uspoređene unutar sustava Linux su Bourne, `bash`, Korn, C/Tenex C, `fish` i `zsh`. Svaka ljuska ima zasebna pravila skriptiranja i navigacije naredbenim retkom i svaka ima različit opseg mogućnosti. To je najizraženije prigodom porabe naredbenoga retka (predviđanje naredaba, dostupnost prošlih naredaba, postavljanje kratica…), a u velikoj se mjeri osjeti i prigodom skriptiranja.

S obzirom na to da su u ovom radu uspoređivani skriptni primjeri, težište će biti na ljuskama u kontekstu skriptiranja. Najrabljenija ljuska kroz većinu sustava Linux i Mac je `bash`, ali ga u proteklih nekoliko godina u sve većoj mjeri zamjenjuje ljuska Z. Neovisno o tom, najviše izvora je napisano za ljusku `bash`. Za početnike u skriptiranju ova ljuska je dobar izbor, upravo zbog količine dokumentacije dostupne preko knjiga i stručne literature te raznih članaka, koji pomažu pri prilikom učenja ljuske `bash`. Uz to, ljuska `bash` je većinski prilagođena standardu POSIX i prenosiva kroz razne distribucije. U ljusci `bash` su dostupne vrlo moćne naredbe, poput `passwd` te dinamičan skriptni jezik koji omogućuje izravno i jednostavno pisanje skripata. Ljuska `bash` je u potpunosti kompatibilna sa ljuskom Z, a po pitanju skriptiranja se razlikuju u nijansama.

Ljuska Z je, prigodom izvođenja složenijih programa brža u odnosu na ljusku `bash`. Kod skriptiranja `repeat` riječ zamjenjuje petlju `for` i slično. Glavna prednost ljuske Z je širok opseg dodataka koje ljuska nudi, posebno unutar naredbenoga retka te niz korisnika koji prinose razvoju ljuske. Skriptiranje u ljusci `fish` je vrlo intuitivno, a sama ljuska je brza prigodom izvršavanja skripata. Ova ljuska je dobar izbor za korisnike koji nemaju potrebu mijenjati radno okruženje, posjeduju znanje i iskustvo u radu naredbene ljuske, s obzirom da je na raspolaganju manje dostupnih izvora za rad s ovom ljuskom. Skripte se pišu intuitivnije i jednostavnije od bilo koje druge. Nema potrebe za brisanjem razmaka između znaka jednakosti ili pisanja zagrada kod postavljanja uvjeta u izrazu `if`.

## 5.2. Preporuke za Windows

U operacijskom sustavu Windows 10 su uspoređene ljuske PowerShell (inačica 5. 1) i Command Prompt. Ove dvije ljuske se u velikoj mjeri razlikuju u pogledu skriptiranja te efektivnosti. PowerShell je snažnija u odnosu na Command Prompt, s intuitivnijim skriptnim jezikom i naredbama koje su dulje, ali se lakše pamte te postoje ugrađene kratice. Za skriptiranje složenijih skripata i potrebe prijenosa poslova na drugi sustav rabi se ljuska PowerShell. Za potrebe pisanja jednostavnijih skripata ili naredaba, obje ljuske su korisne. PowerShell proširuje kapacitete ljuske ugradnjom metoda i razreda programskoga okvira .NET. Skriptiranje je jednostavnije od Command Prompta, razvijena je okolina za skriptiranje, a ljuska je prenosiva između sustava.

Međutim, pisanje jednostavnih skripata, zbog prirode PowerShella oduzima više vremena te postane zamorno pisati dulje retke koda za neku jednostavnu radnju. Command Prompt posjeduje nekoliko odličnih programskih rješenja, primjerice, jednostavan način na koji rabi petlju `for` za prolazak kroz kazalo, unos korisnika je vrlo jednostavan te ne dodjeljuje vrstu podatka, nego ju definira prema unosu. Ljuska Command Prompt je dobro oruđe za pisanje jednostavnih skripata i svakodnevnu porabu naredbenoga retka, dok je ljuska PowerShell vrlo moćan odgovor na zahtjevnije probleme skriptiranja.

# 6. Zaključak

Ovaj rad opisuje znamenite naredbene ljuske na klasičnim distribucijama komercijalnih operacijskih sustava ili onih otvorenoga koda. U izradbi su rabljene knjige i internetski članci, a u praktičnom dijelu su rabljeni virtualni strojevi u sustavu za virtualizaciju VirtualBox. Prvo poglavlje je posvećeno stavljanju naredbene ljuske u kontekst operacijskoga sustava, s ciljem naglašivanja njezine važnosti u procesu interakcije čovjeka i računala. Opisana je generična naredbena ljuska i svi članovi te razvoj i sposobnosti većine znamenitih naredbenih ljusaka. U prvom dijelu rada su opisane najznamenitije naredbene ljuske u sustavima Linux i Windows, njihov razvoj i mogućnosti. U praktičnom dijelu rada su međusobno uspoređene naredbene ljuske u sustavu Linux. Na jednostavnim skriptnim primjerima je prikazana skladnja i mogućnosti ljusaka.

U Windows sustavu su uspoređene skripte ljuske Command Prompt i PowerShell, kako bi se demonstrirale mogućnosti obiju ljusaka. U trećem i posljednjem dijelu su navedene preporuke za odabir ljuske unutar distribucija Linux i Windows. Preporuke se razlikuju, ovisno o potrebama i zanimacijama korisnika. Za osobnu primjenu se preporučuje ljuska `fish` unutar operacijskoga sustava Linux, dok se za profesionalnu primjenu preporučuje ljuska `bash` ili Z. Za početnike u skriptiranju se preporučuje ljuska `bash`, s obzirom na dostupnost dokumentacije i brojnost korisnika. U operacijskom sustavu Windows je situacija malo jednostavnija, s obzirom na manji broj ljusaka, od kojih su danas u porabi samo dvije. PowerShell se preporučuje za potrebe u administracije u sustavima Windows, za pisanje složenijih skripata, zbog vrlo snažnoga skriptnoga jezika te dostupnosti na distribucijama Mac i Linux (Pedamkar, P., bez dat.). CMD se preporučuje za pisanje jednostavnih skripta i jednostavnih, a učinkovitih naredba.

# Popis literature


1. Burk, R., Horvath D. B., (1997.) *UNIX Unleashed: System Administrator's Edition, Sams;* (2nd edition), preuzeto 7. 4. 2021. S http://linuxclass.heinz.cmu.edu/misc/shell-comparison.htm

2. Foster-Johnson, E., Welch J. C., Anderson M., (2005.), Beginning Shell Scripting, Indianapolis, SAD, Wiley Publishing, Inc.

3. Fox, R. (2014.) Linux with operating system concepts, Chapman and Hall/CR, New York, USA

4. Jones, M., (2011.), Evolution of shells in Linux, preuzeto 11. 4. 2021. S https://developer.ibm.com/technologies/linux/tutorials/l-linux-shells/

5. Parker, S., (2011.), Shell Scripting, Indianapolis, SAD John Wiley & Sons, Inc.

6. Red Hat Inc., (bez dat.) What is the Linux kernel, Preuzeto 28.03.2021. s https://www.redhat.com/en/topics/linux/what-is-the-linux-kernel

7. Robbins, A., Beebe, N. (2005.) Classic shell scripting, Sebastopol, SAD, O'Riley Media Inc.

8. Eckert, J. W., (2021.), Talk Tech to Me: 5 Z Shell Features That Will Make You Want to Switch from Bash, preuzeto 17. 4. 2021. s https://www.comptia.org/blog/linux-and-z-shell

9. Falstad, P., de Baker B., (1995.) , An Introduction To The Z Shell, Preuzeto 20. 4. 2021. s https://www.ee.ryerson.ca/guides/zsh-intro.pdf

10. Keel, S. A., (2021.), What is the Bourne Shell? , preuzeto 4. 5. 2021. s https://www.easytechjunkie.com/what-is-the-bourne-shell.htm

11. Pfaff, B., (1999.), Bourne Shell Programming in One Hour, preuzeto 5. 5. 2021. s https://web.stanford.edu/~blp/writings/shell/shell.pdf

12. Stanek, W. R, (2004.), Microsoft Windows Command-Line Administrator's Pocket Consultant, Redmond, Washington, SAD, Microsoft Press

13. Asati, A. (bez dat.) Batch Scripting Commands, preuzeto 8. 5. 2021. s https://www.educba.com/batch-scripting-commands/

14. Holmes, L., (2013.), Windows PowerShell Cookbook, O'Reilly Media, Inc., Sebastopol, Kalifornija, SAD

15. Linux Hint, (bez dat.), Posix Standard, preuzeto 22. 5. 2021. s https://linuxhint.com/posix-standard/

11. Robinson, S., (bez dat.), *Zsh vs Bash*, preuzeto 18. 4. 2021. s https://stackabuse.com/zsh-vs-bash/

*12.* Jha, S. (bez dat), *Ubuntu terminal – dpkg list command output, preuzeto 9. 8. 2021. S https://www.shaileshjha.com/step-by-step-how-to-install-and-uninstall-google-chrome-in-ubuntu/ubuntu-terminal-dpkg-list-command-output/*

13. Joy, M., Jarvis, S., Luck, M., (2002.) *Introducing UNIX and Linux, New York, SAD,* Palgrave Macmillan Ltd.

14. Arensburger, A., (2005.), *Bourne Shell Programming,* preuzeto 14. 5. 2021. s https://neo.dmcs.pl/pios/sh-talk-text.pdf

15. IBM, (bez dat.), *Predefined special variables in the Bourne shell,* preuzeto 15. 5. s https://www.ibm.com/docs/en/aix/7.1?topic=shell-predefined-special-variables-in-bourne

16. Microsoft, (2021.) What is Powershell? Preuzeto 10. 5. 2021. s https://docs.microsoft.com/en-us/powershell/scripting/overview?view=powershell-5.1

22. Dent, C. (2019.), Mastering Windows Powershell Scripting, Pact Publishing Ltd., Birmingham, UK

23. Joy, W., et. al (2001.), *TCSH*, preuzeto 11. 5. 2021. s *https://linux.die.net/man/1/tcsh*

24. Joy, W., et. al., *A Introduction to the C shell,* preuzeto 12. 5. 2021. s https://docs.freebsd.org/44doc/usd/04.csh/paper-2.html#section15

25. Microsoft, (2006.), *Windows PowerShell (Monad) Has Arrived*, preuzeto 27.5.2021. s https://devblogs.microsoft.com/powershell/windows-powers*hell-monad-has-arrived/*

26. Robbins, A., Rosenblatt, B., (2002.), *Learning the Korn Shell*, 2nd Edition, O'Rilley, dostupno na: https://docstore.mik.ua/orelly/unix/ksh/appa_02.htm

27. Budin, L., Golub M., (2010.), et al., *Operacijski sustavi,* Zagreb*,* Element

28. Bhardwaj, P. K., (2006.) *How to Cheat at Windows System Administration using command line scripts,* Syngress Publishing, Inc., Rockland, MA, SAD

29. Ramey, C., (2020.), *Bash Reference Manual*, Free Software Foundation preuzeto 19. 5. 2021. s https://www.gnu.org/software/bash/manual/bash.pdf

30. Olczak, A., (2000.), Korn Shell*: Unix and Linux Programming Manual*, (Third Edition), Edison Weasley Publishing

31. Gillis, A. S., command line interface (CLI), Preuzeto: 28.03.2021. s https://searchwindowsserver.techtarget.com/definition/command-line-interface-CLI

32. Menéndez, M., (2015.), How To Install Fish, A Smart Command Line Shell for Gnu/Linux/Hurd and *BSD, preuzeto 21. 4. 2021. s https://miguelmenendez.pro/en/blog/install-fish-smart-command-line-shell-gnu-linux-hurd-bsd/

33. Pedamkar, P.,(bez dat.) *.NET Core vs .NET Framework,* preuzeto 10. 5. 2021. s https://www.educba.com/dot-net-core-vs-dot-net-framework/?source=leftnav

34. University of Hawai, (2001.) *The C Shell tutorial*, preuzeto 11. 5. 2021. s https://alapono.eng.hawaii.edu/Tutor/csh.html

35. Fish-shell, (bez dat.), *Design, preuzeto 12. 5. 2021.* dostupno na https://fishshell.com/docs/current/design.html

36. Delgado, C.(2017.) *How to use multiple tabs feature in th Kali Linux terminal* preuzeto 8.8.2021. s https://ourcodeworld.com/articles/read/487/how-to-use-the-multiple-tabs-feature-in-the-kali-linux-terminal

37. Chandio, F., (2020.) How to Access Command History in Command Prompt on Windows 10, Preuzeto 9.8. 2021. s https://wccftech.com/how-to/how-to-access-command-history-in-command-prompt-on-windows-10/

38. Kirasić, D., *Shell programiranje*, Škola otvorenog računarstva, FER, Zagreb

39. Fish-shell, bez dat, Tutorial, preuzeto 12. 5. 2021., s https://fishshell.com/docs/current/tutorial.html

40. Oh-my-zsh, bez.dat, preuzeto 26.7. 2021. s https://ohmyz.sh

# Popis slika

# Popis tablica