



Admin Training
Center

Linux L1

Osnove Linux-a

Veselin Mijušković, Ljubiša Radivojević, Marko Uskoković



Admin Training
Center

VESELIN MIJUSKOVIĆ, LJUBIŠA RADIVOJEVIĆ, MARKO USKOKOVIĆ

Linux L1-1

Osnove Linux operativnog sistema

Admin Training Center
Studentski trg 4, VI sprat
11000 Beograd, Srbija
<http://www.atc.rs/>



Copyright ©2013 Veselin Mijušković, Ljubiša Radivojević, Marko Uskoković

Ovaj tekst je licenciran pod Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License. Da biste videli kopiju ove licence posetite:
http://creativecommons.org/licenses/by-nc-nd/3.0/deed.en_US.

Sadržaj

| | |
|---|-----------|
| Uvod - tipografske konvencije | 5 |
| Tipografske konvencije..... | 5 |
| Upoznavanje sa filozofijom open-source pokreta..... | 7 |
| Šta je Linux? | 7 |
| Šta je operativni sistem? | 7 |
| Šta je Unix? | 7 |
| Šta je GNU? Šta je Open Source? | 8 |
| Šta je Minix?..... | 9 |
| Šta su Linux distribucije? | 9 |
| Primena Linuxa | 10 |
| Pitanja i zadaci | 10 |
| Upoznavanje sa osnovama na kojima je zasnovan Linux .. | 11 |
| Fajlovi..... | 11 |
| Imena fajlova..... | 12 |
| Puna imena fajlova..... | 12 |
| Vlasništvo nad fajlovima | 12 |
| Osnovna prava pristupa nad fajlovima..... | 12 |
| Hijerarhijska struktura stabla direktorijuma..... | 13 |
| Posebni fajlovi . i .. | 14 |
| Tačke kačenja (mount points)..... | 14 |
| Procesi | 14 |
| Softverski alati | 15 |
| Pitanja i zadaci | 15 |
| Pristup Linux-u, terminali i SSH..... | 17 |
| Konzolni pristup..... | 17 |
| Mrežni pristup..... | 17 |
| Autentifikacija | 18 |
| Pitanja i zadaci | 18 |
| Rad u komandnoj liniji | 19 |
| Komandni interpreter..... | 21 |
| Pomoć u komandnoj liniji..... | 21 |
| Kretanje po komandnoj liniji..... | 23 |
| Pitanja i zadaci | 24 |
| Osnovne komande za rad sa fajlovima..... | 25 |
| ls – listanje sadržaja direktorijuma | 25 |
| cp – kopiranje fajlova i direktorijuma..... | 28 |
| mv – prebacivanje i preimenovanje fajlova i direktorijuma | 29 |
| rm – brisanje fajlova i direktorijuma | 29 |
| ln – kreiranje linkova..... | 29 |
| file – dobijanje informacija o sadržaju fajla | 30 |
| type – dobijanje informacija o komandama..... | 30 |
| find – pretraživanje direktorijuma i nalaženje fajlova | 30 |
| Pitanja i zadaci | 33 |

| | |
|---|-----------|
| Rad sa direktorijumima | 35 |
| cd – kretanje kroz stablo direktorijuma | 35 |
| pwd – ispis tekućeg direktorijuma | 35 |
| pushd, dirs i popd – manipulacija stekom direktorijuma | 35 |
| mkdir – kreiranje direktorijuma | 36 |
| rmdir – brisanje praznih direktorijuma | 36 |
| Pitanja i zadaci..... | 37 |
| Regularni izrazi | 39 |
| Literali..... | 39 |
| Metaznaci | 40 |
| <i>Metaznaci u osnovnim POSIX regularnim izrazima.....</i> | <i>40</i> |
| <i>Metaznaci kod proširenih POSIX regularnih izraza</i> | <i>41</i> |
| <i>Klase znakova.....</i> | <i>41</i> |
| Mečovanje regularnih izraza | 42 |
| Pitanja i zadaci..... | 43 |
| Komande za rad sa sadržajem fajlova | 45 |
| cat – ispis sadržaja fajla | 45 |
| more i less – ispis sadržaja fajla stranu po stranu | 45 |
| head – ispis početka fajla..... | 46 |
| tail – ispis kraja fajla..... | 46 |
| wc – brojanje znakova, reči i linija u tekstu..... | 46 |
| sort – sortiranje linija u fajlu..... | 47 |
| uniq – izbacivanje identičnih linija iz sortiranog teksta..... | 48 |
| grep – pretraživanje sadržaja tekstualnih fajlova | 49 |
| split – razbijanje fajlova u manje delove | 50 |
| gzip i bzip2 – kompresovanje fajlova..... | 50 |
| tar – pravljenje arhive..... | 51 |
| cut – ispis određenih delova svake linije formatiranog teksta | 52 |
| tr – zamena znakova ili grupa znakova u tekstu..... | 53 |
| sed – neinteraktivni editor teksta..... | 54 |
| awk – razne manipulacije tekstom | 55 |
| diff – nalaženje razlika među fajlovima..... | 55 |
| Pitanja i zadaci..... | 55 |
| Redirekcija I/O i pajpovi | 57 |
| Standardni ulaz, standardni izlaz i standardni izlaz za greške..... | 57 |
| Redirekcija ulaza, izlaza i izlaza za greške komande | 57 |
| Pajpovi..... | 58 |
| Pitanja i zadaci..... | 58 |
| Korisnici i grupe | 59 |
| id – dobijanje informacija o korisniku..... | 60 |
| passwd – menjanje korisnikove lozinke..... | 60 |
| newgrp – promena aktivne grupe..... | 61 |
| chgrp – promena grupe koja je vlasnik fajla | 61 |
| Pitanja i zadaci..... | 61 |
| Prava pristupa - detaljnije | 63 |
| Osnovna prava pristupa..... | 63 |
| Specijalna prava pristupa..... | 65 |
| umask – podešavanje podrazumevanih prava pristupa..... | 66 |
| chmod – promena prava pristupa nad fajlovima..... | 68 |
| Pitanja i zadaci..... | 69 |
| Rad sa procesima | 71 |

| | |
|--|-----------|
| Procesi u pozadini | 71 |
| Izvršavanje niza komandi | 72 |
| Upravljanje procesima | 73 |
| Pitanja i zadaci | 74 |
| Rad sa vim editorom | 75 |
| Pokretanje vim editora..... | 76 |
| Modovi vim editora | 76 |
| <i>Komandni mod</i> | 76 |
| <i>'ex' mod</i> | 77 |
| <i>Modovi za unos teksta</i> | 77 |
| Pomoć u vim editoru..... | 77 |
| Snimanje, menjanje fajla i izlazak iz vim editora | 78 |
| Kretanje kroz tekst u komandnom modu | 79 |
| Višestruki unos | 80 |
| Brisanje teksta..... | 80 |
| Zamena teksta | 81 |
| Selektovanje, kopiranje i lepljenje teksta | 81 |
| Pretraživanje i zamenjivanje teksta | 82 |
| Ostale često korišćene komande..... | 83 |
| <i>Unos fajla</i> | 83 |
| <i>Izvršavanje eksterne komande nad delom teksta</i> | 83 |
| <i>Spajanje dve linije u jednu</i> | 83 |
| <i>Promena kapitalizacije slova</i> | 84 |
| <i>Ponavljanje prethodne komande</i> | 84 |
| <i>Poništavanje prethodne komande</i> | 84 |
| Pitanja i zadaci | 84 |
| Odgovori na neka pitanja i rešenja nekih zadataka | 85 |
| Upoznavanje sa filozofijom open-source pokreta..... | 85 |
| Upoznavanje sa osnovama na kojima je zasnovan Linux | 85 |
| Pristup Linux-u, terminali i SSH | 86 |
| Rad u komandnoj liniji | 86 |
| Osnovne komande za rad sa fajlovima | 87 |
| Rad sa direktorijumima..... | 87 |
| Regularni izrazi | 87 |
| Komande za rad sa sadržajem fajlova..... | 89 |
| Redirekcija I/O i pajpovi..... | 89 |
| Prava pristupa – detaljnije | 89 |
| Rad sa procesima | 89 |

Uvod - tipografske konvencije

Poštovani polaznici, pred vama je prateća skripta za ATC kurs Linux L1-1: Osnove Linux-a. Ova skripta je namenjena kao dodatak predavanjima i kao podsetnik za najčešće upotrebe nekih komandi. Ona će biti u 'perpetualnom razvoju' tako da je svaka povratna informacija od strane vas (ispravke grešaka, upozorenja da nešto nije dovoljno detaljno ili dovoljno jasno objašnjeno i sl.) poželjna.

Skripta je podeljena na poglavlja, a ona na sekcije. Na kraju svakog poglavlja se nalaze pitanja i zadaci, od kojih neki imaju navedene odgovore. Polaznicima se sugerise da sami pokušaju odgovoriti na pitanja i rešiti zadatke, pre nego što pogledaju odgovor.

Takođe, unapred skrećemo pažnju polaznicima da je ova skripta samo deo dokumentacije koju oni treba da koriste, posebno kada se radi o komandama. Polaznicima se savetuje da pročitaju man i help strane za svaku komandu, kao i da potraže na Internetu dodatne informacije i načine kako da iste korsite.

Tipografske konvencije

Radi lakšeg snalaženja u tekstu, koristili smo neke tipografske konvencije na koje vam ovde skrećemo pažnju:

- ukoliko se uvodi neki značajan pojam, on će u prvom pomenu biti ispisan **proporcionalnim bold** tekstom;
- boldovano su prikazane i neke **značajne tvrdnje** na koje treba obratiti pažnju u tekstu;
- *proporcionalnim italikom* su napisane reči na stranom jeziku, najčešće engleskom;
- nazivi programa, opcija i fajlova u tekstu su ispisani neproporcionalnim fontom

Pored standardnog teksta ova skripta sadrži i neke primere pozivanja komandi i prikaz rezultata tih komandi. Takav ispis je prikazan u zasebnom bloku, kao što je ovaj:

```
$ ls -F  
myscript.sh*  Vezbe/
```

Boldovanim tekstom je prikazano ono što polaznik treba da unese onako kako je napisano u skripti. Regularnim tekstom je prikazan ispis programa koji ne treba unositi.

Isti metod se koristi i za prikaz formata komandi i njihovih parametara:

```
$ cp [opcije] source... dest
```


U ovom slučaju je boldovano prikazana sama komanda i tako je treba uneti. Italikom su prikazani promenljivi parametri i to znači da na tom mestu treba uneti neku stvarnu vrednost, a ne ono što piše u komandnoj liniji. Na kraju, opcionih parametara, koji se mogu izostaviti su navedeni u srednjim zagradama []. Naravno, zagrade ovde služe samo kao naznaka i ne unose se! Tri tačke (...) označavaju da je dati parametar moguće navesti više puta.

Gde god smo mislili da nešto posebno treba naglasiti, to smo naveli u zasebnom boksu, koji obično ima naslov 'Važno!' ili 'Napomena', kao ovde:

Važno!

Hard linkovi ne mogu biti kreirani za direktorijume, već samo za regularne fajlove!

Upoznavanje sa filozofijom open-source pokreta

Šta je Linux?

Linux je Unix-oliki operativni sistem otvorenog koda distribuiran pod GNU GPL licencom.

Šta je operativni sistem?

Operativni sistem je skup programa koji imaju za cilj da korisniku omoguće korišćenje računara. Računar bi bez postojanja operativnog sistema bio samo gomila neiskorišćenih elektronskih kola. Operativni sistem je onaj deo računarskog sistema koji ima za cilj iskorišćavanje mogućnosti koje hardver pruža, nadoknađivanje nepostojećih ili neisplativih mogućnosti i olakšavanje korišćenja istih. Operativni sistem pruža korisniku jednostavan pristup hardverskim uređajima i brine o tome da se uređaji koriste na predviđen način. On izvršava programe koje korisnik pokreće i upravlja njima tako da se više programa može izvršavati u isto vreme.

Pod operativnim sistemom se danas ne podrazumeva samo jezgro (deo koji upravlja hardverom), već i skup programa koje korisnik pokreće, uključujući komandni interpreter (shell), grafički interfejs, programe za obradu teksta, slanje elektronske pošte i sve ostalo što korisnik koristi u svakodnevnom radu.

Danas postoji zaista veliki broj operativnih sistema. Oni se razlikuju po mogućnostima, kvalitetu i nameni. Posmatrajući koliko se jedan mainframe računar razlikuje od pametnog mobilnog telefona, lako se dolazi do zaključka da je ta raznolikost i mnogobrojnost neophodna, čak i dobra. Korisniku se tako omogućava da bira sistem pod kojim želi da radi, funkcije koje mu operativni sistem pruža, da li želi da iza njega stoji neka velika firma i koliko želi novca da utroši na kupovinu i održavanje operativnog sistema.

Šta je Unix?

Unix je jedan od najkompletnijih operativnih sistema koji su ikada napravljeni. U njemu su prvi put implementirane mnoge mogućnosti bez kojih bi rad današnjih računara i računarskih mreža bio nezamisliv. Unix je prvi operativni sistem koji je bio napisan u nekom programskom jeziku višeg nivoa (programskom jeziku C, koga je razvio Dennis Ritchie, baš za potrebe pisanja Unix-a), čime je stvorena mogućnost izvršavanja istog operativnog sistema na različitim hardverskim platformama. Popularnost je stekao kako u komercijalnim oblastima, tako i na univerzitetima.

Prvu verziju operativnog sistema UNIX razvio je 1969. godine Ken Thompson član Bell-ove laboratorije unutar firme AT&T sa ciljem da napravi pogodan timesharing sistem. Godine 1976. počinje razvoj Unix-a na Berkeley Univerzitetu i od tada postoje, generalno posmatrano, dve osnovne familije. Unix-a: System V Unix (razvijan komercijalno unutar Bell-ovih laboratorija) i BSD Unix (razvijan kao Berkeley Software Distribution).

Danas je Unix registrovano ime (trademark) firme Open Group (<http://www.opengroup.org>) koja je objavila Single UNIX Specification, spisak standarda koji proširuju IEEE-ove POSIX standarde. Da bi jedan operativni sistem mogao da nosi ime UNIX, on mora da prođe kroz sertifikaciju da zadovoljava te standarde. Takav jedan proces sertifikacije nije besplatan, što implicira komercijalnost Unix sistema.

Šta je GNU? Šta je Open Source?

Kako je računar bez operativnog sistema gotovo beskoristan, a kvalitetni Unix sistemi su ograničeni svojim komercijalnim licencama u pogledu načina upotrebe i ljudi koji mogu da ih novčano priušte, postavlja se pitanje da li je (kvalitetno) korišćenje računara uopšte moguće bez takvih komercijalnih sistema? Odgovor je: DA, a jedan od najubedljivijih ljudi u to, još osamdesetih godina prošlog veka, bio je Richard Stallman, kome su smetala ograničenja koje su mu postavljale komercijalne licence pod kojima je softver tog vremena bio objavljivan, pa se odlučio da napravi kompletan operativni sistem koji će ljudi moći slobodno da koriste, prepravljaju, dele sa drugima, zajedno razvijaju i unapređuju. Tako je nastao projekat GNU. Ime GNU je skraćenica od GNU's Not Unix, čime se nagoveštava da je rađen po ugledu na UNIX, da sadrži sve što i standardan Unix sistem, ali da u sebi ne sadrži ni jednu liniju izvornog koda Unix-a.

Richard Stallman je osnovao Free Software Foundation (FSF) i sastavio GNU General Purpose Licencu (GNU GPL) pod kojim je čitav GNU sistem licenciran. Mnogi programeri su mu se pridružili i podržali ga u tome, tako da je danas GNU sistem gotovo kompletan: on sadrži sistemске programe, sistemске biblioteke, korisničke programe, razvojne alate, pomoćne alatke, igre i kompletnu dokumentaciju celog sistema. Jedina komponenta koja nije (kompletno) dovršena je kernel, jezgro operativnog sistema. Za više detalja o GNU projektu pogledati www.gnu.org.

Cilj GNU projekta je da korisnicima pruži slobodu da izvršavaju, kopiraju, prepravljaju, popravljaju i distribuiraju programe. Jedina stvar u vezi koje GNU GPL ograničava pojedinca je da te slobode uskrati drugima. Niko ne sme GPL licenciranom projektu da zatvori kod, čak ni autor. Modifikovane verzije programa moraju naslediti GNU GPL licencu. Prepravke programa su dobrodošle i najčešće ih originalni autor uključuje u svoju "zvaničnu" verziju. Na taj način program razvija, ne jedan čovek, ne tim od nekoliko programera, već stotine i hiljade programera i korisnika programa i oni zajedno pronalaze nedostatke i greške programa, doprinose novim idejama i mogućnostima.

Open Source je malo širi pojam od GNU GPL. Mnogi proizvođači programa su želeli da otvore svoj kod, da ga daju na uvid korisnicima i programerima, radi unapređivanja istog, ali nisu želeli da koriste GPL, obzirom da je toliko slobodan. Tako su nastale restriktivnije licence, koje npr. zabranjuju da modifikovane verzije programa nose isto ime, ali koje ipak daju dovoljno sloboda korisnicima da bi se mogle smatrati otvorenim. Danas postoji veliki broj Open Source licenci (pogledati www.opensource.org)

Zanimljivo je da GNU GPL ništa ne kaže u vezi novca. Iako nosi ime "free software", program pod GNU GPL licencom se može naplaćivati (često se citira "free as in freedom, not as in free beer"). Bilo ko je slobodan da skupi dovoljno programa na nekom medijumu i da ih za novčanu nadoknadu distribuira drugima. Ono što nije dozvoljeno je da drugima ne dopusti da to rade bez nadoknade ako žele. Pa zašto bi onda neko platio nekom distributeru za software koji može da ga dobije besplatno? Zato što distributer takođe nudi i tehničku podršku, piše uputstva, štampa priručnike, u krajnjem slučaju, oslobađa korisnika muke izabiranja, kompajliranja i sastavljanja kompletnog sistema.

Šta je Minix?

Na mnogim Univerzitetima širom sveta se Unix koristio kao primer dobro dizajniranog operativnog sistema, ali je generalno bio suviše glomazan da bi u potpunosti bio izučavan na časovima. Upravo zbog toga je Andrew S. Tanenbaum napisao Minix, funkcionalan Unix-oliki operativni sistem za PC, koji je ipak bio dovoljno jednostavan da bi mogao kompletno da bude jasan studentima kompjuterskih nauka. Za više informacija pogledati www.cs.vu.nl/~ast/minix.html

Šta su Linux distribucije?

Kada se kaže Linux, misli se prvenstveno na kernel, jezgro operativnog sistema. Kernel je svakako značajan deo operativnog sistema, ali je praktično neupotrebljiv bez dodatnih programa. Linux kernel se prvenstveno koristi u okviru distribucije, tj. skupa sistemskog i korisničkog softera, tako da se dobije kompletan operativni sistem. Najčešće se Linux kernel pakuje zajedno sa GNU alatima i bibliotekama, kao i nekim drugim open-source programima kako bi se dobio Unixoliki operativni sistem, koji se naziva distribucija. Međutim, najpopularniji sistem koji koristi Linux kernel nije baziran na ovim alatima – u pitanju je Android, najpopularniji i najrašireniji operativni sistem za pametne telefone, kojeg je, na osnovi Linux operativnog sistema, razvila kompanija Google.

Zadatak Linux distribucije je da korisnik na jednom mestu može naći konzistentan sistem, sve potrebne programe, neophodne biblioteke i dokumentaciju. Linux distribucija može uključiti svoje alatke za podešavanje sistema, može organizovati svoj packaging sistem kojim se održavanje instaliranih programa i upgrade sistema lakše izvodi, ili može biti zasnovana na nekoj drugoj distribuciji ali se specijalizirajući za neke određene namene. Tako imamo distribucije okrenute desktop nameni, serverske distribucije, distribucije namenjene za specifične zadatke kao što su ruteri, wifi uređaji i sl.

Distribucije mogu održavati pojedinci, firme ili neprofitne organizacije, tako da korisnik može da bira između distribucija na osnovu svojih preferencija, namene, iskustva, novca koji želi da plati, količine sistemske podrške koje dobija od distributera.

Važno je shvatiti da gotovo sve što se može uraditi sa jednom distribucijom, može se postići i sa bilo kojom drugom, nekim možda malo drugačijim (ili težim) postupkom.

RedHat (www.redhat.com) je firma koja tradicionalno proizvodi kvalitetne distribucije. Trenutno održava dve linije: RedHat Enterprise Linux i Fedora Project.

Debian (www.debian.org) je neprofitna organizacija čija se distribucija Linux-a od samog osnivanja smatrala jako dobrim izborom za servere.

Ubuntu (www.ubuntu.com) je novija distribucija opšte namene, koja se, zahvaljujući dobrom razvojnom timu i time što je zasnovana na starijem Debian-u, dobro iskorišćava za sve namene, serverske, desktop i enterprise.

Za informacije o bilo kojoj distribuciji posetite www.distrowatch.com, sajt posvećen praćenju svih Linux distribucija, njihovih mogućnosti i paketa koje sadrže.

Primena Linuxa

Linux sistem je izuzetno upotrebljiv kao serverski operativni sistem, kao razvojna platforma (zahvaljujući ogromnom broju razvojnih alata), biznis platforma (zbog velikog broja Office paketa izuzetne upotrebljivosti) ili kao radna stanica. Linux je veoma popularan među onima koji prave razne računarski upravljane uređaje posebne namene (appliance), bilo da su u pitanju uređaji za opštu namenu kao što su kućni ADSL i WiFi ruteri, pametni televizori, bilo kao specifični kontroleri za kompleksne sisteme, od medija centara u automobilima sve do upravljanja ICBM projektilima! Linux je veoma popularan kao operativni sistem za mobilne telefone – Android platforma, bazirana na Linuxu trenutno drži više od polovine tržišta pametnih telefona, a svi novi operativni sistemi prikazani na MWC konferenciji 2013. godine su bazirani na Linuxu!

Kao server, Linux je upotrebljiv za gotovo sve servise. Može služiti kao mail server, web server, ftp server, dns server, file server, print server, ldap server i mnogo toga drugo. Linux se može koristiti za obrazovanje klastera tipa high availability (HA) i high performance computing (HPC) – preko 90% najbržih računara sa Top500 liste, uključujući i 9 od 10 najbržih računara na svetu radi na Linuxu.

Kao radna stanica Linux se koristi kao tipična desktop inženjerska radna stanica, diskless radna stanica ili kao klijent za terminal servere.

Pitanja i zadaci

1. Po kome je Linux dobio ime?
2. Da li open-source licenca zahteva da softver bude besplatan?
3. Navedite bar po dve Linux distribucije koje pripadaju sledećim kategorijama: serverske, desktop, za mobilne platforme, za razne uređaje (appliance). Koristite Internet ako niste sigurni koja distribucija pripada kojoj kategoriji.
4. Pogledajte sajt <http://top500.org/>. Na kojem mestu se nalazi prvi superračunar sa liste koji ne koristi Linux?

Upoznavanje sa osnovama na kojima je zasnovan Linux

Da bi razumeli Linux i shvatili kako on radi, moramo se prvo upoznati sa osnovama na kojima je ovaj operativni sistem zasnovan, a to su:

- fajlovi
- procesi
- softverski alati

Fajlovi

Linux sledi filozofiju UNIX-a i tretira većinu stvari kao fajlove, ne zadržavajući se na entitetima koje sadrže podatke, ono što uobičajeno nazivamo fajlovima. U Linuxu su i razni elementi hardvera, metodi komunikacije među programima prikazani fajlovima. To nam daje mogućnost da fajlove podelimo na različite tipove:

- **regularni fajlovi** – ovo su fajlovi u uobičajenom smislu reči, kao i kod Windows-a;
- **direktorijumi** – direktorijumi su takođe jedna vrsta fajlova, koji sadrže neke metapodatke o ostalim fajlovima (njihova imena i lokacije strukture koja opisuje fajlove – i-nodove);
- **simbolički linkovi** – soft linkovi su pointeri na druge fajlove, slično kao *.lnk fajlovi kod Windows-a.

Pored ovih tipova fajlova, sa kojima se najčešće susrećemo, na Linuxu postoje i druge vrste fajlova koje omogućavaju pristup nekim elementima hardvera, kao i za komunikaciju među programima:

- **blok specijalni fajlovi** – ovi fajlovi predstavljaju hardverske uređaje kojima je blok najmanja količina podataka koji se mogu pročitati i upisati;
- **karakter specijalni fajlovi** – ovi fajlovi predstavljaju hardverske uređaje kojima je jedan znak najmanja količina podataka koji se mogu pročitati i upisati;
- **imenovani soketi** – ovaj tip fajlova predstavlja sokete kao vrstu međuprocenke komunikacije.

Sami fajlovi, bez obzira na tip, opisani su posebnim strukturama koje sadrže različite metapodatke koji definišu sam fajl, kao što su:

- tip fajla
- prava pristupa nad fajlom
- vlasništvo nad fajlom
- vremena kreiranja, poslednjeg pristupa i poslednje izmene sadržaja fajla
- broj referenci na fajlu (biće objašnjeno u sledećem poglavlju)
- ostali podaci koji zavise od tipa fajla (npr. kod regularnih fajlova i direktorijuma, uređaj na kojem se podaci nalaze i lokacija podataka unutar uređaja; kod specijalnih fajlova, indeks koji određuje kojim drajverom se pristupa uređaju i dodatni podaci specifični za dati drajver).

Imena fajlova

Ukoliko ste pažljivo pročitali prethodni pasus, verovatno ste primetili da ime fajla nije deo metapodataka koji definišu fajl. To je zato što jedan fajl na Linuxu može da ima više ravnopravnih imena – broj različitih imena se čuva u broju referenci, dok se sama imena nalaze u sadržaju direktorijuma. Ova imena se nazivaju hard linkovi i međusobno su ravnopravna. Sve dok postoji i jedno ime koje ukazuje na fajl, taj fajl neće biti obrisan sa diska.

Imena fajlova ne određuju sadržaj fajla, tj. ime fajla može biti proizvoljno i nezavisno od toga šta taj fajl sadrži.

Ime fajla može biti dugačko do 255 znakova. Jedini znaci koji ne mogu biti upotrebljeni u imenu su znak '/' i nul bajt (znak čiji je kod 0). Svi ostali znaci su dozvoljeni, uključujući i neprintabilne znakove.

Ekstenzije kao posebni delovi imena nisu definisani u Linuxu i, ako postoje, koriste se samo radi konvencije.

Posebne karakteristike imaju fajlovi čije ime počinje sa znakom '.' - ovo su takozvani 'skriveni' fajlovi, koji se ne vide kada se posmatra uobičajeni ispis sadržaja direktorijuma, bilo u komandnoj liniji bilo u grafičkom fajl menadžeru. Naravno, ovi fajlovi nemaju nikakvo drugo posebno značenje i mogu se videti kada se uključe opcije koje prikazuju i skrivene fajlove. Namena 'skrivenih' fajlova je samo da ne opterećuju ispis listinga direktorijuma.

Puna imena fajlova

Puno ime fajla (engl. *pathname*) se sastoji od putanje od korena fajl sistema do direktorijuma i samog imena fajla, razdvojenih znakom '/'. Putanja do direktorijuma može biti:

- **apsolutna putanja** – putanja koja počinje od korenog (root) direktorijuma, pri čemu puno ime fajla počinje znakom '/'
- **relativna putanja** – putanja relativna u odnosu na direktorijum u kojem se korisnik trenutno nalazi. Ova putanja nikad ne počinje znakom '/'. Ukoliko se sam fajl nalazi u direktorijumu u kojem se i korisnik trenutno nalazi (tekući direktorijum), onda je samo ime fajla istovremeno i njegovo puno ime sa relativnom putanjom.

Vlasništvo nad fajlovima

Korisnik može pristupiti nekom fajlu ukoliko ima odgovarajuća prava pristupa. Ova prava se definišu nad tri entiteta: vlasnikom fajla, grupom asociranom fajlu i svim ostalim korisnicima (koji nisu vlasnik fajla i nisu članovi grupe asocirane fajlu). Jedini izuzetak od ovog pravila je supervizorski korisnik, koji ima username 'root' (ili korisnik koji privremeno ima prava korisnika 'root').

Vlasništvo nad fajlom može da promeni samo 'root' korisnik. Običan, neprivilegovani korisnik, može samo da izmeni grupu koja je asocirana fajlu, i to sa nekom drugom grupom kojoj taj korisnik pripada.

Osnovna prava pristupa nad fajlovima

Standardna prava pristupa nad fajlovima su:

- pravo izmene (write – w)
- pravo pristupa sadržaju (read – r)
- pravo izvršavanja fajla (execute – x) – ovo je jedini način na koji se označavaju izvršni programi

Ova standardna prava su grupisana u tri grupe:

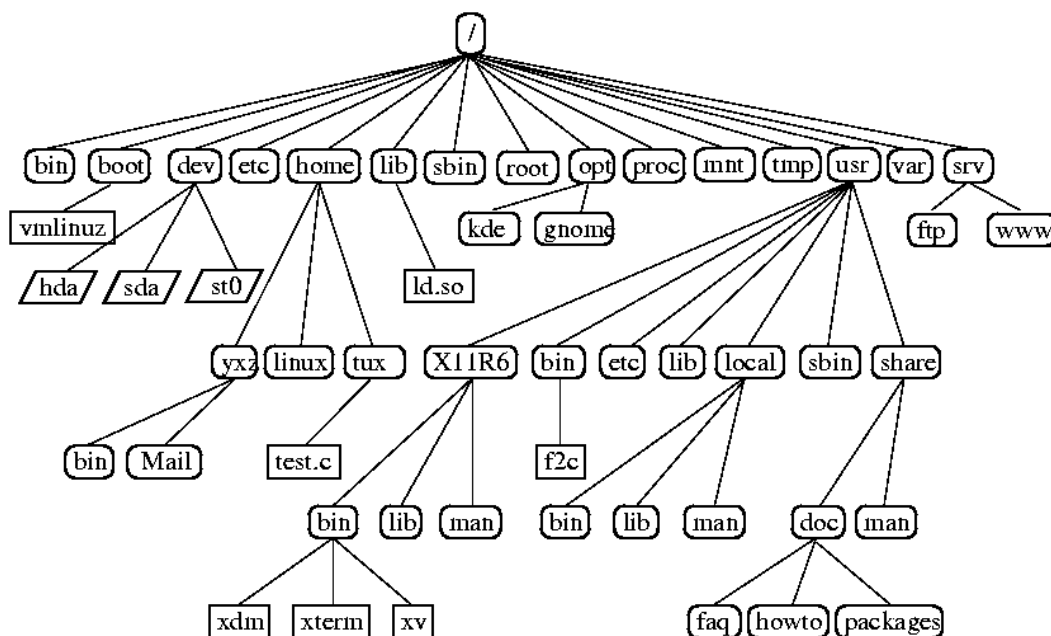
- prava za vlasnika fajla
- prava za grupu kojoj pripada fajl
- prava za sve ostale korisnike

Direktorijumi imaju definisana ista prava kao i ostali tipovi fajlova, ali kod direktorijuma ta prava imaju drugačije značenje:

- write pravo označava da se može menjati sadržaj direktorijuma (dodavati i brisati fajlovi, menjati imena fajlovima)
- read pravo označava da se sadržaj direktorijuma može izlistati – ovo pravo ne znači da se nekom fajlu može i pristupiti
- execute pravo označava da korisnik može pristupiti fajlovima.

Hijerarhijska struktura stabla direktorijuma

Linuxovo stablo direktorijuma je hijerarhijski organizovano i nije dozvoljeno da postoje petlje unutar stabla. Bez obzira na kojim uređajima se fizički nalaze i prostiru fajlovi, oni na jednom sistemu uvek sačinjavaju jednu hijerarhiju – Linux (kao ni ostali UNIX-i) ne poznaje pojam 'diska' niti 'slova koje označava disk'. U ovom slučaju, pojam 'disk' označava disk partitiju na kojoj se nalaze fajlovi ili deljeni mrežni disk.



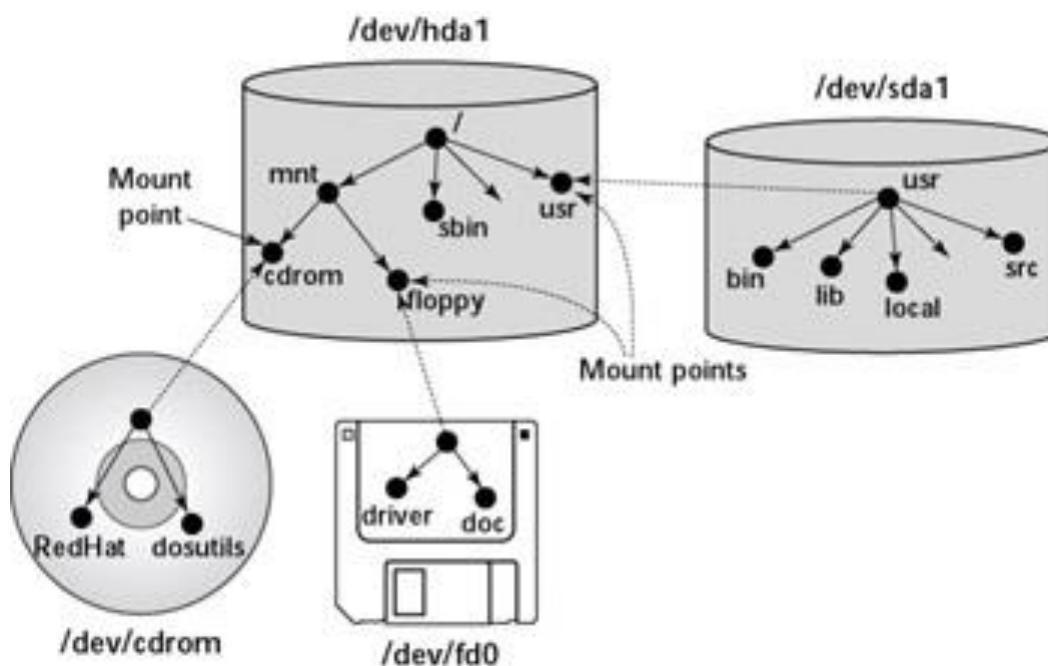
Slika 1 - Hijerarhijska struktura fajl sistema na Linuxu

Posebni fajlovi . i ..

U svakom direktorijumu se nalaze i dva posebna imena: '.' i '..'. Ime '.' je drugo ime za tekući direktorijum, dok je '..' ime za nadređeni direktorijum (izuzev u slučaju '/' direktorijuma koji nema svoj nadređeni direktorijum, pa imena '.' i '..' označavaju tekući direktorijum). Ova dva imena su značajna jer omogućavaju kretanje uz i niz stablo direktorijuma.

Tačke kačenja (mount points)

Tačka kačenja je direktorijum od kojeg se nadovezuje stablo direktorijuma koje se fizički nalazi na jednom uređaju (disk partcija, mrežni disk).



Slika 2 - tačke kačenja pojedinih uređaja

Procesi

Program je izvršni objekat, smešten u fajlu. Da bi se program mogao izvršiti, korisnik koji pokreće program mora imati pravo izvršavanja na fajlu koji sadrži program.

Proces je instanca programa u izvršavanju. Jedan program u jednom trenutku može izvršavati više korisnika (ili jedan korisnik može pokrenuti više instanci istog programa) – iako je u pitanju isti program, svaka instanca koja se izvršava je zaseban proces.

Svaki proces je definisan brojem, koji se naziva **PID** (*Process ID*). Svaki proces pokreće neki korisnik, tako da proces nosi informaciju koji korisnik ga je pokrenuo. Takođe, proces od korisnika nasleđuje i članstvo u grupama koje definiše da li proces ima pravo da pristupi nekom fajlu i na koji način.

Jedan proces nastaje tako što ga startuje drugi proces, tako da svaki proces ima svog 'roditelja', kojeg pamti prema njegovom PID-u, koji se onda naziva **PPID** (*Parent Process ID*). Osnovni proces koji se startuje kada se sistem startuje se naziva 'init' proces i ima PID 1. Ovaj proces je, dakle, 'praroditelj' svim ostalim procesima na sistemu.

Dodatne informacije koje proces čuva su:

- tekući direktorijum
- bitska maska koja određuje koja prava pristupa će imati fajlovi koje proces kreira
- UID vlasnika (realni i efektivni)
- GID-ovi (grupe čiji je vlasnik član) – realni i efektivni
- Okruženje (promenljive okruženja)

Svaki proces, kada se okonča njegovo izvršavanje, vraća statusnu informaciju u obliku izlaznog koda. Po konvenciji, izlazni kod 0 označava da je proces uspešno okončao svoj posao. Izlazni kod različit od 0 označava da proces nije uspeo u svom poslu – specifične vrednosti zavise od programa do programa.

Softverski alati

Linux je baziran na UNIX-u, a osnovna filozofija UNIX-a je da sistem bude sastavljen iz velikog seta programa od kojih svaki radi tačno jednu stvar. Kombinovanjem ovih programa/alata može se obaviti prilično kompleksna obrada podataka.

Da bi ovo moglo da funkcioniše, UNIX sistem je pojednostavljen jer se sastoji iz samo dve grupe objekata:

- fajlova
- procesa

Obrada podataka može da se obavi zato što je precizno definisana i pojednostavljena interakcija:

- procesa sa fajlovima
- procesa sa procesima

Pitanja i zadaci

1. Koja su tri osnovna elementa na kojima je baziran Linux?
2. Nabrojte tipove fajlova koje Linux prepoznaje.
3. Šta je i-nod i šta sadrži?
4. Kako se razlikuje pravo ime fajla od drugog hard linka?
5. Ako nekome saopštite relativnu putanju do nekog fajla, da li je to dovoljan podatak da se taj fajl može locirati?
6. Koja su osnovna prava pristupa?
7. Šta označava x-pravo na direktorijumu?
8. Koja je slovna oznaka glavnog diska na Linuxu?
9. Na šta pokazuje fajl .. u korenom direktorijumu (/) ?

10. Koliko roditelja ima jedan proces?
11. Linux prati filozofiju UNIX-a 'no news is good news'. Kako onda znamo da li se komada izvršila ispravno?
12. Kako fajlovi komuniciraju međusobno?

Pristup Linux-u, terminali i SSH

Linux je multikorisnički operativni sistem. To znači da u jednom trenutku više korisnika istovremeno može da radi na sistemu. Takođe, jedan korisnik može istovremeno više puta biti ulogovan i pokretati različite programe.

Korisnici moraju na neki način pristupiti serveru da bi mogli da se uloguju i počnu da rade. Oni to mogu obaviti direktno na konzoli servera ili preko mreže.

Konzolni pristup

Pod serverskom konzolom se podrazumeva pristup preko tastature i monitora koji su direktno nakačeni na sam server (na virtuelnim mašinama, ovaj pristup se emulira preko konzolnog pristupa u okviru aplikacije za upravljanje hipervizorom).

Ukoliko instalirate punu instalaciju Linux-a (ili odaberete instalaciju za radnu stanicu a ne server) ogromna verovatnoća je da ćete moći da se ulogujete direktno na grafičko okruženje. Međutim, za razliku od Windows-a, Linux serveru je vrlo retko potrebno da ima instalirano grafičko okruženje – u višegodišnjem iskustvu autora ove skripte, koji za to vreme instalirali, konfigurisali i upravljali sa više stotina Linux servera različitih distribucija, grafičko okruženje su na serverima instalirali svega par puta i to zato što je aplikacija koja je trebala da se izvršava na tom serveru to eksplicitno zahtevala! Iz tog razloga ATC Linux kursevi se ne bave grafičkim okruženjem na Linuxu i posvećeni su isključivo radu na tekstualnoj konzoli. Uostalom, u grafičkom okruženju je dovoljno pokrenuti terminalsku aplikaciju (ili više njih paralelno).

No, Linux na konzoli omogućava, pored grafičkog okruženja i direktan terminalski pristup, i to na više terminala paralelno. Prilikom startovanja se obično pokreće nekoliko terminala paralelno sa grafičkim okruženjem. Do ovih terminala iz grafičkog okruženja se dolazi pritiskom na kombinaciju tastera <CTRL><ALT>F1 – F6. Dakle, imate na raspolaganju 6 konzolnih terminala. Kada ste prebačeni na konzolni terminal, ostali terminali su dostupni preko kombinacije tastera <ALT>F1-F6, dok je grafička konzola dostupna sa <ALT>F7 (vodite računa da je na nekim distribucijama ovo promenjeno pa je grafičko okruženje na poziciji 1 ili 2, a tekstualne konzole na ostalim pozicijama).

Mrežni pristup

Daleko češće ćete svojim serverima pristupati preko mreže. Postoji nekoliko protokola koji vam omogućavaju rad sa tekstualnim terminalima preko mreže ali ubedljivo najviše korišćen je SSH (eng. Secure SHell, bezbedno komandno okruženje). SSH je popularan iz nekoliko razloga:

- komunikacija preko mreže je kriptovana i time bezbedna
- postoji više različitih načina autentifikacije na server, pored standardne kombinacije username/password
- SSH klijentske aplikacije su dostupne na svim operativnim sistemima, uključujući i pametne telefone.

- SSH dolazi sa SCP (engl. Secure CoPy, bezbedno kopiranje) komandom, kojom možete kopirati fajlove sa i na udaljenu mašinu.

Autentifikacija

Bez obzira kako pristupate serveru, on vas mora autentifikovati kako bi utvrdio koji ste vi korisnik i koja su vam prava na sistemu.

Svaki korisnik dobija svoj username, koji mora biti jedinstven na jednom serveru (Linux vas inače ne pamti po username-u, već po UID-u, što je jedinstven celi pozitivni broj asociran vašem nalogu). Username je javna informacija i ostali korisnici vide vaš username kao i vi njihov.

Pored username-a, sistem zahteva minimalno još jedan parametar kojim ćete potvrditi da ste upravo vi taj korisnik sa zadatim username-om. Taj podatak mora biti tajan i samo ga vi trebate znati. Najčešće je taj podatak vaša lozinka, ali Linux je dovoljno fleksibilan da se mogu podesiti različiti načini autentifikacije (SSH, recimo, forsira autentifikaciju sa javnim i tajnim ključem, koja je bezbednija od prebacivanja lozinke preko mreže, makar i po enkriptovanom kanalu). Drugi oblici autentifikacije mogu biti primenjeni (OTP, biometrijska autentifikacija), od kojih neki mogu zahtevati više od jednog tajnog podatka.

No, kada se uspešno autentifikujete na sistem i sistem vas prepozna kao validnog korisnika, naći ćete se pred promptom u komandnom interpreteru. O njemu i radu u komandnoj liniji posvećen je ostatak ovog kursa.

Pitanja i zadaci

1. Da li SSH omogućava autentifikaciju preko lozinke?
2. Koja od ovih informacija treba da bude tajna: username, lozinka?

Rad u komandnoj liniji

Komandni interpreter (*shell*, ljuska) je program preko kojeg korisnik interaguje sa sistemom. Može biti bilo kakav program, ali je uobičajeno da to bude neki od standardnih komandnih interpretera. Komandni interpreter se obično startuje kada se ulogujete, ili, u grafičkom okruženju, kada pokrenete terminal. Nezavistan komandni interpreter se obično startuje i kada pokrenete neku od komandnih skripti.

Standardni UNIX komandni interpreteri su **'sh'** (*Bourne shell*) i **'csh'** (*C shell*). Međutim, oni nemaju neke mogućnosti koje su korisnicima često potrebne, kao što je kompleksno nadopunjavanje linije koje skraćuje kucanje i sl. Ove 'novine' su se pojavile sa kasnijim komandnim interpreterima, koji su, zarad kompatibilnosti, zadržali sve prethodne osobine 'sh', odnosno 'csh' interpretera. U standardnim Linux distribucijama, podrazumevani (default) komandni interpreter je **'bash'** (*Bourne Again shell*), koji je nadgradnja osnovnog, 'sh' komandnog interpretera. Analog bash-a, ali sa 'csh' sintaksom je **'tcsh'** (*TC shell*), namenjen onima koji preferiraju sintaksu komandnog interpretera sličnu programskom jeziku C.

No, bilo da radite sa jednim ili drugim komandnim interpreterom, uvidećete da ogromna većina komandi ima istu sintaksu – razlog tome je što to nisu 'ugrađene', tj. interne komande, već eksterni programi! Naime, osnovna filozofija UNIX-a je da ima puno malih programa „koji rade jednu stvar, ali je rade dobro“, tako da, za razliku od npr. Microsoft-ovog DOS-a, u komandni interpreter nije potrebno ugrađivati komande, jer su one implementirane kao zasebni programi (ili komandne skripte). U stvari, jedine komande koje su implementirane u standardnim interpreterima su one koje utiču na ponašanje tekućeg procesa (kao i mali broj često korišćenih komandi kao što je 'echo' koji služi za ispis teksta na ekranu, implementirana u interpreteru radi brzine).

Kada je komandni interpreter spreman da primi komandu od korisnika on ispisuje prompt. Standardni prompt je obično znak '\$' (za root korisnika '#'), ali kod novijih komandnih interpretera prompt je konfigurabilan i obično sadrži neke korisne informacije kao što su korisničko ime, ime mašine i deo ili puna putanja tekućeg direktorijuma. Primer standardnog prompta na RedHat/CentOS sistemima je:

```
[it4biz@vps30305 ~]$
```

gde je:

- it4biz – ime korisnika na sistemu
- vps30305 – ime mašine
- ~ - oznaka dela putanje do tekućeg direktorijuma (znak ~ označava početni direktorijum korisnika).

Mi ćemo u nastavku ovog teksta koristiti samo znakove '\$' za prompt neprivilegovanog korisnika i '#' za prompt administratora/root korisnika.

Standardni oblik komande u Linux-u je:

```
$ komanda -o --duga-opcija=arg argument1 argument2 ...
```

gde je:

- komanda – ime ili ime sa putanjom komande koju želimo izvršiti
- – opcija komande u tzv. kratkom obliku, može imati i svoj argument koji treba navesti odmah iza parametra (kratke opcije se navode sa jednim znakom '-')
- duga-opcija – takođe opcija komande, ali u tzv. dugom obliku, navodi se sa dva znaka '-'. Mnoge opcije imaju svoje kratke, jednoslovne i svoje duge nazive i nije bitno koji oblik primenjujete
- arg – argument opcije; kod dugih opcija se navodi iza opcije, uz znak '='; u slučaju kratke opcije, navodi se iza opcije, razdvojen blanko znakom ili neposredno iza same opcije, bez blanko znaka (zavisno od komande).
- argument1 argument2 ... - argumenti same komande; zavisno od komande može ih biti nula, jedan ili više.

Napomena

Kod većine komandi redosled opcija nije bitan. Takođe, kod većine komandi nije obavezno da opcije komande budu navedene ispred argumenata komande, već argumenti komande i opcije mogu biti izmešani po redosledu (argumenti opcija moraju biti navedeni iza odgovarajuće opcije).

Najčešći argumenti komande su nazivi fajlova, direktorijuma i sl. U tim slučajevima moguće je koristiti tzv. **fajl globing** kako bi se na sažet način navelo više fajlova. Fajl globing koristi tzv. *wildcard* znakove:

- * - označava bilo koji fajl koji ne počinje znakom '.', tj. nije skriveni fajl
- ? - označava tačno jedan znak
- [a-z01.3-] – označava **tačno jedan znak** koji može biti bilo koji znak koji je naveden unutar uglastih zagrada, odnosno znak koji je u ASCII poretku unutar opsega navedenih znakova (opsezi se navode za početnim znakom i krajnjim znakom u nizu, spojenih znakom '-').
- [^a-z] – označava **tačno jedan znak** koji može biti bilo koji znak koji **nije naveden** unutar uglastih zagrada, odnosno znak koji je u ASCII poretku unutar opsega navedenih znakova (opsezi se navode za početnim znakom i krajnjim znakom u nizu, spojenih znakom '-').
- {ime1,ime2,ime3} – na mesto ovog niza može doći bilo koji niz znakova navedenih unutar vitičastih zagrada, razdvojenih zarezom (između niza znakova i znaka ',' ne sme postojati blanko znak.

Osnovne razlike u odnosu na Windows su, pored toga da postoji veći broj wildcard znakova su te da se tzv. ekspanovanje fajl globinga izvršava pre nego što se sama komanda izvrši, tako da komanda ne dobija argumente kao fajl globing znakove, tj. sa wildcard znakovima, već dobija kompletan skup fajlova koji odgovara fajl globing izrazu. Ekspanovanje fajl globing izraza izvršava komandni interpreter (kod Windows-a je to sama komanda) tako da autor komande ne mora da vodi računa o ekspanovanju fajl globinga.

Komandni interpreter

Standardni interpreter komandne linije na Linuxu je bash (/bin/bash ili /bin/sh).

Poseđuje većinu osobina drugih komandnih interpretera a ima i neke još naprednije osobine:

- transparentnu redirekciju
- pajpove
- procese u pozadini
- suspenziju izvršavanja procesa, njihovo nastavljnje i terminaciju
- kompletiranje imena fajlova
- komandnu istoriju

Pomoć u komandnoj liniji

Većina početnika kad se nađe prvi put licem u lice sa komandnim interpreterom, najčešće poseže za nekim oblikom pomoći koji će ih uputiti šta dalje raditi, posebno ako pogreše u sintaksi ili zadavanju argumenata komande. Jedna od prvih komandi koju otkućaju je 'help' i tu ih čeka prvo iznenađenje, koje, priznajemo, može biti pomalo zastrašujuće i odbijajuće.

Naime, komanda help se odnosi isključivo na komandni interpreter i daje vam informacije samo o internim komandama. A većina komandi na Linux-u su eksterne komande, tj. zasebni programi koje korisnik pokreće. Kako dobiti pomoć za te komande?

Prvo, većina komandi ima neko kratko uputstvo koje se dobije kada se komanda pozove sa pogrešnim parametrima, a koje se može dobiti i zadavajući odgovarajuću opciju. Na žalost početnika, to uputstvo se uglavnom odnosi na format same komande i, što je još gore, opcija kojom se to uputstvo poziva se razlikuje od komande do komande. Najčešće je to

```
$ command -h
```

ili

```
$command -help
```

kao na primer:


```

$ mkdir -h
mkdir: invalid option -- 'h'
Try `mkdir --help' for more information.
$ mkdir -help
Usage: mkdir [OPTION]... DIRECTORY...
Create the DIRECTORY(ies), if they do not already exist.

Mandatory arguments to long options are mandatory for short
options too.
  -m, --mode=MODE    set file mode (as in chmod), not a=rwx -
umask
  -p, --parents       no error if existing, make parent
directories as needed
  -v, --verbose       print a message for each created directory
  -Z, --context=CTX   set the SELinux security context of each
created
                        directory to CTX
      --help          display this help and exit
      --version       output version information and exit

Report mkdir bugs to bug-coreutils@gnu.org
GNU coreutils home page: http://www.gnu.org/software/coreutils/
General help using GNU software: http://www.gnu.org/gethelp/
For complete documentation, run: info coreutils 'mkdir
invocation'
$

```

No, ako se želi neko opširnije uputstvo, a 'help' ne radi posao, šta treba uraditi? Na Linux-u, kao i na ostalim Unix operativnim sistemima, uputstvo za neku komandu se dobija sa:

```
$ man [sekcija] termin
```

man je komanda koja prikazuje stranu referentnog uputstva (engl. *manual*, odatle ime komandi). Ovo uputstvo je tako napravljeno da može odmah biti štampano i obično se sastoji iz više standardnih odeljaka (NAME, SYNOPSIS, DESCRIPTION, OPTIONS, RETURN VALUE, EXAMPLES, AUTHOR, SEE ALSO, COPYRIGHT i sl.).

Parametar *termin* označava ono o čemu tražimo stranu uputstva, a to je obično komanda, sistemski poziv ili neka biblioteka funkcija (programersko uputstvo je takođe u sklopu man strana). Takođe, neki koncepti i konfiguracioni fajlovi imaju svoje zasebne man strane. Pošto je u tom slučaju moguće da jedan isti termin označava više različitih stvari, man strane su podeljene u sekcije. Na taj način možemo dobiti odgovarajuću stranu, ako znamo kojoj grupi željeni termin pripada. Na primer, termin 'printf' označava i komandu za formatirani ispis, ali i standardnu biblioteku funkciju koja obavlja isti posao u programskom jeziku C. Iz tog razloga, ako hoćete da dobijete uputstvo za C funkciju, obično 'man printf' neće pomoći jer će man prikazati uputstvo za komandu 'printf' jer je ona u sekciji 1, a C funkcija u sekciji 3.

Standardna podela man sekcija je:

- 1 – korisničke komande

- 2 – sistemski pozivi (programerska uputstva)
- 3 – funkcije C biblioteke (programerska uputstva)
- 4 – Uređaji i specijalni fajlovi
- 5 – Formati fajlova i standardi
- 6 – Igre i sl.
- 7 – Razne stvari
- 8 – Alati za sistemsku andinistraciju i dimoni

O svakoj sekciji možete da dobijete uputstvo ako otkucate:

```
$ man broj_sekcije intro
```

Na kraju, uputstvo za sam man program možete dobiti sa:

```
$ man man
```

Kretanje po komandnoj liniji

Možete koristiti kursorske tastere za kretanje po komandnoj liniji i editovanje tekuće linije. Man strana za bash daje detalje o svim sekvencama.

Bash pamti ranije unete komande (u 'komandnoj istoriji'). Stare komande su dostupne na različite načine:

- ponovite prethodnu komandu ukucavanjem '!!'
- izvršite n-tu prethodnu komandu ukucavajući '!-n'
- da bi videli listu komandi u komandnoj istoriji, liniju po liniju, koristite kurzorske tastere za gore i dole
- pregledajte listu komandi u komandnoj istoriji u bilo kojem trenutku komandom

```
$ history
```

- ukucavajući

```
$ !string
```

ponavljate poslednju komandu koja je počinjala sa *string*

- pritiskom na CTRL-R, pa unoseći neki tekst vraćate se unazad po komandnoj istoriji na komandu koja počinje unetim tekstom. Prelazak na komandu je interaktivan i dešava se posle svakog unetog znaka.

Pitanja i zadaci

1. Šta je fajl globing?
2. Koji wildcard znak označava tačno jedan znak?
3. Kako biste uz pomoć wildcard znakova naveli sve fajlove i direktorijume unutar tekućeg direktorijuma, uključujući i skrivene fajlove, ali bez fajlova . i .. ?
4. Kako dobijamo pomoć za interne komande interpretera?
5. Šta se nalazi u sekciji 5 referentnog uputstva?
6. Kako možemo da vidmo koje prethodne komande smo zadali?

Osnovne komande za rad sa fajlovima

ls – listanje sadržaja direktorijuma

Prikaz sadržaja direktorijuma se dobija komandom `ls`. Najjednostavniji oblik ove komande je:

```
$ cd /
$ ls
backups  cgroup  home    lost+found  opt  sbin    sys  var
bin      dev     lib     media      proc selinux tmp
boot     etc     lib64   mnt        root  srv     usr
```

Bez ikakvih argumenata, komanda `ls` prikazuje sadržaj tekućeg direktorijuma. Ako se komanda pozove sa argumentima koji su fajlovi i/ili direktorijumi, `ls` prikazuje informacije o tim fajlovima, odnosno sadržaje tih direktorijuma

Ova komanda ima gomilu opcija pa pogledajte njenu man stranu da biste ih upoznali. Ovde ćemo navesti samo neke:

| Opcija | Značenje |
|-----------|---|
| -a | Prikaz 'skrivenih' fajlova i direktorijuma (čija imena počinju sa '.') |
| -l | Detaljniji prikaz informacija o fajlovima |
| -1 | Ispis jednog fajla u jednoj koloni |
| -d | Ukoliko je argument komande 'ls' direktorijum, sa ovom opcijom će biti prikazane informacije o tom direktorijumu umesto njegovog sadržaja |
| -F | Označava direktorijume, izvršne fajlove i simboličke linkove dodajući na kraj imena znake /, * i @, respektivno. |
| -R | Rekurzivno izlistava sadržaj poddirektorijuma |
| -t | Sortra ispis po vremenu. |
| -r | Reverzno sortiranje. |

Primeri ovih opcija:

```

$ cd /
$ ls
backups  cgroup  home    lost+found  opt  sbin    sys  var
bin      dev      lib      media      proc  selinux  tmp
boot     etc      lib64    mnt         root  srv      usr
$ ls -a
.          backups  dev      lib64        opt  selinux  usr
..         bin      etc      lost+found   proc  srv      var
.autofsck  boot     home     media        root  sys
.auorelabel cgroup   lib      mnt          sbin  tmp
$ ls -al
total 110
dr-xr-xr-x. 24 root root 4096 Aug  1 17:22 .
dr-xr-xr-x. 24 root root 4096 Aug  1 17:22 ..
-rw-r--r--  1 root root    0 Aug  1 17:22 .autofsck
-rw-r--r--  1 root root    0 Feb  7  2013 .autorelabel
drwxr-xr-x  2 root root 4096 Mar 13  2013 backups
dr-xr-xr-x.  2 root root 4096 Sep 30 12:03 bin
dr-xr-xr-x.  5 root root 1024 Feb  7  2013 boot
drwxr-xr-x.  2 root root 4096 Jun 22  2012 cgroup
drwxr-xr-x 17 root root 3720 Aug  1 17:22 dev
drwxr-xr-x. 72 root root 4096 Sep 30 14:42 etc
drwxr-xr-x.  4 root root 4096 Mar  6  2013 home
dr-xr-xr-x.  8 root root 4096 Feb  6  2013 lib
dr-xr-xr-x.  9 root root 12288 Mar 13  2013 lib64
drwx-----  2 root root 16384 Feb  6  2013 lost+found
drwxr-xr-x.  2 root root 4096 Sep 23  2011 media
drwxr-xr-x.  2 root root 4096 Sep 23  2011 mnt
drwxr-xr-x 12 root root 4096 Mar 19  2013 opt
dr-xr-xr-x 177 root root    0 Aug  1 17:22 proc
dr-xr-x---.  5 root root 4096 Sep 30 13:41 root
dr-xr-xr-x.  2 root root 12288 Mar 13  2013 sbin
drwxr-xr-x.  2 root root 4096 Feb  6  2013 selinux
drwxr-xr-x.  2 root root 4096 Sep 23  2011 srv
drwxr-xr-x 13 root root    0 Aug  1 17:22 sys
drwxrwxrwt.  6 root root 4096 Sep 30 16:06 tmp
drwxr-xr-x. 13 root root 4096 Feb  6  2013 usr
lrwxrwxrwx  1 root root    8 Mar 19  2013 var -> /opt/var
$ ls -l /home
total 8
drwx----- 2 strahinja strahinja 4096 Mar  6  2013 strahinja
drwx----- 2 strahinja mail      4096 Feb  7  2013 vmail
$ ls -ld /home
drwxr-xr-x. 4 root root 4096 Mar  6  2013 /home

```

Informacije koje prikazuje `ls -l` su sledeće:

- tip fajla i prava pristupa
- broj referenci (hard linkova, različitih imena fajla)
- vlasnik fajla
- grupa koja je vlasnik fajla

- zauzeće prostora u bajtovima
- Mesec (troslovna skraćenica) kada je fajl kreiran
- Dan u mesecu kada je fajl kreiran
- Vreme kreiranja fajla ili godina ako je fajl stariji od 3 meseca
- ime fajla uz eventualne pokazatelje za simboličke linkove

Tip fajla i prava pristupa su prikazana na sledeći način:

tuuugggooo.

gde je:

- t – tip fajla:
 - t – za regularne fajlove
 - d za direktorijume
 - l za simboličke linkove
 - b za blok specijalne fajlove
 - c za karakter specijalne fajlove
 - p za imenovane pajpove
 - s za sokete
- uuu, ggg i ooo su prava pristupa za vlasnika, grupu i ostale, respektivno. Svi imaju isti oblik i to:
 - prvi znak je r ako postoji pravo čitanja fajla / listanja sadržaja direktorijuma ili – ako ovo pravo ne postoji
 - drugi znak je w ako postoji pravo izmene sadržaja fajla / dodavanja, brisanja i preimenovanja fajlova u direktorijumu ili – ako ovo pravo ne postoji
 - treći znak je x ako je fajl proglašen izvršnim / korisnik može proći kroz direktorijum ili – ako ovo pravo ne postoji.

Dodatno, ako u delu prava pristupa za vlasnika umesto x stoji S, fajl je izvršan i ima postavljen setUID bit. Ako umesto x stoji S, onda fajl ima postavljen setUID bit ali nema pravo izvršavanja

(što nije smisljena kombinacija pošto setUID bit deluje samo na izvršne fajlove i direktorijume u kojim korisnik ima mogućnost pristupa/prolaska).

Ako u delu prava pristupa za grupu umesto x stoji S, fajl je izvršan i ima postavljen setGID bit. Ako umesto x stoji S, fajl nije izvršan i ima setGID postavljen (što nije smisljena kombinacija).

Ako u delu prava pristupa za ostale umesto x stoji t, direktorijum ima postavljen sticky-bit.

Napomena

SetUID, setGID i sticky-bit su specijalna prava pristupa koja se obrađuju u poglavlju Prava pristupa – dodatno

cp – kopiranje fajlova i direktorijuma

Kopiranje fajlova se obavlja komandom cp.

```
$ cp [opcije] source... dest
```

Ova komanda mora imati najmanje dva argumenta. Ako je komanda pozvana sa tačno dva argumenta, onda je prvi argument fajl ili direktorijum koji se kopira, a drugi argument je destinacija i može biti:

- fajl – ako je prvi argument fajl onda destinacija može biti fajl. Ako destinacioni fajl postoji biće prebrisan
- direktorijum – označava destinacioni direktorijum i mora postojati.

Ukoliko komanda cp ima više argumenata, onda poslednji argument mora biti neki postojeći direktorijum.

Najčešće korišćene opcije komande 'cp' su:

| Opcija | Značenje |
|--------|---|
| -a | Destinacioni fajlovi dobijaju isto vlasništvo kao i izvorni fajlovi (važi samo za root korisnika) |
| -r | Rekurzivno kopiranje |
| -v | Ispisivanje naziva izvorišnog i destinacionog fajla – objašnjava šta se gde kopira |
| -p | Destinacioni fajl dobija iste parametre kao i izvorišni fajl (važi samo za root korisnika) |

mv – prebacivanje i preimenovanje fajlova i direktorijuma

Komanda 'mv' se koristi za preimenovanje i pomeranje fajlova iz jednog direktorijuma u drugi. Ova komanda je veoma brza jer ne kopira sadržaj fajlova već samo menja zapise u direktorijumima.

```
$ mv fajl1 fajl2
```

Komanda 'mv' mora imati najmanje dva argumenta, od kojih je prvi izvorišni fajl a drugi destinacioni fajl (novo ime) ili direktorijum (nova lokacija). Ako komanda ima više od dva argumenta, poslednji argument mora biti direktorijum koji već postoji, pa se tu radi isključivo o pomeranju fajlova.

rm – brisanje fajlova i direktorijuma

Fajlovi se brišu komandom 'rm'. Ova komanda je potencijalno opasna jer ne zahteva da korisnik potvrdi brisanje fajla:

```
$ rm fajl1 fajl2 ...
```

Ovom komandom se mogu rekurzivno brisati direktorijumi koji ne moraju biti prazni:

```
$ rm -r dir1 dir2 ...
```

Ako želimo da komanda zahteva potvrdu pre brisanja nekog fajla, zadaćemo '-i' opciju.

ln – kreiranje linkova

Komandom ln možemo napraviti hard i soft linkove. Hard linkovi su jednostavno dodatna imena fajlova i u tom slučaju drugi argument (novo ime) mora ukazivati na lokaciju u fajl sistemu koja je fizički na istoj particiji kao i izvorišni fajl.

Važno!

Hard linkovi ne mogu biti kreirani za direktorijume, već samo za regularne fajlove!

```
$ ln staro_ime novo_ime
```

Soft linkovi su samo pokazivači na neki drugi fajl ili direktorijum. Za njih ne postoje ograničenja po pitanju gde se izvorišni fajl nalazi u odnosu na sam link:


```
$ ln -s staro_ime novo_ime
```

U ovom obliku, novo_ime ne sme da postoji, a staro_ime mora postojati. Međutim, možemo forsirati kreiranje soft linka koji ukazuje na neki fajl ili direktorijum koji ne postoji:

```
$ ln -sf nepostojece_ime novo_ime
```

I u slučaju da soft link ukazuje na direktorijum, link se briše komandom `rm` kao običan fajl.

file – dobijanje informacija o sadržaju fajla

Komanda `file` prikazuje kog je tipa **sadržaj** fajla. Primer:

```
$ file /bin/mv
/bin/mv: ELF 64-bit LSB executable, x86-64, version 1 (SYSV),
dynamically linked (uses shared libs), for GNU/Linux 2.6.24,
BuildID[sha1]=0xe438743eb3051f02aff0dd6e051e7ad7d035c286,
stripped
veselin@rnd:~$ file /etc/passwd
/etc/passwd: ASCII text
veselin@rnd:~$
```

'Magija' ove komande se krije u fajlu `/usr/share/magic` (`/usr/share/misc/magic` kod Debian i izvedenih distribucija). Ovaj fajl sadrži pravila koja komanda `file` primenjuje da bi detektovala sa sigurnošću tip sadržaja fajla.

type – dobijanje informacija o komandama

Komanda `type` je interna komanda komandnog interpretera i ona nam može dati informacije o samoj komandi, tj. da li je komanda interna, eksterna ili alias za drugu komandu, kao i gde se nalazi izvršni fajl komande. Primeri:

```
$ type ls
ls is aliased to `ls --color=auto'
$ type mkdir
mkdir is /bin/mkdir
$ type cd
cd is a shell builtin
```

find – pretraživanje direktorijuma i nalaženje fajlova

Komanda `find` je jedna od moćnijih komandi na Linux-u i služi da locira fajlove po nekom zadatom uslovu. Ovaj uslov može biti kompleksan, tj. da se sastoji od više jednostavnijih uslova povezanih logičkim operacijama 'i', 'ili' i 'ne'. Uslovi se odnose isključivo na

metapodatke o fajlu, tj. ne odnose se na sadržaj fajla. Komanda `find` može i da primeni neku drugu komandu na pronađene fajlove. Fajlovi se pretražuju u realnom vremenu.

Komanda `find` treba da uvek ima dva argumenta, od kojih oba mogu biti složeni.

§ find putanja... izraz

Prvi argument je uvek spisak putanja koje se pretražuju (mada korisnici najčešće zadaju samo jednu putanju). Ove putanje predstavljaju početne direktorijume koji se onda rekurzivno pretražuju.

Drugi argument je složeni izraz kojim se u stvari postavlja uslov pretrage. Sam izraz je sačinjen od opcija, testova i akcija. Opcije definišu izračunavanje testova. Testovi se primenjuju na svakom pronađenom fajlu i vraćaju vrednost 'tačno' ili 'netačno'. Akcije definišu šta se radi sa fajlovima koji prođu testove. Opcije, testovi i akcije su povezani logičkim operatorima. Ako se operatori izostave (što je čest slučaj) podrazumeva se operator `-and` koji označava logičku operaciju 'i'. Ostali operatori su `-or` (logičko 'ili') i `!` (logičko 'ne'). Radi računanja izraza opcije i akcije uvek imaju vrednost 'tačno'.

Neke bitnije opcije su:

| Opcija | Značenje |
|--------------------|---|
| -daystart | Vremenski testovi (vidi dalje u tekstu) se računaju od početka tekućeg dana a ne od pre 24 časa |
| -maxdepth n | Idi ukupno u dubinu n nivoa. <code>-maxdepth 0</code> označava da se procesiraju samo direktorijumi navedeni u putanji |
| -mindepth n | Procesirani fajlovi moraju biti na minimalnoj dubini n od direktorijuma navedenih u putanji. <code>-mindepth 1</code> označava da će se procesirati svi fajlovi i direktorijumi unutar direktorijuma navedenih u putanji, ali ne i sami direktorijumi navedeni u putanji (suprotno od <code>-maxdepth 0</code>). |

Testovi predstavljaju u osnovi uslove kojima pretražujemo fajlove. Odnose se na različite metapodatke. Ukoliko imaju numerički argument onda je njegovo značenje sledeće:

- `n` – označava tačno `n` nečega
- `+n` – označava više od `n` nečega
- `-n` – označava manje od `n` nečega

Najčešće korišćeni testovi su:

| Test | Značenje |
|-----------------|---|
| -amin n | Fajlu je pristupano pre <code>n</code> minuta. |
| -atime n | Fajlu je pristupano pre <code>n*24</code> časa. |

| | |
|--|---|
| -cmin n | Status fajla je promenjen pre n minuta. |
| -ctime n | Status fajla je promenjen pre n*24 časa. |
| -mmin n | Sadržaj fajla je poslednji put menjan pre n minuta. |
| -mtime n | Sadržaj fajla je poslednji put menjan pre n*24 časa. |
| -anewer file -cnewer file -newer file | Isto kao -atime , -ctime i -mtime ali se umesto numeričkog argumenta uzima kao parametar odgovarajuće vreme koje ima zadati fajl file |
| -empty | Fajl ili direktorijum su prazni |
| -group name | Fajl pripada grupi name (name može biti i numerički GID) |
| -user uname | Fajl pripada korisniku uname (uname može biti i korisnikov UID) |
| -nogroup | Fajl pripada grupi koja ne postoji na sistemu |
| -nouser | Fajl pripada korisniku koji ne postoji na sistemu |
| -name ime -iname ime | Fajl/direktorijum ima ime ime. U imenu se mogu koristiti wildcard znaci. Varijanta -iname ne pravi razliku između malih i velikih slova. |
| -type t | Tip fajla je t. Parametar t može biti b za blok specijalne fajlove, c za karakter specijalne fajlove, d za direktorijume, p za imenovane FIFO pajpove, f za obične fajlove, l za simboličke linkove i s za sokete. |
| -perm mod -perm -mod -perm /mod | Prava pristupa na fajlu su: tačno mod; fajl ima sva prava navedena u -mod; fajl ima bilo koje od prava navedenih u /mod. mod može biti zadat numerički ili simbolički (vidi 'Prava pristupa - detaljnije' dalje u tekstu) |
| -size n[suffix] | Fajl je veličine n bajtova. Za veličinu se mogu koristiti sufixi: 'k' za klio, 'M' za mega i 'G' za giga. |

Ukoliko nijedna akcija nije zadata, podrazumeva se akcija **-print**. Akcija definiše šta se radi sa pronađenim fajlom. Ostale češće korišćene akcije su:

| Akcija | Značenje |
|-------------------------|---|
| -print | Ispiši pun naziv fajla |
| -ls | Za svaki pronađeni fajl izvrši komandu ls -dils . |
| -exec komanda \; | Za svaki pronađeni fajl izvrši komandu komanda. Ukoliko komanda zahteva argument koji je ime pronađenog fajla, onda se na mesto tog argumenta piše {}. Komanda mora da se zavr{} . Komanda obavezno mora da se završi sa \; |

Primeri find komande:

- da bi pronašli fajlove koji se završavaju sa '.jpg' u tekućem direktorijumu i svim njegovim poddirektorijumima:

```
$ find . -name "*.jpg"
```

- da bi pronašli sve fajlove koji se završavaju sa '.conf' i modifikovani su u poslednjih 7 dana, ispod /etc direktorijuma:

```
$ find /etc -name "*.conf" -mtime -7
```

(ovo je ekvivalentno sa: `$ find /etc -name "*.conf" -and -mtime -7`)

- da bi pronašli sve fajlove koji se **ne** završavaju sa '.jpg' **ili** sa '.gif' u poddirektorijumu Pictures/ ispod početnog direktorijuma korisnika:

```
$ find ~/Pictures ! \( -name "*.jpg" -or -name "*.gif" \)
```

Važno!

Uslovi u izrazu se mogu kombinovati i grupisati zagradama. No, pošto zagrade imaju specijalno značenje u komandnom interpreteru, u find komandi se moraju navoditi kao `\(i \)`!

- da bi izlistali sve fajlove nad kojima vlasnik ima sva prava komandom 'ls -l':

```
$ find . -perm -u+rwX -exec ls -l {} \;
```

Pitanja i zadaci

1. Kako da dobijemo spisak svih fajlova u tekućem direktorijumu u jednoj koloni?
2. Kako prepoznamo šta su regularni fajlovi u `ls -l` ispisu?
3. Zašto direktorijumi uvek imaju više od 1 reference?
4. Ako u komandi `cp` navedemo više od dva argumenta, da li postoji neko ograničenje za poslednji argument?
5. Zašto se ista komanda koristi za prebacivanje i preimenovanje fajlova?
6. Da li komandom `rm` možemo obrisati direktorijum u kojem se nalazi drugi, prazan, poddirektorijum?

7. Kreirajte neki simbolički link, a zatim izlistajte sa `ls -lF` direktorijum u kojem ste kreirali taj link. Uporedite dužinu linka i putanju na koju ukazuje link. Šta zaključujete?
8. Skinite sa interneta neki fajl sa slikom. Komandom `file` proverite šta taj fajl sadrži. Preimenujte fajl u `proba.txt`. Probajte komandu `file` na `proba.txt`. Šta dobijate?
9. Pronađite sve direktorijume ispod `/usr` koji imaju početno slovo imena `l`.
10. Izlistajte sve fajlove ispod `/usr/share` koji su veći od 1MB.

Rad sa direktorijumima

cd – kretanje kroz stablo direktorijuma

Kretanje kroz direktorijume se ostvaruje komandom cd:

```
$ cd /usr/local  
$
```

Ukoliko se komanda cd navede bez argumenata korisnik će biti prebačen u svoj početni direktorijum. Argument komande može biti apsolutna putanja ili putanja relativna u odnosu na tekući direktorijum. Znak '~' se može koristiti kao zamena u putanji za početni direktorijum. Zamena '~korisnik' označava početni direktorijum korisnika 'korisnik'.

pwd – ispis tekućeg direktorijuma

Komandom pwd se dobija informacija koji je tekući direktorijum:

```
$ cd /usr/local  
$ pwd  
/usr/local  
$
```

Ova komanda ne prihvata argumente.

pushd, dirs i popd – manipulacija stekom direktorijuma

Komandni interpreter nudi mogućnost da 'zapamti' pojedine direktorijume. Za 'pamćenje' koristi stek-strukturu, gde se prvi zapamćeni direktorijum uvek nalazi na kraju liste, a poslednji na početku. Ova manipulacija direktorijumima je zgodna kada se nalazite u nekom direktorijumu čija je putanja prilično dugačka, a privremeno morate da se pozicionirate na neki drugi direktorijum (obično u zasebnom podstablu), odradite neku komandu u tom direktorijumu i ponovo se vratite u direktorijum odakle ste pošli. Za manipulaciju stekom direktorijuma i kretanjima koriste se komande pushd i popd, dok sa dirs možete videti šta se u tom trenutku nalazi na steku. Osnovni manipulacija stekom je prikazana na sledećem primeru:

```
$ pwd
/usr/local/lib/mylibs
$ pushd .
/usr/local/lib/mylibs /usr/local/lib/mylibs
$ cd /var/log/httpd
$ pwd
/var/log/httpd
$ dirs
/var/log/httpd /usr/local/lib/mylibs
$ popd
/usr/local/lib/mylibs
$ pwd
/usr/local/lib/mylibs
```

Napomena

Komande 'cd', 'pwd', 'pushd', 'dirs' i 'popd' su interne komande komandnog interpretera. Za razlog videti poglavlje **Procesi**.

mkdir – kreiranje direktorijuma

Kreiranje novog direktorijuma se obavlja komandom 'mkdir':

```
$ mkdir dir1 dir2/dir3
$
```

Komanda 'mkdir' mora imati makar jedan argument. Argumenti komande 'mkdir' su putanje do direktorijuma koje želimo da kreiramo. Ukoliko zadata putanja sadrži poddirektorijume, oni moraju postojati. Opcijom '-p' možemo kreirati i nedostajuće nadređene direktorijume. Argumenti komande 'mkdir' mogu biti apsolutne ili relativne putanje (u odnosu na tekući direktorijum). Mogu se koristiti iste zamene kao kod 'cd' komande.

```
$ mkdir -p ~/subdir/subsubdir
```

U gornjem primeru, ako u početnom direktorijumu korisnika ne postoji direktorijum 'subdir' on će biti kreiran, a zatim će se u njemu kreirati direktorijum 'subsubdir'.

rmdir – brisanje praznih direktorijuma

Brisanje direktorijuma se obavlja komandom 'rmdir'. Ova komanda ne može da radi rekurzivno i zahteva da je direktorijum koji želimo da obrišemo prazan, tj. da u njemu ne postoje fajlovi i drugi direktorijumi (osim '.' i '..').

Pitanja i zadaci

1. Navedite dva načina na koji možete da se pozicionirate na svoj početni (home) direktorijum.
2. Jednom komandom kreirajte dva direktorijuma, A i B, od kojih svaki treba da ima po dva poddirektorijuma, Prvi i Drugi.
3. Da li komandom rmdir možemo obrisati direktorijum u kojem se nalazi drugi, prazan, poddirektorijum?

Regularni izrazi

Regularni izrazi (engl. *regular expressions*, skraćeno **regex** ili *regex*) predstavljaju metod opisivanja nekog skupa stringova. Alternativna definicija je da pomoću regularnih izraza iz jednog skupa stringova izdvajamo neki podskup koji je opisan datim regularnim izrazom. Ovo su možda prilično apstraktne definicije, ali posledica iste je relativno lako shvatljiva: uz pomoć regularnih izraza možemo iz jednog fajla izdvojiti sve stringove ili linije koje odgovaraju datom regularnom izrazu.

Regularni izrazi su moćan alat za pretraživanje i manipulaciju tekstem. Nekoliko programskih jezika kao što su Perl, AWK, Ruby i Tcl uključuju regularne izraze u samo jezgro jezika, dok većina drugih programskih jezika obezbeđuje podršku za regularne izraze u svojim bibliotekama.

Linux adminis se dele na one koji su gospodari regularnih izraza (manjina) i one koji regularne izraze mrze iz dna duše. No, bez panike – ova materija je savladljiva i sa vremenom ćete steći iskustvo i kreiranje regularnih izraza će vam biti sve lakše i brže.

Regularni izrazi na prvi pogled izgledaju kao wildcard izrazi na steroidima, što nije daleko od istine. No, specijalna značenja nekih znakova u wildcard izrazima imaju drugačije značenje u regularnim izrazima, što često zbunjuje početnike.

Da bi se stvar dodatno zakomplikovala, postoji nekoliko vrsta regularnih izraza, međusobno veoma sličnih ali sa suptilnim razlikama:

- osnovni POSIX regularni izrazi
- prošireni POSIX regularni izrazi
- Perl regularni izrazi
- Vim regularni izrazi

Regularni izrazi se, kao i bilo koji string, sastoje od znakova. Za razliku od običnih stringova, gde svaki znak predstavlja samog sebe, kod regularnih izraza znake delimo u dve grupe:

- **literali** – znakovi koji predstavljaju sami sebe
- **metaznaci** (engl. *metacharacters*) – znaci koji imaju posebno značenje. Svaki metaznak može da se 'pretvori' u literal ako se ispred njega stavi znak '\' (iz ovog logično sledi da je i sam '\' metaznak, pa ako želimo literal '\' onda pišemo '\\').

Da ništa nije tako jednostavno, postoje neki znaci koji su bez vodeće '\' literali, a sa '\' ispred sebe postaju metaznaci (ovo važi za osnovne POSIX regularne izraze)!

Literali

Svi printabilni znaci, izuzev male grupe metaznakova, su literali:

- mala i velika slova

- cifre
- neki znaci interpunkcije

Kao što rekosmo, metaznaci sa vodećim '\' takođe postaju literali.

Primer regularnog izraza koji se sastoji samo od literala su:

- 123 – mečuje string '123', ali ne i '01234'.
- Linux – mečuje string 'Linux' ali ne i 'linux' (mala i velika slova se razlikuju!)

Metaznaci

Metaznaci u osnovnim POSIX regularnim izrazima

Metaznaci imaju specijalno značenje u regularnim izrazima:

- . - mečuje bilo koji string dužine 1 (izuzev znaka za novi red)
- [] - ova dva metaznaka definiše podskup znakova koji mečuju string dužine 1, tj. jedan znak. Podskup je definisan kao niz literala (npr. [acd13] mečuje bilo koji od stringova 'a', 'c', 'd', '1' ili '3'), opseg znakova (npr. [a-z] mečuje bilo koje malo slovo engleske abecede), ili kombinacija prethodna dva (npr. [123a-c] mečuje bilo koji string '1', '2', '3', 'a', 'b' ili 'c').
- [^] - takođe definiše skup kao i [] ali mečuje sve što **ne pripada** zadatom skupu. Tako, na primer, regularni izraz [^a-z] mečuje bilo koji string dužine 1 koji nije malo slovo.
- ^ - ovaj metaznak, kada nije unutar uglastih zagrada ne mečuje nijedan znak, već početak stringa. Njegova uloga je da 'pozicionira' regularni izraz.
- \$ - ovaj metaznak takođe ne mečuje nijedan string već kraj stringa. I njegova uloga je da 'pozicionira' regularni izraz.
- \ (\) - ova dva metaznaka takođe idu u paru i služe da grupišu deo regularnog izraza u podizraz, koji se kasnije može referencirati (vidi sledeću stavku). Na primer, abc i \ (ab\) c su dva regularna izraza koja oba mečuju string 'abc', ali u drugom slučaju smo označili ab kao podizraz koji kasnije možemo referencirati.
- \ n - n je cifra 1 do 9, a ovaj metaznak referencira n-ti podizraz. Na primer, regularni izraz (ab) c \ 1 mečuje string 'abcab', ali ne i 'abc'.
- * - ovaj metaznak označava da se znak pre njega (bilo da je literal ili neki drugi metaznak) može ponoviti 0 ili više puta. Na primer, izraz ab * c mečuje stringove 'ac' (b se ponavlja 0 puta), 'abc' (b se ponavlja jednom), 'abbc' (b se ponavlja 2 puta), 'abbbc' itd.)

Važno!

Uočite razliku u značenju znaka '*' u wildcard izrazima i u regularnim izrazima!

- $\{m, n\}$ – slično kao *, ali definiše da se prethodni znak ponavlja najmanje m a najviše n puta. Primer: $a\{2, 4\}$ mećuje stringove 'aa', 'aaa' i 'aaaa' ali ne 'a' ili 'aaaaa'.

Metaznaci kod proširenih POSIX regularnih izraza

Prošireni POSIX regularni izrazi, suprotno od osnovnih POSIX regularnih izraza, znakove $(,), \{, \}$ tretiraju kao metaznake a $\backslash(, \backslash), \backslash\{, \backslash\}$ kao literale. Oni takođe definišu još neke metaznake:

- $?$ - označava da se prethodni znak ponavlja 0 ili 1 put. Na primer, izraz $ab?c$ mećuje stringove 'ac' i 'abc' ali ne i 'abbc'.
- $+$ - označava da se prethodni znak može ponavljati 1 ili više puta. Na primer ab^+ mećuje 'a', 'ab', 'abb' itd. ali ab^+ ne mećuje 'a' već samo 'ab', 'abb' itd.
- $|$ - ovaj znak spaja dva regularna izraza i kreira novi izraz koji onda mećuje sve stringove koje mećuje regularni izraz levo od znaka $|$ i sve stringove koji mećuje regularni izraz desno od znaka $|$. Tako, izraz $ab?c$ mećuje samo 'ac' i 'abc', a $de\{2, 3\}f$ mećuje 'deef' i 'deeff', ali $ab?c|d\{2, 3\}f$ mećuje 'ac', 'abc', 'deef' i 'deeff', ali ne i 'acdeef', 'abcdeef', 'acdeeff' ili 'abcdeeff'.

Klase znakova

Klase znakova su takođe metaznaci i mećuju stringove dužine 1. Najjednostavnije je da klase znakova tretirate samo kao drugi način da se navede izraz oblika '[<svi znaci koji pripadaju datoj klasi>]'.

U sledećoj tabeli su prikazane klase znakova u POSIX regularnim izrazima i u izrazima za Vim editor teksta (trebaće nam za zadnje poglavlje), šta oni znače u ASCII alfabetu i njihov opis:

| POSIX | Vim | ASCII | Značenje |
|--------------------------|--------------------------|------------------------------|--|
| <code>[:alnum:]</code> | | <code>[A-Za-z0-9]</code> | Alfanumerički znakovi |
| | <code>\w</code> | <code>[A-Za-z0-9_]</code> | Alfanumerički znakovi plus '_' |
| | <code>\W</code> | <code>[^A-Za-z0-9_]</code> | Znaci koji ne pripadaju rečima |
| <code>[:alpha:]</code> | <code>\a</code> | <code>[A-Za-z]</code> | Slova |
| <code>[:blank:]</code> | <code>\< \></code> | | Granice reči |
| <code>[:cntrl:]</code> | | <code>[\x00-\x1F\x7F]</code> | Kontrolni znaci (imaju ASCII kod ispod 32) |
| <code>[:digit:]</code> | <code>\d</code> | <code>[0-9]</code> | Cifre |

| | | | |
|------------|-----|--------------|---|
| | \D | [^0-9] | Sve izuzev cifri |
| [:graph:] | | [\x20-\x7E] | Vidljivi znaci (suprotno od [[:cntrl:]]) |
| [:lower:] | \l | [a-z] | Mala slova |
| [:print:] | \p | [\x20-\x7E] | Printabilni znaci (vidljivi znaci plus ' ') |
| [:punct:] | | | Interpunkcijski znaci |
| [:space:] | _s | [\t\r\n\v\f] | Beline |
| [:upper:] | \u | [A-Z] | Velika slova |
| [:xdigit:] | \x | [A-Fa-f0-9] | Heksadecimalne cifre |

U POSIX izrazima, klase znakova se mogu koristiti samo unutar uglastih zagrada. Na primer, `[[:digit:]]abc` mećuje cifre i 'a', 'b' i 'c', dok `[[:digit:]]abc` mećuje 'abc', 'dabc', 'iabc', 'gabc' i 'tabc'.

Mećovanje regularnih izraza

Kaže se da su regularni izrazi 'pohlepni' (engl. greedy), što znači da teže da mećuju što duži string. To se odnosi na metaznak *, koji implementira tu 'pohlepnost'. Na primer, ako u sledećem tekstu:

```
<table>
  <tr>
    <td>##param1##</td><td>##param21##</td>
  </tr>
</table>
```

želimo da referenciramo `##param1##`, onda sledeći regularni izraz **neće** odraditi svoj posao:

```
##.*1##
```

jer će ovaj izraz mećovati deo teksta `'##param1##</td><td>##param21##'`.

Da bismo ovo sprečili potrebno je modifikovati prethodni regularni izraz i u njega ubaciti precizniju specifikaciju koji znaci su prihvatljivi između dva stringa '##'. Pošto nije prihvatljiv znak '#' jer on pripada delimiteru, onda možemo da prepravimo izraz u:

```
##[^#]*##
```

Ovo u nekim slučajevima, gde su primnjeni moderniji regularni izrazi, može da se napiše i kao:

##.*?##

Ovde ? ne označava da se prethodni znak ponavlja 0 ili 1 put, već sekvenca '*?' označava tzv. 'lenji' (engl. lazy) kvantifikator. Kvantifikator je 'lenji' jer će se zadovoljiti najmanjim stringom koji mećuje dati izraz, za razliku od normalnog, 'pohlepnog' izraza koji teži da mećuje najduži mogući niz znakova.

Pitanja i zadaci

1. Šta je '(' kod osnovnih POSIX regularnih izraza?
2. Koja je razlika između '*' i '+' kod osnovnih POSIX regularnih izraza?
3. Koja je razlika između '*' i '+' kod proširenih POSIX regularnih izraza?
4. Zadat je spisak telefonskih brojeva:

+381111123456
+381631234567
+381659876543

Napišite regularne izraze kojim ćete ovaj zapis pretvoriti u sledeći:

(011) 123-456
(063) 123-4567
(065) 987-6543

5. Prethodni zadatak uradite pomoću osnovnih POSIX regularnih izraza, proširenih POSIX regularnih izraza i Vim regularnih izraza.

Komande za rad sa sadržajem fajlova

cat – ispis sadržaja fajla

Komanda `cat` prikazuje sadržaj nekog fajla ili nadovezuje (engl. *concatenate*, odatle ime) više fajlova. Standardno, `cat` rezultat šalje na standardni izlaz odakle ga možete preusmeriti u neki fajl.

Primeri:

```
$ cat prvi.txt
Ovo je prva linija.
Ovo je druga linija.
$ cat drugi.txt
Ovo je fajl 'drugi.txt'

Iznad mene je prazna linija.
$ cat prvi.txt drugi.txt
Ovo je prva linija.
Ovo je druga linija.
Ovo je fajl 'drugi.txt'

Iznad mene je prazna linija.
$
```

Komanda `cat` se može iskoristiti i kao prost editor teksta. Ako kao argument, umesto imena fajla, napišemo `-`, `cat` će učitavati liniju po liniju sa standardnog ulaza i ispisivati je na standardni izlaz, sve dok korisnik ne unese oznaku za kraj unosa (`CTRL-D`):

```
$ cat - > neki_fajl.txt
Ovo je primer upotrebe 'cat' komande kao editora.

Ovo je treća linija teksta.<CTRL-D>
$ cat neki_fajl.txt
Ovo je primer upotrebe 'cat' komande kao editora.

Ovo je treća linija teksta.
$
```

(<CTRL-D> znači da treba da pritisnete istovremeno tastere CTRL i D)

more i less – ispis sadržaja fajla stranu po stranu

Komande `more` i `less` su standardne implementacije tzv. pager-a, tj. komandi koje prikazuju tekst ekran po ekran. Obe komande omogućavaju pretraživanje teksta, kretanje red po red unapred ili unazad. Osnovne komande unutar ovog interaktivnog programa su:

- g< ili ESC-< - pozicioniranje na vrh strane
- G> ili ESC-> - pozicioniranje na dno strane
- SPACE - napred jedan ekran
- b - nazad jedan ekran
- RETURN - napred jednu liniju
- y - nazad jednu liniju
- /pattern - pretraživanje napred po regularnom izrazu pattern
- ?pattern - pretraživanje nazad po regularnom izrazu pattern
- n - sledeće pojavljivanje tražene reči
- N - prethodno pojavljivanje tražene reči
- ng - postavljanje na liniju n teksta
- v - pozivanje editora

head – ispis početka fajla

Ako je potrebno prikazati početak nekog tekstualnog fajla, onda za to služi komanda head. Ova komanda standardno prikazuje prvih 10 linija teksta iz zadatih fajlova. Ako se argumenti izostave ili se umesto njih upiše -, head čita sadržaj sa standardnog ulaza.

Najčešće korišćena opcija je -n N koja prikazuje N linija umesto standardnih 10:

```
$ head -n 4 /etc/passwd
```

tail – ispis kraja fajla

Komanda tail je pandan komandi head, s tim što tail prikazuje kraj fajla. Ova komanda je posebno zgodna jer radi u realnom vremenu i može da radi kontinualno, ako se pozove sa opcijom -f. U tom slučaju komanda konstantno nadgleda navedeni fajl i čim se u taj fajl upiše nova linija, tail ju je prikazuje. Primer:

```
$ tail -f /var/log/httpd/access_log
```

wc – brojanje znakova, reči i linija u tekstu

Komanda wc (engl. *word count*, brojanje reči) prikazuje broj znakova, reči i linija u tekstu. Reč se u ovom slučaju definiše kao niz znakova omeđen belinama (početkom ili krajem teksta, znakom tab i razmakom, ili kombinacijom istih). Kao argument prima jedan ili više fajlova:

```
$ wc [opcije] [fajl ...]
```

Opcije su:

| Opcija | Značenje |
|-----------|-----------------------------|
| -l | Prikazati samo broj linija. |
| -w | Prikaži samo broj reči. |
| -c | Prikaži samo broj znakova. |

sort – sortiranje linija u fajlu

Komanda `sort` sortira fajlove po leksičkom ili numeričkom principu. Ako se pozove bez argumenata, sortira ono što dobija iz standardnog ulaza. Ako se pozove sa više argumenata, prethodno spaja fajlove koji su joj navedeni kao argumenti i onda dobijeni spojeni fajl sortira. Rezultat ispisuje na standardni izlaz.

Format komande je:

```
$ sort [opcije] [fajl...]
```

Opcije definišu način sortiranja i ključ po kojem se sortiraju linije. Bitnije opcije su:

| Opcija | Značenje |
|----------------------|--|
| -d | Leksičko sortiranje, podrazumevano ako se nijedna vrsta sortiranja ne navede. |
| -n | Numeričko sortiranje. |
| -r | Reverzno sortiranje. |
| -b | Ignoriši beline (TAB, SPACE) |
| -f | Prebaci mala slova u velika pre sortiranja |
| -i | Ignoriši neprintabilne znakove |
| -m | Spoji fajlove bez provere da li su prethodno sortirani |
| -tX | Postavi znak X kao delimiter polja |
| -u | Ispiši samo one linije koje se ne ponavljaju |
| -kPOS1[,POS2] | Ključ za sortiranje počinje poljem POS1 a završava se sa poljem POS2 ili krajem linije, ako POS2 nije naveden. Pozicije se broje počevši od 1. POS ima format P[C][opcije], gde je P redni broj polja, C je redni broj znaka unutar polja (ako nećemo da sortiramo |

| | |
|--|---|
| | po čitavom polju), a opcije su iste one koje određuju tip sortiranja. Polja su delimitovana belinama ili argumentom opcije -t. Ukoliko se ključ sortiranja ne navede eksplicitno, podrazumeva se da je cela linija ključ. |
|--|---|

Primere za komandu sort ćemo dati koristeći kao argument fajl `/etc/passwd`, koji ima sledeći format:

```
username:password:UID:GID:name:homedir:shell
```

- Sortiranje po polju 'username':

```
$ sort -t: -f -k1,2 /etc/passwd
```

- numeričko sortiranje po polju 'UID':

```
$ sort -t: -n -k3,4 /etc/passwd
```

- numeričko sortiranje po polju 'GID', a zatim, unutar svake grupe sa istim GID-om, sortiranje po polju 'name':

```
$ sort -t: -k4n,5 -k5f,6 /etc/passwd
```

uniq – izbacivanje identičnih linija iz sortiranog teksta

Komanda `uniq` podrazumeva da joj se kao argumenti prosleđuju sortirani fajlovi ili sortiran standardni ulaz ako se poziva bez argumenata. Podrazumevano ova komanda modifikuje ispis tako što izbacuje duplikate linija (koje inicijalno moraju biti sukcesivne jer je ulazni fajl sortiran). Uz pomoć opcija, izlaz se može modifikovati drugačije:

| Opcija | Značenje |
|--------|--|
| -c | Prikaži pre svake linije broj njenih ponavljanja u ulaznom fajlu |
| -d | Prikaži samo duplicirane linije |
| -u | Prikaži samo linije koje se ne ponavljaju |

grep – pretraživanje sadržaja tekstualnih fajlova

Komanda `grep` je jedna od češće korišćenih komandi u svakodnevnom radu administratora sistema. Ovaj komanda pretražuje ulazne fajlove (navedene kao argumente ili standardni ulaz ako nijedan argument nije zadat) i prikazuje linije koje odgovaraju zadatom regularnom izrazu. Brojne opcije služe da modifikuju standardno ponašanje ove komande. Njen format je:

```
$ grep [opcije] regularni_izraz [fajl...]
```

Najpopularnije opcije ove komande su:

| Opcija | Značenje |
|-------------|--|
| -i | Ignoriši razliku između malih i velikih slova |
| -l | Ispiši samo nazive fajlova koji sadrže regularni izraz |
| -v | Prikaži samo one linije koje ne odgovaraju zadatom regularnom izrazu. |
| -A N | Prikaži N linija iza linije koja odgovara zadatom regularnom izrazu |
| -B N | Prikaži N linija pre linije koja odgovara zadatom regularnom izrazu |
| -E | Koristi proširene regularne izraze |
| -P | Koristi Perl kompatibilne regularne izraze |
| -F | Tretiraj zadati izraz kao običan string a ne kao regularan izraz. |
| -r | rekurzivno pretraživanje. Kao argument zadajte * i grep će rekurzivno pretražiti svaki fajl u svim poddirektorijumima. |

Trik sa `grep` komandom je da se zada što precizniji regularni izraz. Evo par primera:

- prikaži sve linije u kojima se nalazi niz znakova 'vic' u fajlu `/etc/passwd`

```
$ grep -F 'vic' /etc/passwd
```

- prikaži sve linije koje počinju sa slovom g u istom fajlu

```
$ grep '^g' /etc/passwd
```

- prikaži sve linije u fajlovima u tekućem direktorijumu koje ne sadrže dva ili više uzastopnih slova o

```
$ grep -E -v 'oo+' *
```

(-E je obavezno zbog upotrebe specijalnog znaka +)

split – razbijanje fajlova u manje delove

Komanda `split` deli fajl u manje fajlove. Format ove komande je:

```
$ split [opcije] [ulazni_fajl [prefiks]]
```

Argument *prefiks* definiše početni deo imena rezultujućih fajlova, tako da će se rezultujući fajlovi zvati: *prefiksaa*, *prefiksab*... *prefiksaz*, *prefiksba*, *prefiksbb*... *prefiksbz*...*prefikszz*.

Opcije komande su:

| Opcija | Značenje |
|----------------|---|
| -a N | Broj znakova u sufiksu imena treba da bude N umesto standardna 2 |
| -b SIZE | Izlazni fajlovi treba da budu dugački SIZE bajtova |
| -c SIZE | Kreiraj izlazne fajlove tako da imaju kompletne linije ukupne maks. dužine SIZE |
| -d | Koristi numeričke sufikse umesto alfabetskih |
| -l N | Kreiraj izlazne fajlove tako da imaju po N linija teksta. |

gzip i bzip2 – kompresovanje fajlova

Dva osnovna alata za kompresiju na Linux-u su `gzip` i `bzip2`. Razlikuju se po upotrebljenom algoritmu za kompresiju, pa tako `bzip2`, koji je noviji, ima nešto bolji stepen kompresije ali je `gzip` brži u radu.

Nijedan od ovih programa ne pravi arhivu već kompresuje pojedinačne fajlove, koji su navedeni kao argumenti komande. Takođe, ovi programi **brišu** originalne fajlove i zamenjuju ih njihovim kompresovanim pandanima, koji imaju ista prava pristupa i vlasništvo kao originalni fajlovi, sa dodatkom ekstenzijom `.gz` za `gzip` kompresovane fajlove, odnosno `.bz2` za `bzip2` kompresovane fajlove. Isti slučaj je i prilikom dekompresije fajlova – kompresovani fajl se briše i zamenjuje dekompresovanom verzijom, sa odstranjenom ekstenzijom.

Stepen kompresije se može navesti kao numerička opcija, i to `-0` označava najniži i najbrži stepen kompresije a `-9` je najveći stepen kompresije uz najsporiji rad.

Komande za dekompresiju su `gunzip` i `bunzip2`, koje su ekvivalentne sa `gzip -d`, odnosno `bzip2 -d`.

Posebne komande, `zcat` i `bzcat` služe da u letu raspakuju kompresovane fajlove i pošalju ih na standardni izlaz. Ove komande ne brišu originalne kompresovane fajlove.

Komande `gzip` i `bzip2` se često korsite bez argumenata i onda sadržaj koji treba kompresovati uzimaju iz standardnog ulaza, a kompresovani sadržaj šalju na standardni izlaz.

Primer:

```
$ ls
dokument.txt
$ gzip *
$ ls
dokument.txt.gz
$ gunzip *
$ ls
dokument.txt
```

tar – pravljenje arhive

Komanda `tar` je standardna komanda za pravljenje arhiva fajlova. Za razliku od brojnih sličnih komandi na Windows sistemima, `tar` podrazumevano **ne kompresuje** fajlove u arhivi. Razlog tome je što je `tar` komanda potekla sa Unix-a i inicijalno je služila da kreira arhive na trakama, odakle joj i ime (eng. *tape archiver*, arhivar na trakama). `tar` ne predviđa nikakva uputstva kako će se fajl sa arhivom zvati, ali je standard da, ukoliko arhiva nije kompresovana, fajl ima ekstenziju `.tar`. Arhive se naknadno mogu kompresovati programima `gzip` i `bzip2`, a to može i sama komanda `tar` da uradi automatski, ukoliko je pozvana sa odgovarajućom opcijom.

`tar` ima tri moda rada: kreiranje arhive, provera arhive i raspakivanje arhive, odnosno vađenje pojedinačnih fajlova iz arhive.

Osnovne opcije komande `tar` su:

| Opcija | Značenje |
|------------------|---|
| -c | Krairaj arhivu |
| -t | Testiraj (izlistaj) arhivu |
| -x | Raspakuj arhivu |
| -r | Dodaj nove fajlove u postojeću arhivu |
| -f arhiva | Koristi arhivu <code>arhiva</code> . Ukoliko želimo da korsitimo standardni ulaz ili izlaz (u zavisnosti od operacije) na mesto <code>arhiva</code> pišemo <code>-</code> . |
| -v | Detaljniji ispis programa |

| | |
|----------------|---|
| -j / -y | kompresuj/dekompresuj arhivu sa programom bzip2 (-y je opcija na Debian i izvedenim distribucijama) |
| -z | kompresuj/dekompresuj arhivu sa programom gzip |
| -C dir | Koristi direktorijum dir za operacije. |
| -p | Sačuvaj prava pristupa i vlasništvo nad fajlovima |

Ova kmdanda se često koristi pa se njene osnovne varijante lako upamte:

- Arhiviraj sadržaj /etc direktorijuma u fajl etc-arhiva.tar

```
$ tar -cvf etc-arhiva.tar /etc
```

- Isto kao prethodno, samo kompresuj programom gzip

```
$ tar -zcvf etc-arhiva.tar.gz /etc
```

- Isto kao prethodno, samo kompresuj programom bzip2

```
$ tar -jcvf etc-arhiva.tar.bz2 /etc
```

- Proveri sadržaj arhive etc-arhiva.tar.gz i izlistaj je na ekranu:

```
$ tar -ztvf etc-arhiva.tar.gz
```

- Raspakuj arhivu etc-arhiva.tar.bz2 u direktorijum /tmp:

```
$ tar -jxvf etc-arhiva.tar.bz2 -C /tmp
```

cut – ispis određenih delova svake linije formatiranog teksta

Ako imate formatiran tekst i želite da izvojite određene kolone ili delove koje želite da prikazete, onda koristite komandu cut. Ova komanda ima standardan format:

```
$ cut [opcije] [fajl ...]
```

gde su opcije:

| Opcija | Značenje |
|----------------|--|
| -bOPSEG | Prikaži samo bajtove unutar opsega OPSEG |
| -c | Prikaži samo znakove unutar opsega OPSEG |
| -f | Prikaži samo polja unutar opsega OPSEG |
| -dDELIM | Koristi znak DELIM kao delimiter polja umesto podrazumevanog znaka TAB |

Opseg može biti jednostavan ili složen, koji je sastavljen od više jednostavnih opsega razdvojenih zarezom (bez belina između!). Jednostavan opseg ima oblik:

- N – n-ti znak od početka linije, računajući prvi znak kao 1
- N- - opseg od n-tog znaka (uključivo) pa do kraja linije
- -N – opseg od prvog do N-tog znaka (uključivo)
- N-M – opseg od N-tog do M-tog znaka

Primer:

Prikaži prvih dvadeset znakova svake linije fajla test.txt:

```
$ cut -c-20 test.txt
```

Prikaži UID i GID polja fajla -/etc/passwd:

```
$ cut -d: -f3-4 /etc/passwd
```

tr – zamena znakova ili grupa znakova u tekstu

Komanda `tr` se koristi da iz standardnog ulaza obriše svaki znak zadat kao argument, ili da zameni svaku pojavu nekog znaka iz prvog argumenta, odgovarajućim znakom iz drugog argumenta. Dakle, argumenti ove komande su skupovi znakova. Ukoliko se zada samo jedan skup onda se podrazumeva da se znakovi iz tog skupa brišu, ukoliko se pojave na standardnom ulazu. Ukoliko su zadata oba skupa, pretpostavlja se da su iste dužine i onda se n-ti znak iz prvog skupa menja u n-ti znak iz drugog skupa, gde je n broj između 1 i dužine skupova. Rezultujući niz

znakova se onda prikazuje na standardnom izlazu. Drugim rečima, ova komanda radi isključivo kao filter i ne uzima fajlove za argumente.

Opcije ove komande su:

| Opcija | Značenje |
|--------|--|
| -d | Obriši svako pojavljivanje nekog znaka iz prvog argumenta u standardnom ulazu |
| -s | Zameni svako uzastopno pojavljivanje nekog znaka iz zadatog skupa sa samo jednim takvim znakom |
| -t | Smanji dužinu prvog argumenta na dužinu drugog argumenta |

Primer ove komande koristi pajp da bi preusmerio izlaz komande `cat` u ulaz komande `tr`. Primer se odnosi na zamenu svih malih slova u fajlu `/etc/passwd` sa odgovarajućim velikim slovom:

```
$ cat /etc/passwd | tr 'a-z' 'A-Z'
```

sed – neinteraktivni editor teksta

Editor `sed` (engl. *streams editor*, editor tokova) je neinteraktivni tekst editor. Njegova osnovna namena je da se koristi u složenim komandama i komandnim skriptovima gde može da obavi složenu manipulaciju teksta. Ovaj editor se obilno oslanja na regularne izraze i može odraditi prilično komplikovane izmene na tekstu, ali se najčešće koristi da izvrši pretrage i izmene na tekstu, zadajući kompleksne regularne izraze. Iako može imati fajlove kao argumente najčešće se koristi kao filter.

Kao osnovni argument, `sed` editoru se zadaje 'program' koji instruiše ovaj editor kako da manipuliše tekstem. Programi za `sed` mogu biti veoma komplektni i njihovo proučavanje se ostavlja polazniku kao vežba. Mi ćemo ovde prikazati samo jedan program koji modifikuje ulazni fajl `/etc/passwd` tako da prikaže tekst 'Korisnik <username> ima UID <UID>', gde su <username> i <UID> odgovarajuće vrednosti polja iz ulaznog fajla.

Da bismo ovo ostvarili koristimo `sed` komandu 's/original/zamena/opcije' koji implementira search & replace funkciju. Nizovi 'origina' i 'zamena' su regularni izrazi. Komanda glasi:

```
$ cat /etc/passwd | sed -e 's/^\([^:]*\):[^:]*:[0-9]*$ /Korisnik \1 ima UID \2/'
```

(Autoru ovog teksta je trebalo nekih 5 minuta i nekih 10-ak iteracija da sastavi ovaj regularni izraz)

awk – razne manipulacije tekстом

Komanda awk odrađuje sličan posao kao komanda sed – manipuliše tekстом na način opisan u programu koji se prosleđuje komandi awk. Ime ove komande je skup inicijala njenih autora (Aho, Weinberg, Keringhan) i nastala je na Unix-u još pre nego što je Linux začet. awk ima svoj programski jezik i, do pojave Perl-a, je korišćen kao alat za složene obrade teksta. Obični korisnici ga najčešće koriste za sitne manipulacije tekстом jer je nešto jednostavniji od sed-a za razumevanje.

```
$ cat /etc/passwd | awk -F: '{ print "Korisnik " $1 " ima UID " $3 }'
```

Isti primer koji smo imali kod sed-a možemo demonstrirati i sa awk-om:

diff – nalaženje razlika među fajlovima

Komanda diff prikazuje razlike između dva fajla, liniju po liniju. Postoji nekoliko načina prikaza ovih razlika ali je unificirani najpoznatiji. Ovu komandu je najbolje demonstrirati na primeru dva slična ali ne i identična fajla:

```
$ cat prvi.txt
Ovo je jednostavan test
za demonstriranje diff
komande.
$ cat drugi.txt
Ovo je jednostvan test
za demonstriranje diff
komande
$ diff -u prvi.txt drugi.txt
--- prvi.txt      2013-10-01 18:55:32.000000000 +0200
+++ drugi.txt     2013-10-01 18:55:45.000000000 +0200
@@ -1,5 +1,5 @@
-Ovo je jednostavan test
+Ovo je jednostvan test
  za demonstriranje diff
-komande.
+komande
```

Pitanja i zadaci

1. Uz pomoć komande 'cat' napravite fajl sa sledećim sadržajem:

```
Ja sam ponosni polaznik kursa
Linux L1-1 u ATC-u
```

2. Prikažite 3 početne linije fajla /etc/passwd
3. Napišite komandu za reverzno numeričko sortiranje fajla /etc/group po polju GID.

4. Uz pomoć grep komande prikažite sve korisnike koji imaju komandni interpreter /bin/bash.
5. Uz pomoć grep komande prikažite sve korisnike koji nemaju komandni interpreter /bin/bash.
6. Uz pomoć find komande, ispod /usr/share direktorijuma pronađite fajl koji je veći od 1MB. Uz pomoć komande split razbijte taj fajl na fajlove ne veće od 200K.
7. Fajl iz prethodnog zadatka kopirajte u svoj početni direktorijum, u dve kopije. Kompresujte jednu kopiju komandom gzip a drugu bzip2. Koja daje manji rezultujući fajl.
8. Dekompresujte oba fajla iz prethodnog zadatka, a zatim jedan kompresujte sa gzip -1 a drugi sa gzip -9. Koji je zaključak?
9. U svom početnom direktorijumu napravite arhivu /bin direktorijuma, tako da svi fajlovi imaju imena /bin/...
10. Arhivu dobijenu iz prethodnog zadatka raspakujte u svom početnom direktorijumu. Šta zaključujete?
11. Arhivu iz prethodnog zadatka kompresujte programom gzip. Zatim dobijeni fajl probajte da raspakujete sa tar -zxvf ime_arhive. Šta zaključujete?
12. Napišite komandu kojom zbacujete sve cifre iz fajla /etc/passwd.

Redirekcija I/O i pajpovi

Standardni ulaz, standardni izlaz i standardni izlaz za greške

Kada neki proces otvori fajl za čitanje ili za pisanje, otvorenom fajlu se u okviru datog procesa asocira broj koji nazivamo **fajl deskriptor** (u pitanju je indeks u globalnu strukturu koja pamti sve trenutno otvorene fajlove na sistemu).

Kada se proces startuje on unapred ima otvorena tri 'fajla':

- **STDIN** je standardni ulaz i označen je fajl deskriptorom 0. Kod interaktivnih procesa, u STDIN je obično preusmeren izlaz sa tastature/terminala jer preko ovog fajla proces dobija svoj input.
- **STDOUT** je standardni izlaz i označen je fajl deskriptorom 1. Kod interaktivnih procesa STDOUT je preusmeren na ekran ili prozor terminala jer preko ovog fajla proces upisuje standardne poruke.
- **STDERR** je standardni izlaz za greške i označen je fajl deskriptorom 2. Kod interaktivnih procesa STDERR je preusmeren u STDOUT, tj. na ekran ili prozor terminala. Preko ovog fajla process obično ispisuje poruke o greškama.

Redirekcija ulaza, izlaza i izlaza za greške komande

Redirekcija ulaza se vrši korišćenjem znaka <. Primer:

```
$ command < ulaz.txt
```

prikazuje komandu *command* koja umesto tastature, čita ulazne podatke iz fajla *ulaz.txt*.

Redirekcija izlaza se vrši korišćenjem znaka >. Primer:

```
$ command > output
```

kreira fajl *output* ili ga prepisuje ukoliko već postoji i u njega upisuje sve što komanda šalje na standardni izlaz. Umesto '>' može se koristiti i '1>'. Ukoliko ne želimo da ispis komande prebriše prethodni sadržaj fajla ako ovaj postoji već hoćemo da dodopišemo taj ispis na kraj postojećeg sadržaja, koristimo umesto znaka >, znak >>.

```
$ command 2> error-out
```

U gornjem primeru je preusmeren samo STDERR u fajl `error-out`, dok će standardni ispis i dalje biti prosleđen na ekran ili prozor terminala. Sadržaj fajla `error-out` će biti prebrisan, ako fajl već postoji. Ako ovo želimo da izbegnemo, koristićemo `2>>` umesto `2>`.

Moguće je kombinovati ova preusmeravanja:

```
$ command > output.txt 2> error.txt < ulaz.txt
```

U gornjem primeru, komanda `command` je u potpunosti odvojena od ekrana i tastature, čak iako je komanda interaktivna. Ovakav oblik naredbi se često koristi kod tzv. batch obrade, gde se komanda šalje na odloženo izvršavanje ili izvršavanje u pozadini.

Pajpovi

Izlaz jedne komande može biti poslat drugoj komandi korišćenjem znaka `|`. Ovaj simbol je poznat kao **pajp** (engl. *pipe*). Pajp povezuje izlaz jednog procesa na ulaz drugog. Podaci koji čekaju da budu prosleđeni su smešteni u bafer. Oba procesa se izvršavaju istovremeno. Linux vodi računa da procesi prate jedan drugog. Primer:

```
$ sort fajl.txt | uniq | mail user@example.org
```

prikazuje omandu `sort` koja sortira alfabetski, po rastućem nizu, liniju po liniju fajla `fajl.txt` i rezultat prosleđuje, umesto na ekran, standardnom ulazu komande `uniq` koja izbacuje duplikate i rezultat, umesto na ekran, prosleđuje kao ulaz komandi `mail` koja taj rezultat šalje na zadatu adresu.

Korišćenje pajpova je jedna od najjačih osobina Linux-a i na ovaj način se mogu jednostavno obaviti prilično složene obrade podataka.

Pitanja i zadaci

1. Uz pomoć komandi `head` i `tail` prikažite treću liniju od kraja fajla `/etc/passwd`.
2. U direktorijumu `/bin` prebrojte koliko ima fajlova. Koristite komande `ls` i `wc`.

Korisnici i grupe

Već smo ranije rekli da je Linux višekorisnički sistem. Korisnici su vlasnici procesa i fajlova. Takođe, korisnike smeštamo u grupe. Svaki korisnik mora biti član najmanje jedne grupe, a maksimalno može biti član osam grupa. Na modernijim Linux distribucijama uobičajeno je da se za svakog korisnika kreira zasebna grupa koja se ima isto ime kao username korisnika.

Korisniku je lakše da zapamti svoj username (koji je obično smislen) nego neki ceo broj pod kojim ga sistem vidi. Taj broj se naziva **UID** (engl. *User ID*). UID je jedinstven na jednom sistemu za svakog korisnika i nije tajna informacija.

Grupe su takođe definisane preko svojih celih brojeva koji se zovu **GID**-ovi (engl. *Group ID*).

Razlog zašto sistem definiše korisnike i grupe preko celih brojeva je taj što ceo broj zauzima uvek isti broj bajtova, dok imena korisnika i grupa mogu biti proizvoljno dugačka.

Mapiranje 'username → UID', kao i neke druge bitne informacije se standardno čuvaju u fajlu `/etc/passwd`. Ovaj fajl ima definisan format:

- svaka linija predstavlja jednog korisnika
- nisu dopuštene linije koje su prazne ili koje nisu u zadatom formatu
- svaka linija podeljena je na sedam polja razdvojenih znakom ':' - to su
 - username
 - password (više se ne čuva tu nego u drugom fajlu)
 - UID
 - Login GID (početna korisnikova grupa)
 - tzv. GECOS polje, može biti prazno a obično sadrži ime i prezime korisnika – sadržaj je proizvoljan
 - početni (engl. home) direktorijum korisnika. U ovom direktorijumu se korisnik nalazi kada se uloguje na sistem i smatra se da je to njegova 'kuća' na sistemu.
 - puno ime programa koji se izvršava kao komandni interpreter korisnika

Informacije o grupama i članstvu se čuvaju u zasebnom fajlu, `/etc/group`. Ovaj fajl ima sličan format kao `/etc/passwd`, s tim što su polja drugačija:

- ime grupe
- password (obično prazno, lozinka je namenjena da se korisnik autentifikuje kao član grupe kada istu menja u toku rada).
- GID

- spisak članova grupe – u obliku liste username-ova razdvojenih zarezom. Korisnici kojima je data grupa njihova login grupa obično nisu na tom spisku – na njega se stavljaju oni korisnici kojima je to dodatna grupa.

Prilikom rada, jedna od grupa u koje je korisnik učlanjen ima posebno značenje – ta grupa je tzv. **aktivna grupa**. Inicijalno, aktivna grupa je korisnikova login grupa a korisnik za svoju aktivnu grupu može izabrati neku drugu u toku rada. Ova grupa je bitna iz razloga što, kada korisnik kroz neki svoj proces kreira novi fajl (bilo kog tipa), aktivna grupa korisnika postaje vlasnik tog fajla (izuzeci su objašnjeni u sledećem poglavlju).

id – dobijanje informacija o korisniku

Komanda `id` služi da prikaže koji je UID nekog korisnika i kojim grupama pripada. Ako je navedete bez argumenata, dobijate informacije o sebi.

```
$ id
uid=1000(veselin) gid=1000(veselin)
groups=1000(veselin),4(adm),24(cdrom),27(sudo),30(dip),
46(plugdev),107(lpadmin),124(sambashare)
$ id root
uid=0(root) gid=0(root) groups=0(root)
```

passwd – menjanje korisnikove lozinke

Za promenu sopstvene lozinke koristi se komanda `passwd`. Ova komanda je interaktivna i ne prima nikakve argumente u ovom scenariju. Program će, iz bezbednosnih razloga, prvo tražiti da korisnik unese postojeću lozinku, a zatim novu, i to dva puta (kako bi se izbegle eventualne greške u kucanju). Prilikom unosa bilo koje lozinke ista se ne vidi na ekranu, čak se ne pojavljuju ni znaci '*' koji su uobičajeni kod npr. GUI okruženja.

```
$ passwd
Changing password for user veselin.
Changing password for veselin.
(current) UNIX password:
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
```

Administrator sistema može postaviti neka ograničenja na oblik i dužinu lozinke – obično se zahteva da lozinka ne bude kraća od nekog specificiranog broja znakova, kao i da lozinka obuhvata određene klase znakova (mala slova, velika slova, cifre, interpunkcijske znake). Ekstremno paranoični mogu uključiti i brzu proveru na 'kvalitet' lozinke upoređujući je sa nekim rečničkim stavkama i odbijajući da prihvate neke očigledne, koje npr. sadrže ime korisnika, username, datume i bilo koji drugi javno dostupan podatak o korisniku.

Napomena

Lozinka je tajna informacija i samo vi treba da je znate. Uvek birajte svoje lozinke tako da ne sadrže neke poznate podatke o vama ili sistemu na kojem radite. Trudite se da lozinke budu min. 8 znakova i da sadrže mala i velika slova, cifre i interpunkcijske znake. Ne zapisujte lozinke i pokušajte da ih pamtite mnemonički – npr. setite se nekog stiha neke pesme i onda za lozinku uzmite prva slova svake reči, menjajući neka slova u velika, cifre (npr. po sličnosti sa latiničnim slovima) i ubacite neki interpunkcijski znak.

newgrp – promena aktivne grupe

Ranije u ovom poglavlju smo napomenuli da korisnik može istovremeno biti član više grupa, ali samo je jedna u jednom trenutku tzv. aktivna grupa – grupa koja će biti vlasnik svih fajlova koje korisnik bude kreirao. No, aktivna grupa može da se menja u toku rada i komanda za to je newgrp. Format ove komande je:

```
$ newgrp [grupa]
```

Bez argumenata, ova komanda postavlja za aktivnu grupu login grupu korisnika (ona čiji je GID u /etc/passwd fajlu). Sa argumentom, aktivna grupa se menja u navedenu. Podrazumeva se da korisnik mora biti član nove aktivne grupe.

chgrp – promena grupe koja je vlasnik fajla

Korisnik može promeniti grupu koja je vlasnik fajla na fajlovima čiji je on vlasnik. Nova grupa mora biti jedna od grupa kojoj korisnik pripada. Komanda kojom se menja grupa koja je vlasnik fajla je chgrp. Ona ima dva formata:

```
$ chgrp [opcije] grupa fajl...  
$ chgrp [opcije] -reference=r_fajl fajl...
```

Od opcija, najkorišćenija je -R koja, ako je kao argument zadat direktorijum, rekurzivno menja grupu svim poddirektorijumima i fajlovima u njima, kao i direktorijumu koji je naveden kao argument. U prvom obliku komande, morate navesti grupu koja će biti novi vlasnik fajla. U drugom obliku komande grupa se ne navodi već se navodi neki postojeći fajl (r_fajl) koji služi kao referenca – grupa koja je vlasnik tog fajla će biti nova grupa vlasnik fajlova navedenih kao argumenti.

Pitanja i zadaci

1. Kojim grupama pripada vaš korisnik?

Prava pristupa - detaljnije

Osnovna prava pristupa

Već smo napomenuli da svaki fajl na Linuxu ima definisana prava pristupa koji određuju da li korisnik:

| Pravo | Značenje u slučaju regularnih fajlova | Značenje u slučaju direktorijuma |
|----------|---------------------------------------|--|
| r | može da pročita fajl | može da izlista direktorijum |
| w | može da izmeni sadržaj fajla | može da doda, obriše ili promeni ime fajlovima u direktorijumu |
| x | može da izvrši fajl | može da se pozicionira ili prođe kroz direktorijum |

Napomena

Prava pristupa na direktorijumu možda ćete bolje shvatiti ako zamislite da je direktorijum samo još jedan fajl koji se sastoji od zapisa i gde svaki zapis ima sledeći format:

pokazatelj na sledeći zapis u fajlu

pokazatelj na i-nod fajla

ime fajla.

U tom slučaju r-pravo se svodi na čitanje sadržaja direktorijum-fajla, w-pravo se svodi na menjanje sadržaja direktorijum-fajla.

Pravo x možete zapamtiti ako znate da se x na engleskom ponekad izgovara kao 'cross', što je reč koja kao imenica ima značenje 'krst' (liči na oblik slova x), ali kao glagol ima značenje 'prelaziti'.

Ova prava su zatim grupisana u tri grupe:

- prava koja pripadaju vlasniku fajla
- prava koja pripadaju grupi koja je vlasnik fajla
- prava koja pripadaju svim ostalim korisnicima

Pošto fajlovima pristupaju procesi, onda se prava računaju na sledeći način:

1. ako je efektivni UID procesa (objašnjeno u sledećoj sekciji) isti kao UID vlasnika fajla, primenjuju se prava za vlasnika fajla; inače
2. ako je GID grupe koja je vlasnik fajla u skupu efektivnih GID-ova procesa, primenjuju se prava za grupu; inače
3. primenju se prava za ostale

Kompletan set prava se na sistemu zapisuje kao ceo broj. Taj broj je najbolje predstaviti u oktalnom zapisu (oktalni zapis koristi samo cifre 0-7 i pozicije cifara su stepeni broja 8 – oktalni zapis broja uvek počinje sa 0):

0SUGO

gde su:

0 – vodeća nula koja u komandnom interpreteru (i mnogim programskim jezicima) označava da je vrednost broja zapisana u oktalnom formatu

- S – oktalna cifra koja označava specijalna prava pristupa (objašnjeno u sledećoj sekciji)
- U – oktalna cifra koja označava prava za vlasnika (engl. user)
- G – oktalna cifra koja označava prava za grupu (engl. group)
- O – oktalna cifra koja označava prava za ostale (engl. others)

Svaka oktalna cifra predstavlja zbir numeričkih vrednosti svakog od prava, ukoliko to pravo postoji:

| Pravo | Binarni zapis | Oktalni/decimalni zapis |
|----------|---------------|-------------------------|
| r | 100 | 4 |
| w | 010 | 2 |
| x | 001 | 1 |

Tako npr. ako su na nekom direktorijumu definisana sledeća prava:

- ne postoje specijalna prava
- vlasnik ima sva prava
- grupa ima pravo izlistavanja i prolaska/pozicioniranja na direktorijum
- ostali imaju samo pravo pozicioniranja/prolaska

to će se numerički prikazati kao:

```
S (---) = 0 + 0 + 0 = 0
U (rwx) = 4 + 2 + 1 = 7
G (r-x) = 4 + 0 + 1 = 5
O (--x) = 0 + 0 + 1 = 1
pravo pristupa: 00751
```

Obrnuto, ako znamo da je oktalna vrednost prava pristupa na nekom fajlu 00644, možemo razlučiti koja prava ima vlasnik, grupa i ostali:

- vlasnik: $6 = 4 + 2 + 0 \rightarrow rw-$, vlasnik ima pravo čitanja i izmene sadržaja
- grupa: $4 = 4 + 0 + 0 \rightarrow r--$, grupa ima pravo čitanja
- ostali: $4 = 4 + 0 + 0 \rightarrow r--$, ostali imaju pravo čitanja

Napomena

Oktalni zapis se koristi jer su prava pristupa prikazana kao 3-bitni broj. Jedna oktalna cifra se koduje sa tačno 3-bita. Pošto imamo četiri grupe prava pristupa (specijalni, vlasnik, grupa, ostali) to znači da je broj koji predstavlja prava pristupa enkodovan sa 12 bitova.

Specijalna prava pristupa

Specijalna prava pristupa modifikuju način pristupa samog procesa fajlu. U slučaju regularnih fajlova, specijalna prava pristupa imaju smisla samo kod izvršnih fajlova, gde menjaju efektivne UID-e i GID-ove procesa.

Ranije smo rekli da proces pamti dva UID-a: realni i efektivni. Realni UID je UID korisnika koji pokreće program i on je nepromenljiv. Efektivni UID je onaj UID koji se koristi kada se računaju različita prava koja proces ostvaruje. Inicijalno, efektivni UID je kopija realnog UID-a i za najveći broj procesa to ostaje tako. Međutim, efektivni UID je moguće promeniti specijalnim pravima pristupa (zato i postoje dva različita UID-a) – ponekad je potrebno da se prava pristupa ne izračunavaju u odnosu na stvarnog korisnika koji je vlasnik procesa, već u odnosu na nekog drugog korisnika. Taj drugi korisnik je vlasnik fajla koji je pokrenut i, dok traje izvršavanje programa koji se smešten u tom fajlu, efektivni UID procesa postaje UID vlasnika fajla. Ovo je potrebno zato što je onda moguće ograničiti prava pristupa nekim fajlovima samo na određenog korisnika a onda standardnim pravima pristupa odrediti ko ima pravo da izvrši taj fajl. Dok se fajl izvršava, i samo onda, moguće je da korisnici koji izvršavaju taj fajl pristupaju podacima koji su zaštićeni na gore opisan način. U praksi, ovo se koristi samo kada je potrebno da korisnik privremeno preuzme prava root korisnika (superuser-a, za kojeg ne važe prava prisupa) i na taj način odradi neki posao za koji inače ne bi mogao da odradi sa svojim standardnim privilegijama. Naravno, ovo preuzimanje prava superuser-a je bezbednosno veoma opasno i mora se voditi računa kojim programima se dodeljuju takva prava.

Ista logika važi i za efektivni GID aktivne grupe koju proces takođe pamti.

Specijalna prava pristupa na direktorijumu imaju slično značenje – menjaju efektivne UID-e i GID-ove, ali samo kada se u tim direktorijumima kreiraju novi fajlovi (čime pada u vodu priča o dva različita UID-a i GID-a za proces – njih ima više, ali je preopterećujuće razmišljati o tome i lakše je zamišljati da ih ima samo dva: realni i efektivni).

Specijalna prava pristupa imaju iste numeričke vrednosti kao i standardna prava (4, 2 i 1) jer ih ima tačno tri:

| Specijalno pravo | Numerička vrednost | Značenje kod izvršnih fajlova | Značenje kod direktorijuma |
|------------------|--------------------|--|--|
| setUID | 4 | Privremeno menja efektivni UID procesa u UID vlasnika fajla | Obezbeđuje da novokreirani fajlovi u direktorijumu imaju UID vlasnika direktorijuma a ne UID korisnika koji ih je kreirao |
| setGID | 2 | Privremeno menja efektivni GID aktivne grupe u GID grupe koja je vlasnik fajla | Obezbeđuje da novokreirani fajlovi u direktorijumu imaju GID grupe koja je vlasnik direktorijuma a ne GID aktivne grupe korisnika koji ih je kreirao |
| sticky bit | 1 | Ne koristi se više | Obezbeđuje da korisnik može da briše i preimenuje isključivo fajlove čiji je sam vlasnik iako ima w-pravo na direktorijumu. Koristi se kod /tmp direktorijuma gde svi korisnici imaju w-pravo kako bi zaštitio fajlove različitih korisnika. |

umask – podešavanje podrazumevanih prava pristupa

Svaki put kada kreiramo neki fajl, on dobija neka prava pristupa. Koja su to prava i da li mi nekako možemo da izmenimo da podrazumevana prava? Odgovor na ova pitanja daje interna komanda interpretera `umask`. Format ove komande je:

```
$ umask [opcije] [mod]
```

Ako zadamo ovu komandu bez argumenta `mod`, dobićemo podrazumevana prava, odnosno masku za prava pristupa.

```
$ umask
0002
```

Hm, prema ovome, samo ostali imaju pravo pisanja (2)? Ne, `umask`, bez ikakvih parametara vraća oktalnu masku – maskirane bitove u pravima pristupa koji **neće** biti postavljeni. Kako da dobijemo onda podrazumevana prava pristupa? Sračunaćemo ih po formuli:

```
$ umask
0002
0666 - 0002 = 0664
```

Dakle, podrazumevana prava pristupa u ovom slučaju su: vlasnik i grupa imaju pravo čitanja i pisanja, ostali samo pravo čitanja (izvršno pravo se nikad ne postavlja kroz `umask` niti kao podrazumevano pravo). Ovo izgleda prilično nedovršeno, zašto niko nije na kraju programa dodao ovu jednostavnu matematičku operaciju? Niko to ne zna. Srećom, u `bash` komandnom interpreteru, ako pozovete komandu `umask` sa opcijom `-S`, dobićete simbolički opisana prava pristupa:

```
$ umask -S
u=rwx,g=rwx,o=rx
```

Ovo je već puno razumljivije!

Postavljanje podrazumevanih prava pristupa se opet može odraditi preko maske koju moramo sami sračunati ili simbolički, sa opcijom `-S`. Maska, odnosno simbolička prava pristupa su argument `mod`.

Dakle, ako hoćemo da postavimo podrazumevana prava pristupa takva da:

- korisnik ima sva prava
- grupa ima samo pravo čitanja
- ostali nemaju nikakva prava

koristimo formulu:

```
0666 - podrazumevana_prava_pristupa = maska
```

U našem slučaju, podrazumevana prava pristupa su: 00640, pa je onda maska:

```
0666 - 0640 = 0026
$ umask 00026
$ umask -S
u=rwx,g=rx,o=x
```

Ako bismo koristili simboličku reprezentaciju onda bi naša komanda glasila:

```
$ umask -S u=rwx,g=rx,o=x
$ umask
0026
```

chmod – promena prava pristupa nad fajlovima

Prava pristupa na fajlovima menjamo komandom `chmod`. Ova komanda ima format:

```
$ chmod [opcije] mod fajl...
```

gdje je od opcija najčešće korišćena `-R` za rekurzivno menjanje prava u direktorijumu koji je naveden kao argument.

Argument `mod` predstavlja prava pristupa koja želimo da postavimo, dodamo ili oduzmemo fajlovima `fajl...`. Mod može biti zadat numerički, kao niz oktalnih cifara ili simbolički, slično komandi `umask`. U slučaju kada zadajemo mod numerički uvek postavljamo nova prava pristupa. Ako mod zadajemo simbolički, imamo mogućnost da dodamo, oduzmemo ili postavimo prava.

Numerički argument je jednostavan: sračunamo koja prava želimo da fajl ima i onda ih pretvorimo u oktalni broj i to zadamo kao argument. Na primer, ako želimo da fajl `test.sh` ima sledeća prava:

- vlasnik ima sva prava ($\text{rwx} \rightarrow 4 + 2 + 1 = 7$)
- grupa ima pravo čitanja i izvršavanja ($\text{r-x} \rightarrow 4 + 0 + 1 = 5$)
- ostali nemaju nikakva prava ($\text{---} \rightarrow 0 + 0 + 0 = 0$)

zadajemo komandu:

```
$ chmod 0750 test.sh
```

Ako želimo da postavimo ista ova prava na istom fajlu, ali da dodamo `setUID` i `setGID` specijalna prava, onda računamo:

- specijalna prava ($\text{setUID} + \text{setGID} = 4 + 2 + 0 = 6$)
- vlasnik ima sva prava ($\text{rwx} \rightarrow 4 + 2 + 1 = 7$)
- grupa ima pravo čitanja i izvršavanja ($\text{r-x} \rightarrow 4 + 0 + 1 = 5$)
- ostali nemaju nikakva prava ($\text{---} \rightarrow 0 + 0 + 0 = 0$)

Zadajemo prava:

```
$ chmod 06750 test.sh
```

(vodite računa da postavljanje specijalnih prava može da uradi samo root korisnik!)

Loša strana numeričkog argumenta je da ne možemo da dodamo ili oduzmemo neko pravo od postojećih. To možemo da uradimo sa simboličkim pravima. Simbolička prava imaju sledeći format:

```
ObjekatAkcijaPravo
```

gde su:

- **Objekat** – na koga se odnosi pravo. Može biti u za vlasnika, g za grupu i o za ostale, kao i kombinacija ovih znakova, ako se postavljanje odnosi na više objekata, npr. ug ili go. Ako **Objekat** izostavimo, podrazumeva se da postavljamo prava za sve, kao da smo zadali ugo.
- **Akcija** – označava da li dodajemo prava (+), ukidamo prava (-) ili postavljamo prava (=). Akcija može biti samo jedna.

Simbolička prava mogu da se grupišu tako što se razdvajaju znakom '!'.

- **Pravo** je slovna reprezentacija prava koja postavljamo, ukidamo ili dodajemo.

Na primer, ako želimo da svima dodamo pravo listanja i pristupa tekućem direktorijumu, zadajemo komandu:

```
$ chmod +rx .
```

Ako želimo da korisnik ima pravo pristupa direktorijumu Podaci./ svim poddirektorijumima i fajlovima, a grupa i ostali nikakva prava, onda možemo zadati sledeću komandu:

```
$ chmod -R u+x,go= Podaci/
```

Ako želimo da ukinemo setUID i setGID pravo svim fajlovima i direktorijumima u tekućem direktorijumu, onda zadajemo:

```
$ chmod u-s,g-s *
```

Pitanja i zadaci

1. Šta predstavlja oktalna vrednost prava pristupa 04711?
2. Promenite svoj umask tako da samo vi imate sva prava a grupa i ostali nikakva?

Rad sa procesima

Procesi u pozadini

Većina komandi se izvršava do svog kraja pre nego sto dobijete nazad prompt komandnog interpretera. Proces u 'pozadini' nastavlja da se izvršava dok vi dobijate prompt odmah po zadavanju komande. Da bi poslali komandu da se izvršava u pozadini, postavite na kraj komandne linije znak `&`:

```
$ command &
```

Kada se proces pošalje u pozadinu, njegov STDIN, STDOUT i STDERR su i dalje vezani za terminal, kao i kad bi se ista komanda izvršavala na standardan način. Iz tog razloga uvek je poželjno preusmeriti ulaz i izlaze takve komande u fajl, kako ne bi dolazilo do mešanja ispisa više komandi. Ako proces zahteva ulazne podatke od korisnika i ne može ih dobiti iz fajla, onda se on 'zaustavlja'. Proces neće nastaviti izvršavanje dok se ne pošalje u prvi plan da bi primio ulazne podatke.

Ponekad je potrebno startovati proces i ostaviti ga da radi i kad se izlogujete, odnosno zatvorite prozor terminala. Ako terminišete komandni interpreter, svi procesi u pozadini će takode biti terminisani. Komanda `nohup` može zaobići ovo tako što otkaći proces od terminala.

Važno!

Uvek preusmerite izlaz i izlaz za greške kada koristite `nohup`.

```
$ nohup sort bigfile > out.txt 2> err.txt
```

Ako ne preusmerite STDOUT i STDERR, oni će biti zapisani u fajl `nohup.out` koji će biti kreiran ili u tekućem direktorijumu (u kojem ste bili kada ste zadali `nohup...` komandu) ili u vašem početnom direktorijumu..

Komanda koja se dugo izvršava može biti privremeno zaustavljena tako što otkucate CTRL-Z. Ovu komandu zatim možete poslati da se nastavi sa izvršavanjem u pozadini tako što zadate komandu

```
$ bg
```

Komandu koja je poslata da se izvršava u pozadini možete 'vratiti' u prednji plan tako što zadate komandu:

```
$ fg
```

Komanda `fg`, bez argumenata će vratiti u prednji plan poslednju komandu poslatu u pozadinu. Ako želite da vratite neku drugu komandu koja se izvršava u pozadini u prednji plan, pozovite komandu `fg` sa argumentom koji predstavlja **JobID** komande u pozadini. Da biste dobili JobID svih komandi koje se trenutno izvršavaju u pozadini, zadajte komandu:

```
$ jobs
```

Izvršavanje niza komandi

Bash može da izvrši više komandi u jednoj liniji. Već smo videli način na koji paralelno izvršavamo više komandi koristeći pajpove. No, ukoliko nam nije potrebno da izlaz jedne komande preusmerimo u ulaz druge, onda možemo zadati više komandi razdvojenih znakom `;` (komande će biti sekvencijalno izvršavane)

```
$ command1 ; command2 ; command3
```

Više komandi može da se pošalje da se izvršava u svom zasebnom interpreteru (subshell), tako što se navedu unutar zagrada:

```
$ ( command1; command2 )
```

Komande možemo povezati u niz u kojem će startovanje sledeće komande biti uslovljeno uspešnošću izvršavanja prethodne komande. Ako npr. želimo da izvršimo komandu `command2` isključivo ako se komanda `command1` uspešno izvrši, onda koristimo znak `&&` za uslovno spajanje ove dve komande:

```
$ command1 && command2
```

Ukoliko se komanda `command1` ne izvrši uspešno, komanda `command2` uopšte neće biti startovana!

Ako želimo da izvršimo `command2` samo ako `command1` ne uspe da se izvrši uspešno, onda koristimo znakove `||` za spajane dve komande:

```
$ command1 || command2
```

U ovom slučaju, ako komanda `command1` bude uspešno izvršena, neće se izvršavati `command2`.

Moguće je kombinovati uslovna izvršavanja:

```
$ command1 && command2 || command3
```

U ovom primeru, ako `command1` bude uspešno izvršena, izvršiće se `command2` a `command3` se neće izvršavati. U protivnom, izvršiće se `command3` a `command 2` se neće izvršavati.

```
$ command1 || command2 && command3
```

U ovom primeru, ako se `command1` ne izvrši uspešno izvršiće se `command2`. Ukoliko se bilo `command1`, bilo `command2` izvrši uspešno, izvršiće se i `command3`.

Važno!

Počtnici često prave greške računajući da će se sve komande u komandnoj liniji izvršiti ili izvršavati paralelno!

Napomena

Kako komandni interpreter zna koja komanda je izvršena uspešno a koja nije? To se određuje na osnovu izlaznog statusa (engl. exit status) komande. Izlazni status je jedan celi broj koji svaki proces koji se izvršio vraća procesu koji ju je pokrenuo. Po konvenciji izlazni status 0 označava da je komanda izvršena uspešno. Izlazni status različit od nule označava da komanda/proces nije izvršena uspešno. Značenje izlaznog statusa varira od komande do komande, ali su to najčešće ID-ovi standardnih grešaka.

Upravljanje procesima

Komanda `ps` ispisuje informacije o korisnikovim procesima:

```
$ ps
  PID TTY          TIME CMD
 31944 pts/2    00:00:00 bash
 31994 pts/2    00:00:00 ps
$
```

Ova komanda ima mnogo različitih opcija, pogledajte njenu man stranu.

Komanda `wait` odlaže izvršavanje komandnog interpretera dok se ne završi proces, obično zadavajući PID procesa kao argument komande. Ako ni jedan argument nije zadat, komanda čeka dok se svi komandni interpreteri ne završe. Ova komanda se retko koristi.

Komanda `kill` se koristi da bi se poslali signali procesima. Na ovaj način možete prekinuti procese u pozadini. Neki procesi koriste signale za indicaciju određenog posla, npr. rotaciju log fajlova, ponovno učitavanje konfiguracionog fajla, itd. Komandi `kill` se može signal zadati preko njegovog imena ili broja:

```
$ kill -IME_SIGNALA PID ...
```

ili

```
$ kill -BROJ_SIGNALA PID ...
```

Najčešći signali su:

| Naziv signala | IME_SIGNALA | BROJ_SIGNALA | Standardno značenje |
|---------------|-------------|--------------|---|
| SIGHUP | HUP | 1 | Detektovano je otkaćinjanje od terminala ili smrt procesa roditelja |
| SIGINT | INT | 2 | Prekid sa tastature (npr. CTRL-Z) |
| SIGQUIT | QUIT | 3 | Prekid sa tastature |
| SIGKILL | KILL | 9 | Signal za bezuslovno okončanje procesa |
| SIGTERM | TERM | 15 | Signal za regularno okončavanje procesa |
| SIGUSR1 | - | 10,16,30 | Korisnički definsan signal 1 |
| SIGUSR2 | - | 12,17,31 | Korisnički definisan signal 2 |

Ukoliko drugačije nije zadato, kill šalje SIGTERM signal koji za najveći broj procesa znači regularno okončavanje izvršavanja. Ukoliko proces ne odgovara na signale, može se nasilno okončati slanjem SIGKILL signala.

Samo vlasnik procesa (ili super-user) može slati signale procesima.

Pitanja i zadaci

1. Izlistajte sve procese na sistemu (pogledajte man stranu za odgovarajuće opcije).
2. Izdvojte svoje procese iz spiska. Probajte da ubijete proces 'bash'.

Rad sa vim editorom

Editovanje teksta na Linuxu je mnogo zastupljenija aktivnost nego što je to na Windows-u. Razlog tome je što su konfiguracije programa na Linuxu skoro uvek tekstualni fajlovi koje je potrebno ručno menjati. Takođe, Linux omogućava mnogo veći stepen automatizacije standardnih poslova koristeći komandne skripte koje su obični tekstualni fajlovi koji sadrže komande komandnog interpretera i pozive programa.

Iz gore navedenih razloga, editor teksta je jedan od bitnijih alata na Linux-u i postoji priličan izbor ovih programa. No, jedan editor je uvek 'standardan' i može se naći na svim distribucijama – to je editor teksta vi, odnosno njegova unapređena implementacija vim.

Vi je dobio naziv od engleske reči visual, pošto potiče od najranijih dana Unix operativnog sistema (i time je stariji od Linux-a), gde je na početku standardni editor bio ed, linijiski editor (mogli ste da editujete liniju po liniju teksta, bez slobodnog kretanja kroz isti). Vi je napravljen u doba kada je osnovna komunikacija sa sistemom bila preko serijskih terminala. Terminali su poticali od različitih proizvođača i imali različite mogućnosti – neki su bili rudimentarni i imali samo osnovni deo tastature i rudimentarnu kontrolu pozicioniranja na ekranu, dok su drugi bili opremljeni punom tastaturom sa dvadesetak funkcijskih tastera koji su mogli da se programiraju i sa punim grafičkim setom znakova. Vi je morao da podrži sve vrste terminala tako da su njegovi autori morali da pribegnu svođenju upotrebljivog dela tastature na osnovni set znakova – ono što su svi terminali podržavali. Pošto nije bilo moguće koristiti funkcijske i dopunske tastere (kao što su npr. kurzorske strelice), onda je svaki dostupni znak sa tastature morao da dobije neku komandu, a pošto je taj isti znak i deo teksta, autori su odlučili da uvedu tzv. modove rada – u određenim (komandnim) modovima pritisak na neko slovo bi odradio komandu vezanu za to slovo, a u modu unošenja teksta pritisak na isti taster bi unosio to slovo u tekst (kao i kod svih modernijih editora). Ovo danas izgleda zastarelo, i mnogi korisnici Linux-a beže od vi/vim editora i koriste neke druge koji imaju 'moderniji' način rada. No, nije mali broj ni onih koji rade na drugim operativnim sistemima, kako što je Windows a koji često koriste editor teksta (kao što su programeri) koji namerno biraju Vim editor kao svoj osnovni alat! Razlog tome je što ta svedenost na osnovni deo tastature kao posledicu ima da korisnikove ruke nemaju potrebe da se pomeraju ka ostalim tasterima (ili mišu) što ubrzava rad, posebno za one koji znaju slepo da kucaju. Sa druge strane, mnogi Linux administratori su, jednom kad su savladali strmu krivu učenja vi/vim komandi, postali 'zavisni' od ovog editora i koriste ga isključivo, čak i za najduže i najzahtevnije obrade teksta. Razlog ovome je taj što vi/vim editor sadrži moćne komande za obradu teksta, a vim je dodatno i veoma programabilan i proširljiv sa plugin-ovima i sopstvenim komandnim skriptama preko kojih možete deifinisati sopstvene komande i modifikovati ponašanje ovog editora.

Vim editor je toliko složen da za njega postoje zasebne knjige koje su veće od cele ove skripte, a, npr, obuhvataju samo napredan rad sa ovim editorom. Mi ćemo u ovoj skripti obrađivati samo osnovne komande a korisniku koji se navikne na vim i zavoli ga preporučujemo da na Internetu potraži dodatne tutorijale, uputstva, opisane trikove i knjige koje će mu otkriti moćan set komandi i mogućnosti ovog editora.

Pokretanje vim editora

Vim editor se može startovati na više načina:

```
$ vi [opcije] [fajl...]
```

će pozvati vim editor koji će se onda ponašati kao vi editor, tj. nijedno proširenje koje vim editor donosi neće biti dostupno.

```
$ vim [opcije] [fajl...]
```

će pozvati vim editor sa svim poboljšanjima koje ovaj editor donosi.

Na kraju:

```
$ view [opcije] fajl...
```

će pozvati vim editor u tzv. read-only modu gde nećete moći da vršite izmenu sadržaja fajlova.

Najčešće korišćene opcije su:

- +NUM – postavlja se na liniju broj NUM u prvom navedenom fajlu
- +/PATTERN – postavlja se na prvo pojavljivanje stringa PATTERN u prvom navedenom fajlu
- -b – omogućava editovanje binarnih fajlova i programa

Kao argument se navode imena fajlova koje treba editovati, ili kreirati. Vim omogućava paralelno editovanje više fajlova koje on smešta u različite bafere. Uvek se prikazuje prvi navedeni fajl.

Modovi vim editora

Već smo napomenuli da je vim tzv. 'modalni' editor, tj. da se rad u njemu odvija u nekom od modova. Inicijalni mod, u kojem se editor uvek startuje je komandni mod.

Komandni mod

U ovom modu nije moguće unositi tekst direktno sa tastature već on služi za zadavanje komandi, kao što su komande za kretanje po tekstu, komande za brisanje, kopiranje, lepljenje teksta i sl.

Važno!

Povratak iz moda za unos teksta u komandni mod se vrši pritiskom na taster ESC ili CTRL-[, Vodite računa da u ovom poslednjem slučaju, [ne označava taster na kojem piše ovaj znak, već sam taj znak (koji će možda biti mapiran na nekom drugom tasteru ako ne koristite standardni US ASCII).

'ex' mod

Dodatni komandni mod je tzv. 'ex' mod, gde vim imitira linijski editor teksta 'ex'. U ovaj mod se ulazi isključivo iz komandnog moda, unosom znaka '!'. U dnu ekrana će se pojaviti prompt '!' iza kojeg možete otkucati odgovarajuću komandu. 'Ex' komande su oblika:

```
: [opseg] komanda argumenti
```

gde je opseg specifikacija na koji deo teksta se odnosi komanda. Opseg je zadat u obliku

```
POCETAK, KRAJ
```

ili '%' za ceo tekst. POCETAK i KRAJ su početni i krajnji broj linije teksta, uključivo, na koji komanda treba da se odnosi. Za tekuću liniju umesto broja koristite znak '!'. Za kraj teksta (poslednju liniju) koristite '\$', tako da npr. '5,\$' označava opseg od pete linije do kraja teksta, a '!, \$' označava tekst od tekuće linije do kraja teksta.

Modovi za unos teksta

Postoji nekoliko modova za unos teksta i u sve se ulazi samo iz komandnog moda. Ulaz u neki od ovih modova postiže se unosom odgovarajućeg slova (ne vidi se na ekranu):

| Znak za ulaz u mod | Mod |
|--------------------|---|
| i | insert mod – tekst se unosi u tekuću liniju, iza kursora |
| I | insert mod – tekst se unosi u tekuću liniju, od početka linije |
| a | append mod – tekst se unosi u tekuću liniju, ispred kursora |
| A | append mod – tekst se unosi u tekuću liniju, od kraja linije |
| o | open mod – ispod tekuće linije se dodaje nova, prazna linija i tekst se unosi u nju |
| O | open mod – iznad tekuće linije se dodaje nova, prazna linija i tekst se unosi u nju |

Modovi za unos teksta nisu isključivi jedan u odnosu na drugi – npr. iz insert moda se mogu emulirati svi ostali modovi, prethodnim postavljanjem na odgovarajuću poziciju u tekstu.

Povratak u komandni mod se obavlja pritiskom na taster ESC ili CTRL-[,

Pomoć u vim editoru

Za razliku od vi editora, vim editor ima prilično dobro napisanu on-line dokumentaciju kojom se pristupa iz ex-moda, komandom:


```
:help [tema]
```

Ovo će otvoriti zaseban prozor sa dokumentacijom. Kretanje kroz dokumentaciju se obavlja tako što se kursor postavi na početak označenog teksta i pritisne CTRL-]. Povratak nazad sa teme se obavlja pritiskom na CTRL-T ili CTRL-O. Izlaz i zatvaranje help prozora se obavlja iz ex-moda:

```
:q!
```

Snimanje, menjanje fajla i izlazak iz vim editora

Kada otvorite neki fajl, njegova kopija se smešta u jedan od bafera editora. Sve izmene u tekstu se obavljaju sa kopijom koja se nalazi u baferu. Izmena originalnog fajla će se obaviti tek kada to eksplicitno zadate tako što ćete snimiti sadržaj bafera u fajl.

Snimanje sadržaja bafera u fajl se obavlja ex-mod komandom:

```
: [opseg] w [novo_ime_fajla]
```

Samo:

```
:w
```

će snimiti ceo bafer u postojeći fajl.

Ako, umesto toga želite da snimate sadržaj bafera u novi fajl, iza 'w' unesite ime novog fajla.

Moguće je snimiti samo deo bafera, zadavanjem opsega.

Ukoliko želite da poništite izmene koje ste uneli u bafer i da bafer zatvorite bez snimanja fajla, koristite komandu:

```
:q
```

Ako je bafer menjan od poslednjeg snimanja fajla, editor neće dopustiti da se bafer zatvori već će vas opomenuti da u baferu postoje izmene koje nisu snimljene. Da ipak zatvorite bafer bez izmena, koristite modifikator komande '!', kao:

```
:q!
```

Istovremeno snimanje bafera i njegovo zatvaranje se obavlja spajanjem 'w' i 'q' komandi u komandni niz:

```
:wq
```

Napomena

Redosled ex-mod komandi u komandnom nizu je bitan. Npr. 'qw' neće snimiti izmene jer je q komanda ispred w komande.

Umesto ove komande ex-moda, možete iz komandnog moda otkucati 'ZZ' (dva velika slova Z!).

Ukoliko je bafer koji zatvarate poslednji bafer sa tekstom, automatski izlazite iz vim editora.

Kretanje kroz tekst u komandnom modu

U komandnom modu, vi editor koristi sledeće tastere za kretanje kroz tekst

| Unos | Akcija |
|----------------|--|
| h | kretanje za jedan znak ulevo |
| l | kretanje za jedan znak udesno |
| j | kretanje za jedan red nadole |
| k | kretanje za jedan red na gore |
| w | pozicioniranje na početak sledeće reči (znaci interpunkcije se tretiraju kao zasebne reči) |
| W | pozicioniranje na početak sledeće reči (znaci interpunkcije se tretiraju kao deo reči) |
| e | pozicioniranje na kraj sledeće reči (znaci interpunkcije se tretiraju kao zasebne reči) |
| E | pozicioniranje na kraj sledeće reči (znaci interpunkcije se tretiraju kao deo reči) |
| b | pozicioniranje na početak prethodne reči (znaci interpunkcije se tretiraju kao zasebne reči) |
| B | pozicioniranje na početak prethodne reči (znaci interpunkcije se tretiraju kao deo reči) |
| ^ ili 0 | pozicioniranje na početak tekućeg reda |
| \$ | pozicioniranje na kraj tekućeg reda |
| CTRL-F | kretanje unapred za jedan ekran |
| CTRL-B | kretanje unazad za jedan ekran |
| CTRL-U | kretanje unapred za pola ekrana |

| | |
|---------------|--------------------------------|
| CTRL-D | kretanje unazad za pola ekrana |
|---------------|--------------------------------|

Svaka od ovih komandi može imati brožani prefiks koji kazuje koliko puta se ponavlja komanda. Tako, npr. ako otkucate '5j', spustićete se 5 linija na dole (ako ispod tekuće linije ima toliko teksta).

Vim editor dozvoljava korišćenje i kurzorskih tastera, tastera PgUp i PgDn, ako i Home i End tastera i u komandnom i u modovima za unos teksta, za razliku od vi editora.

Višestruki unos

Ulazak u neki od modova za unošenje teksta, sve do povratka u komandni mod predstavlja jedan unos teksta. Ovaj unos može da se ponovi ako se komanda za ulaz u mod za unos teksta prefiskuje numeričkim prefiskom. U tom slučaju, unos će biti ponovljen onoliko puta koliki je numerički prefiks minus 1, zbog toga što ste već uneli tekst jednom. Ponavljanje će se desiti po povratku u komandni mod.

Na primer, ako ste u komandnom modu i unesete sledeći niz znakova (<ENTER> označava pritisak na taster RETURN ili ENTER; <ESC> označava pritisak na staser ESC ili CTRL-[]):

```
5iOvo je red.<ENTER><ESC>
```

dobićete:

```
Ovo je red.
Ovo je red.
Ovo je red.
Ovo je red.
Ovo je red.
```

Brisanje teksta

U modu za unos teksta, taster BACKSPACE briše znak iza kursora. U vim editoru, taster DELETE takođe briše znak ispred kursora, kada ste u modu za unos teksta.

U komandnom modu, brisanje znaka ispod kursora se ostvaruje unosom malog slova 'x' (u doba pisaćih mašina, jedini način da 'obrišete' neki tekst je da ga prekucate sa znacima 'x', odatle komanda).

Brisanje većih delova teksta obavlja se komandom d, koja obavezno mora da ima sufiks koji označava celinu koja se briše. Tako dobijamo niz sledećih komandi:

| Unos | Akcija |
|-----------|--|
| dw | brisanje teksta od kursora do početka sledeće reči |
| de | brisanje od kursora do kraja tekuće reči |
| db | brisanje od početka prethodne reči do kursora |

| | |
|-----------------|-------------------------------------|
| d\$ | brisanje od kursora do kraja reda |
| d0 | brisanje od početka reda do kursora |
| dd ili D | brisanje celog reda |

Naravno, sve ove komande mogu imati numerički prefiks koji ponavlja zadatu akciju.

Zamena teksta

Komande za zamenu teksta su identične komandama za brisanje dela teksta, s tom razlikom što se kod komandi za zamenu teksta odmah prebacujete u mod za unos teksta, dok kod komandi za brisanje teksta ostajete u komandnom modu.

Komande za zamenu teksta počinju sa 'c' (od *change*, promena):

| Unos | Akcija |
|------------|--|
| cw | menja od kursora do početka sledeće reči |
| ce | menja od kursora do kraja tekuće reči |
| cb | menja od početka prethodne reči do kursora |
| c\$ | menja od kursora do kraja reda |
| c0 | menja od početka reda do kursora |

Kao i uvek, ove komande mogu imati numerički prefix, koji se u ovom slučaju odnosi na količinu teksta koja će biti obrisana, a ne na unos teksta.

Selektovanje, kopiranje i lepljenje teksta

Selektovanje teksta u standardnom vi editoru se obavlja komandom 'y' (engl. *yank*, istrgnuti) koja, naravno, ima sufikse koji definišu opseg selektovanja:

| Unos | Akcija |
|------------|--|
| yw | selektovanje teksta od kursora do početka sledeće reči |
| ye | selektovanje od kursora do kraja tekuće reči |
| yb | selektovanje od početka prethodne reči do kursora |
| y\$ | selektovanje od kursora do kraja reda |

| | |
|-----------------|---|
| y0 | selektovanje od početka reda do kursora |
| yy ili Y | selektovanje celog reda |

Prilikom ove vrste selektovanja teksta, selektovani deo se ističe na ekranu, tj. ne vidi se da li je tekst selektovan i koji deo teksta je selektovan. Ove komande, normalno, prihvataju numeričke prefike.

Vim editor omogućava vizuelnu selekciju, koja se dobija komandama, 'v', 'V' ili CTRL-V. Prilikom ove vrste selekcije, selektovani tekst je naglašen, obično postavljanjem kontrastne boje pozadine selektovanih znakova. Selektovanje teksta znak po znak se vrši komandom 'v', gde se posle tekst selektuje pomeranjem kursora. Komanda 'V' služi za selektovanje celih linija, gde se linije odabiraju pomeranjem kursora. Komanda CTRL-V služi za pravougaono selektovanje teksta, gde je to moguće, gde se tekst selektuje po kolonama – pokretanjem kursora se selektuje širina i visina pravougaonika koji se selektuje. Selekcija može poslužiti u dve svrhe:

- sa 'y' ćemo prekopirati selektovani tekst u yank bafer
- sa 'd' ćemo odseći selektovani tekst i smestiti ga u yank bafer
- ako odmah po selektovanju teksta pređemo u ex-mod, vim će automatski dopisati opseg koji definiše selektovani deo teksta kako bismo nad tom selekcijom mogli da zadamo neku od ex komandi.

Lepljenje teksta se obavlja komandama 'p' i 'P' (od paste, zalepiti). Komanda 'p' lepi tekst počevši od pozicije iza kursora, dok 'P' lepi od kursora.

Napomena

Svaka komanda za brisanje teksta takođe automatski kopira obrisani sadržaj u yank bafer.

Vi nudi više bafera za smeštanje obrisano, tj. kopiranog teksta. To su tzv. imenovani yank baferi i pre nego što zadamo neku od komandi koja kopira tekst u yank bafer, moramo zadati komandu koja odabira novi imenovani bafer. Takoće, pre nego što zadamo komandu za lepljenje, moramo zadati komandu za odabir odgovarajućeg bafera. Imenovani baferi su označeni sa malim slovima a-z, a komanda za odabir bafera je:

```
"ime_bafera
```

Na ovaj način možemo imati do 24 teksta koji su spremni za nalepljivanje na željenu lokaciju.

Pretraživanje i zamenjivanje teksta

Pretraživanje teksta se zadaje iz komandnog moda, komandama '/' i '?'. Komanda '/' pretražuje tekst od kursora do kraja teksta. Komanda '?' pretražuje tekst od kursora unazad ka početku teksta. Kao argument se zadaje regularni izraz za pretragu. Za razliku od drugih komandi iz

komandne linije, ova komanda je vidljiva u dnu ekrana dok se unosi. Kraj unosa se označava sa tasterom RETURN ili ENTER. Ukoliko je pronađen tekst koji odgovara zadatom regularnom izrazu, kursor se pozicionira na početak tog teksta. Skok na sledeće pojavljivanje teksta koji odgovara zadatom regularnom izrazu obavlja se sa 'n' (engl. *next*, sledeći). Ukoliko pretraga dođe do granice teksta (početak ili kraj), pretraga se nastavlja od druge granice ka kursoru, uz vizuelnu napomenu.

Zamenjivanje teksta drugim se zadaje kroz komandu ex-moda 's' (engl. *substitute*, zameni):

```
: [opseg] s/trazeni_string/novi_string/[opcije]
```

trazeni_string i novi_string su regularni izrazi u Vim notaciji. Najčešće korišćene opcije su:

- g – (global) traži regularni izraz i dalje u liniji u kojoj si već našao jedan izraz
- i – (ignore case) ignoriši mala i velika slova

Ostale često korišćene komande

Unos fajla

Komandom 'ex-moda 'r' možemo učitati fajl koji će biti umetnut u tekući bafer od pozicije kursora:

```
:r ime_fajla
```

Izvršavanje eksterne komande nad delom teksta

Jedna od moćnih strana vim editora je što omogućava da izvršite neku od eksternih komandi za obradu teksta nad selektovanim delom bafera. Ovaj selektovani deo se eksternoj komandi isporučuje na standardni ulaz. Konačni standardni izlaz izvršene komande zamenjuje selektovani tekst u tekućem baferu. Komanda je:

```
!:eksterna_komanda
```

Na primer, da biste sortirali tekst u baferu i izbacili duplikate linija, zadajte:

```
:% !sort | uniq
```

Spajanje dve linije u jednu

U komandnom modu, zadavanjem komande 'J' se spaja tekuća linija sa linijom ispod. Znak za novi red u tekućoj liniji se menja u blanko i dopisuje se tekst sledeće linije, koja se onda briše.

Promena kapitalizacije slova

Ukoliko vam je potrebno da promenite malo slovo ispod kursora u veliko (ili obratno), u komandnom modu zadajte komandu '~' (tilda).

Ponavljanje prethodne komande

Svaku komandu u komandnom modu možemo ponoviti jednostavnim unosom znaka '!'.

Poništavanje prethodne komande

Komanda 'u' u komandnom modu poništava akciju prethodne komande i vraća tekst u oblik pre zadatavnja poništene komande.

Pitanja i zadaci

1. Iskopirajte neki tekstualni fajl (npr. /etc/passwd) i probajte sve komande koje su navedene u ovom skriptu.
2. Kreirajte sopstvene fajlove koristeći komande iz ovog dokumenta.

Odgovori na neka pitanja i rešenja nekih zadataka

Upoznavanje sa filozofijom open-source pokreta

1. Linux je dobio ime po svom tvorcu, Linusu Torvaldsu.
2. Ne. Štaviše, open-source licence ne zahtevaju da izvorni kod bude dostupan svima, već samo onima koji imaju binarnu verziju softvera.

Upoznavanje sa osnovama na kojima je zasnovan Linux

1. Linux je baziran na tri osnovna elementa: fajlovima, procesima i softverskim alatima
2. Tipovi fajlova koje Linux prepoznaje su:
 - regularni fajlovi
 - direktorijumi
 - simbolički linkovi
 - blok specijalni fajlovi
 - karakter specijalni fajlovi
 - imenovani pajpovi
 - soketi
3. i-nod je struktura koja sadrži metapodatke o fajlu i jednoznačno ga određuje. Između ostalog, i-nod za jedan fajl sadrži:
 - tip fajla
 - prava pristupa nad fajlom
 - vlasništvo nad fajlom
 - vremena kreiranja, poslednjeg pristupa i poslednje izmene sadržaja fajla
 - broj referenci na fajlu (biće objašnjeno u sledećem poglavlju)
 - ostali podaci koji zavise od tipa fajla (npr. kod regularnih fajlova i direktorijuma, uređaj na kojem se podaci nalaze i lokacija podataka unutar uređaja; kod specijalnih fajlova, indeks

koji određuje kojim drajverom se pristupa uređaju i dodatni podaci specifični za dati drajver).

4. Nema razlike, sva imena su ravnopravna, tj. sva imena su hard linkovi.
5. Ne, morate mu saopštiti u odnosu na koji direktorijum zadete relativnu putanju.
6. Osnovna prava pristupa nad regularnim fajlovima su:
 - read pravo, pravo iščitavanja sadržaja fajla
 - write pravo, pravo izmene sadržaja fajla
 - execute pravo, mogućnost izvršavanja fajla kao programa/komande
7. x pravo na direktorijumu označava da korisnik može da se pozicionira ili da 'prođe' kroz direktorijum. Drugim rečima, da bi neki korisnik mogao da pristupi nekom fajlu, mora da ima minimum x-pravo na svim direktorijumima u putanji tog faja.
8. Ne postoje slovne oznake za diskove na Linuxu.
9. Normalno, .. pokazuje na nadređeni direktorijum. Pošto root direktorijum (/) nema nadređeni direktorijum, .. pokazuje na /.
10. Svaki proces ima tačno jednog roditelja.
11. Svaka komanda vraća izlazni status procesu roditelju. Pošto je komandni interpreter roditelj svim komandama koje izvršavamo on dobija izlazni status komande.
12. Fajlovi ne mogu direktno da komuniciraju međusobno, već isključivo preko nekog procesa.

Pristup Linux-u, terminali i SSH

1. SSH omogućava autentifikaciju preko lozinke, ali postoje i sigurniji načini autentifikacije preko SSH protokola, kao što je autentifikacija preko javnih i tajnih ključeva.
2. Lozinka je informacija koju korisnik ne bi trealo da obznanjuje drugima.

Rad u komandnoj liniji

1. Fajl globing je način na koji se može jednostavno zadati veći set fajlova-argumenata.
2. Znak '?' označava tačno jedan znak kod wildcard izraza.
3. Odgovor:

```
*.[^]*
```
4. help komanda
5. U sekciji 5 se nalaze opisi konfiguracionih fajlova i fomrata

6. Komadnom 'history' ili strelicom na gore.

Osnovne komande za rad sa fajlovima

1. `ls -l`
2. Prvi znak redu je znak '-l'.
3. Svaki direktorijum ima više imena. Npr. za direktorijum 'prvi' koji smo kreirali u našem početnom direktorijumu imena su:

- `~/prvi`
- `~/prvi/`.

Ako direktorijum ima n poddirektorijuma onda je broj referenci $2 + n$, jer u svakom od poddirektorijuma postoji ime `..` koje pokazuje na nadređeni direktorijum.

4. Poslednji argument u tom slučaju mora biti neki direktorijum koji postoji.
5. Zato što je prebacivanje fajlova (ako se radi o istoj particiji) ista operacija kao i preimenovanje fajlova: prvo se obriše postojeće ime a zatim se doda novo (razlika je samo što se kod preimenovanja novo ime dodaje u isti direktorijum gde je bilo i staro ime). Zbog kompatibilnosti, komanda `mv` se koristi i za prebacivanje fajlova koji su na različitim particijama (misli se na izvoriste i destinaciju), iako ta operacija zahteva fizičko kopiranje sadržaja.
6. Da, komandom `'rm -r'`.
7. Veličina linka u bajtovima odgovara broju znakova u referenci linka, bez znaka za novi red.
8. Komanda i dalje vraća isti rezultat.
9. `find /usr -type d -name 'l*'`
10. `find /usr/share -size +1M`

Rad sa direktorijumima

1. `cd ; cd ~`
2. `mkdir -p {A,B}/{prvi,drugi}`
3. Ne, komanda `'rmdir'` zahteva da direktorijum koji se briše bude prazan.

Regularni izrazi

1. Kod osnovnih POSIX regularnih izraza, znak `'\'` je literal.
2. Kod osnovnih POSIX regularnih izraza, znak `'*'` je metaznak, a `'+'` je literal.

3. Kod proširenih POSIX regularnih izraza, oba znaka su metaznaci, a razlika je u tome što '+' zahteva da prethodni znak bude ponavljen makar jednom.

4. Možemo identifikovati tri grupe znakova u izrazu za broj telefona, koje ćemo označiti sa a, b i c:

`^+381aabbcccc$` ili `^+381aabbcccc$`. Drugi izraz treba da ima oblik `(0aa) bbb-ccc` ili `(0aa) bbb-cccc`.

Pošto su aa dve cifre možemo pisati `[0-9][0-9]` ili `[0-9]\{2,2\}`

Pošto su bbb tri cifre možemo pisati `[0-9][0-9][0-9]` ili `[0-9]\{3,3\}`

Pošto su ccc i cccc tri ili četiri cifre imamo tri fiksna znaka i jedan koji može da postoji ili ne: `[0-9][0-9][0-9][0-9]?` ili `[0-9]\{3,4\}` (? označava da se prethodni znak ponavlja 0 ili jedan put).

Rezultujući izraz je:

```
^+381\ ([0-9][0-9])\ ([0-9]\{3,3\})\ ([0-9]\{3,4\})$
```

Drugi izraz je:

```
(0\1) \2-\3
```

(koristimo reference \1 za aa, \2 za bbb i \3 za ccc(c)).

5. Osnovni POSIX regularni izrazi:

Prvi izraz je:

```
^+381\ ([0-9][0-9])\ ([0-9]\{3,3\})\ ([0-9]\{3,4\})$
```

Drugi izraz je:

```
(0\1) \2-\3
```

Prošireni POSIX regularni izrazi:

Prvi izraz je:

```
^\+([0-9][0-9])([0-9]\{3,3\})([0-9]\{3,4\})$
```

Drugi izraz je:

```
(0\1) \2-\3
```

Vim regularni izrazi:

Prvi izraz je:

```
^\+(\d\d)(\d{3,3})(\d{3,4})$
```

Drugi izraz je:

```
(0\1) \2-\3
```

Komande za rad sa sadržajem fajlova

1.

```
$ cat - > zad1.txt  
Ja sam ponosni polaznik kursa  
Linux L1-1 u ATC-u<CTRL-D>
```

2. `head -n 3 /etc/passwd`

3. `sort -rn -t: -k3 /etc/group`

4. `grep /bin/bash /etc/passwd`

5. `grep -v /bin/bash /etc/passwd`

12. `cat /etc/passwd | tr -d '0-9'`

Redirekcija I/O i pajpovi

1. `tail -n 3 /etc/passwd | head -n 1`

2. `ls -l /bin | wc -l`

Prava pristupa – detaljnije

1. 04711:

- korisnik ima sva prava
- grupa i ostali imaju pravo izvršavanja
- postavljen je setUID bit

2. umask 066

Rad sa procesima

1. `ps -edaf`



Admin Training
Center

VESELIN MIJUSKOVIĆ

Linux L1-2

Bash shell scripting

Admin Training Center
Studentski trg 4, VI sprat
11000 Beograd, Srbija
<http://www.atc.rs/>



Copyright ©2013 Veselin Mijušković, Ljubiša Radivojević, Marko Uskoković

Ovaj tekst je licenciran pod Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License. Da biste videli kopiju ove licence posetite:
http://creativecommons.org/licenses/by-nc-nd/3.0/deed.en_US

Sadržaj

| | |
|--|-----------|
| Uvod - tipografske konvencije | 3 |
| Tipografske konvencije | 3 |
| Šta je skript? | 5 |
| Format skripta | 5 |
| 'hashbang' putanja | 6 |
| Kako bash izvršava komandnu liniju? | 7 |
| Redosled transformacija | 7 |
| Podela komandne linije na reči | 7 |
| Navođenja | 9 |
| Ekspanzija vitičastih zagrada | 9 |
| Ekspanzija znaka '~' | 10 |
| Ekspanzija parametara | 10 |
| Zamena komandi | 11 |
| Aritmetička ekspanzija | 11 |
| Podela na reči | 12 |
| Ekspanzija putanja | 12 |
| Parametri i promenljive | 13 |
| Različiti načini referenciranja parametara | 14 |
| Ispis podrazumevane vrednosti | 15 |
| Ispis podstringa | 16 |
| Uklanjanje prefiksa i sufiksa | 16 |
| Zamena unutar stringa | 17 |
| Promenljive okruženja | 17 |
| Promenljive okruženja koje postavlja komandni interpreter | 18 |
| Nizovi | 20 |
| Indeksirani nizovi | 20 |
| Asocijativni nizovi | 21 |
| Kontrola toka programa | 22 |
| Kontrola toka: if | 22 |
| Petlje: while i for | 23 |
| Grananje: case | 25 |
| Funkcije | 27 |
| Prenošenje argumenata u shell skript | 29 |
| Uslovni izrazi | 30 |
| Subshellovi i izvršavanje komandi iz drugog skripta u istom procesu | 33 |
| Rad sa opcijama | 34 |
| Aritmetika | 36 |
| Napredan rad sa terminalima | 38 |

Uvod - tipografske konvencije

Poštovani polaznici, pred vama je prateća skripta za ATC kurs Linux L1-2: Shell scripting. Ova skripta je namenjena kao dodatak predavanjima i kao podsetnik za sintaksu shell skriptova. Ona će biti u 'perpetualnom razvoju' tako da je svaka povratna informacija od strane vas (ispravke grešaka, upozorenja da nešto nije dovoljno detaljno ili dovoljno jasno objašnjeno i sl.) poželjna.

Skripta je podeljena na poglavlja, a ona na sekcije. Unapred skrećemo pažnju polaznicima da je ova skripta samo deo dokumentacije koju oni treba da koriste. Polaznicima se savetuje da pročitaju man i help strane za svaku komandu, kao i da potraže na Internetu dodatne informacije i načine kako da iste korsite.

Tipografske konvencije

Radi lakšeg snalaženja u tekstu, koristili smo neke tipografske konvencije na koje vam ovde skrećemo pažnju:

- ukoliko se uvodi neki značajan pojam, on će u prvom pomenu biti ispisan **proporcionalnim bold** tekstom;
- boldovano su prikazane i neke **značajne tvrdnje** na koje treba obratiti pažnju u tekstu;
- *proporcionalnim italikom* su napisane reči na stranom jeziku, najčešće engleskom;
- nazivi programa, opcija i fajlova u tekstu su ispisani neproporcionalnim fontom

U ovom dokumentu ćete naći dosta primera celih shell skripti, kao i delova koda. Taj ispis je prikazan u zasebnom bloku, kao što je sledeći primer:

```
#!/bin/bash
#
# args.sh - primer prenosjenja argumenata u skript

$prog=`basename $0`

if [[ $# -ne 2 ]]; then
    echo "Upotreba: $prog fajl.in fajl.out"
    exit 1
fi
```

Primetićete da su ključne reči ispisane **boldovano**.

Pored standardnog teksta ova skripta sadrži i neke primere pozivanja komandi i prikaz rezultata tih komandi. Takav ispis je prikazan u zasebnom bloku, kao što je ovaj:

```
$ ls -F  
myscript.sh*  Vezbe/
```

Boldovanim tekstom je prikazano ono što polaznik treba da unese onako kako je napisano u skripti. Regularnim tekstom je prikazan ispis programa koji ne treba unositi.

Isti metod se koristi i za prikaz formata komandi i njihovih parametara:

```
$ cp [opcije] source... dest
```

U ovom slučaju je boldovano prikazana sama komanda i tako je treba uneti. Italikom su prikazani promenljivi parametri i to znači da na tom mestu treba uneti neku stvarnu vrednost, a ne ono što piše u komandnoj liniji. Na kraju, opcioní parametri, koji se mogu izostaviti su navedeni u srednjim zagradama []. Naravno, zagrade ovde služe samo kao naznaka i ne unose se! Tri tačke (...) označavaju da je dati parametar moguće navesti više puta.

Gde god smo mislili da nešto posebno treba naglasiti, to smo naveli u zasebnom boksu, koi obično ima naslov 'Važno!' ili 'Napomena', kao ovde:

Važno!

Hard linkovi ne mogu biti kreirani za direktorijume, već samo za regularne fajlove!

Šta je skript?

Najveći deo posla jednog Linux administratora se sastoji u regularnom ponavljanju istog niza komandi. Još češće, ovi nizovi se razlikuju samo u pojedinim detaljima. Još češće, administrator mora da odluči na osnovu par parametara da li će da izvrši jedan niz komandi ili drugi. Kako bi dobro bilo kada bi se ovakvi slučajevi automatizovali i kada bi administrator mogao samo da odabere koji niz komandi treba da izvrši, bez da poziva jednu po jednu komandu svaki put.

Srećom, gore navedena automatizacija je moguća - štaviše, ona je česta i spada u jednu od bitnih osobina posla Linux administratora. Ova automatizacija je moguća kroz kreiranje skripti koje sadrže navedene nizove komandi.

Dakle, skripte su obični tekstualni fajlovi koji sadrže neki niz komandi koje normalno izvršavamo u komandnom interpreteru. Kada su ove komande smeštene u jedan tekstualni fajl, onda taj fajl možemo proslediti komandnom interpreteru na izvršavanje, bilo da se te komande izvršavaju unutar tekućeg procesa komandnog interpretera, bilo da se pokreće zaseban podproces za izvršenje ovih komandi. Još bolje, ovaj tekstualni fajl može postati nova komanda unutar našeg okruženja, ukoliko se vodi računa o odgovarajućem formatu ovog fajla i ukoliko mu se dodeli pravo izvršavanja. Takav fajl, posebno ako je smešten u neki od direktorijuma koje komandni interpreter podrazumevano pretražuje kada traži zadatu komandu, postaje punovažna komanda koja se ne razlikuje od ostalih na sistemu (od kojih je jedan deo njih takođe u obliku komandnih skriptova).

Format skripta

Da bi skript mogao da se izvršava mora da poštuje dva jednostavna pravila koja definišu njegov format:

- komande unutar skripta moraju biti napisane na isti način kao i pri interaktivnom radu,
- skript može sadržati prazne linije (linije koje se sastoje samo od belina), kao i komentare koji počinju znakom '#' i završavaju se znakom za kraj reda (newline). Ovi komentari se zanemaruju prilikom izvršavanja komandi.

Kao što se iz prethodnog pasusa može zaključiti, komandni interpreter parsira skript liniju po liniju i tako i izvršava komande na isti način kao kad bi one bile zadate preko standardnog ulaza. **Sve** komande koje su dostupne u interaktivnom modu su dostupne i za skriptove i obratno.

Skript se najčešće izvršava u zasebnom procesu, kao i bilo koja druga komanda. Najjednostavniji način zadavanja izvršavanja nekog skripta je:

```
$ bash < ./skript.sh
```

Naravno, možemo i bez redirekcije - bash prima ime skripta kao argument:

```
$ bash ./skript.sh
```

'hashbang' putanja

Da bi se skript mogao pokretati kao zasebna komanda, pored toga što korisnik treba da ima pravo izvršavanja skripta, skript mora zadovoljiti još jedno pravilo. Naime, __prva__ linija skripta mora početi sa:

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

Ova linija se naziva 'hashbang' putanja jer prvi znak u prvom redu mora biti znak '#' (engl. *hash sign*), a drugi znak '!' (engl. (slang) *bang*). Ova dva znaka **moraju** biti dva prva znaka u fajlu. Sve iza znaka '!' pa do kraja linije se uzima kao komanda kojoj se na standardni ulaz prosleđuje ceo skript, uključujući i ovu prvu liniju. U našem slučaju, ostatak linije je apsolutna putanja do bash komandnog interpretera (/bin/bash). Navedena komanda sama ne sme biti skript, a morate navesti punu putanju do nje. Pošto je prvi znak u liniji znak '#', cela linija se od strane komandnog interpretera tretira kao komentar i zanemaruje se.

Napomena

Skriptovi ne moraju biti pisani samo za izvršavanje u komandnom interpreteru. Navedeni format skript fajla dozvoljava da bilo koji program koji može da čita svoje komande sa standardnog ulaza može biti korišćen kao interpreter komandi u skriptu. Na taj način, npr. pišu se Perl ili Python programi, gde prva linija posle 'hashbang' znakova sadrži apsolutnu putanju do Perl ili Python interpretera, a ostatak teksta su Perl ili Python komande. Jedino ograničenje je da ovi programski jezici ne tumače prvu liniju (hashbang liniju) kao komandu što je rešeno tako da i u ovim jezicima linijski komentari počinju znakom '#!'

Kako bash izvršava komandnu liniju?

Možda će vas iznenaditi činjenica da komandna linija, koju ste upravo ukucali u interaktivnom modu ili koja je napisana u skriptu, prolazi kroz nekoliko transformacija pre nego što dobije svoj konačni oblik koji se šalje na izvršavanje. Sve ove transformacije odrađuje bash komandni interpreter. Poznavanje ovih transformacija, kao i kojim redosledom se izvršavaju je od velikog značaja pri radu sa komandnim interpreterom, posebno kada pravimo skriptove.

Redosled transformacija

Kada komandna linija uneta (bilo interaktivno bilo da je pročitana iz skripta), bash nad njom primenjuje sledeće transformacije:

- podela komandne linije na reči
- ekspanzija vitičastih zagrada (engl. brace expansion)
- ekspanzija znaka '~' (engl. tilde expansion)
- ekspanzija parametara i varijabli (engl. parameter and variable expansion)
- zamena komandi (engl. command substitution)
- aritmetička ekspanzija (engl. arithmetic expansion)
- podela reči (engl. word splitting)
- ekspanzija putanja (engl. pathname expansion)

Sve ekspanzije se obavljaju redosledom sa leva u desno.

Tek kad završi sve ove transformacije, tako izmenjena komandna linija se prosleđuje na izvršavanje.

Na primeru sledeće komandne linije ćemo prikazati sve transformacije:

```
$ cp ~pera/{dev,prod}/*.py \  
$HOME/`date +%Y`/`date +%m`/$((`date %d`-1))/"pera backup"/
```

Podela komandne linije na reči

Uneta komandna linija se tokenizuje, tj. izdvajaju se tokeni po zadatom redosledu. Token je niz znakova koji predstavlja pojedinačni entitet. Za token se često koristi sinonim 'reč'. Token, odnosno reč koja se sastoji samo od alfanumeričkih znakova i znaka '_', a koja počinje slovnim znakom ili znakom '_' se naziva 'ime' ili 'identifikator'.

U komandnoj liniji, koja je običan niz znakova, tokeni su razdvojeni tzv. metaznacima (ovi metaznaci nisu isto što i metaznaci kod regularnih izraza!):

- |
- &
- ;
- (
-)
- <
- >
- belina (razmaknica ili tab)

Pojedini tokeni su tzv. kontrolni operatori:

- ||
- &
- &&
- ;
- ;;
- (
-)
- |
- |&
- znak za novi red (newline)

Da bi poništili značenje pojedinih znakova (većinom metaznakova) ili specifičnih reči koristimo navođenja.

U našem primeru komandna linija je podeljena ovako:

```
$ cp ~pera/{dev,prod}/*.py \
  $HOME/`date +%Y`/`date +%m`/`${date +%d}-1`/"pera backup"/
`cp'
`~pera/{dev,prod}/*.py'
`$HOME/`date +%Y`/`date +%m`/`${date +%d}-1`/"pera backup"/'
```

Navođenja

Bash poznaje tri vrste navođenja:

- navođenje preko escape znaka
- navođenje preko jednostrukih znakova navoda
- navođenje preko dvostrukih znakova navoda

Escape znak je '\'. Ako stoji neposredno ispred nekog metaznaka onda poništava njegovo posebno značenje i vraća mu obično leksičko značenje. Ako stoji neposredno ispred nekog od slovnih znakova, menja značenje tog znaka u specijalno. Neke od tih znakova ćemo pominjati u narednim sekcijama.

Kod navođenja preko jednostrukih znakova navoda, svaki znak unutar ovih znakova navoda ima svoje literalno značenje, bez izuzetka.

Kod navođenja preko dvostrukih znakova navoda, svaki znak unutar ovih znakova navoda ima svoje literalno značenje, sa izuzetkom znakova '\$', '"' i '\' koji i dalje zadržavaju svoje specijalno značenje.

Ekspanzija vitičastih zagrada

Vitičaste zagrade su mehanizam kojim se mogu generisati proizvoljni stringovi unutar komandne linije. Stringovi koji se ekspanduju imaju format:

```
[preamble]{str1,str2,...}[postsript]
```

ili

```
[preamble]{poc..kraj[,inkrement]}[postsript]
```

gde su 'preamble' i 'postsript' opcionni nizovi znakova. Ako je string koji se ekspanduje zadat u prvom formatu, onda će se na njegovo mesto ubaciti niz stringova:

```
preamblestr1postsript preamblestr2postsript ...
```

Ako je string zadat u drugom formatu, onda se ekspanzija odigrava isto kao i u prethodnom slučaju, sem što se za promenljivi deo uzimaju sve vrednosti počevši od 'poc', zaključno sa 'kraj', eventualno u koracima 'inkrement'. 'poc' i 'kraj' moraju biti ili celi brojevi ili pojedini znakovi. 'inkrement' uvek mora biti ceo broj, a ako se izostavi podrazumeva se vrednost 1 ili -1 (u zavisnosti da li je 'poc' u numeričkom ili leksikografskom nizu iza ili ispred 'kraj').

U našem primeru:


```
'~pera/{dev,prod}/*.py'
```

je jedini token kod kojeg treba izvršiti ekspanziju vitičastih zagrada. U ovom slučaju, preambula je '~pera/', a postskript je '/*.py', pa posle ekspanzije komanda izgleda ovako:

```
'cp'  
'~pera/dev/*.py'  
'~pera/prod/*.py'  
'$HOME/`date +%Y`/`date +%m`/$(($`date +%d`-1`))/"pera backup"/'
```

Ekspanzija znaka '~'

Ako neka reč/token počinje nenavedenim znakom '~' onda se svi znaci iza ovog znaka, a do prvog znaka za razdvajanje direktorijuma (/) tretiraju kao tilda-prefiks. Ako je tilda-prefix prazan string, onda se znak '~' zamenjuje vrednošću parametra HOME ili, ako ovaj nije postavljen, putanjom početnog direktorijuma korisnika koji izvršava komandu. Ako pak tilda-prefiks nije prazan string, onda se podrazumeva da je u pitanju validan username na sistemu, i ceo niz ~user se zamenjuje sa putanjom početnog direktorijuma korisnika 'user'. Ako korisnik 'user' ne postoji, token ostaje nepromenjen.

U našem primeru početni direktorijum korisnika 'pera' je /home/pera, pa komandna linija postaje:

```
'cp'  
'/home/pera/dev/*.py'  
'/home/pera/prod/*.py'  
'$HOME/`date +%Y`/`date +%m`/$(($`date +%d`-1`))/"pera backup"/'
```

Ekspanzija parametara

Parametri su entiteti koji pohranjuju neku vrednost. Varijable takođe spadaju u grupu parametara. Referenciranje nekog parametra u komandnoj liniji se zadaje znakom '\$' koji je praćen imenom parametra. Obratite pažnju da znak '\$' može takođe označavati zamenu komandi ili aritmetički izraz. Ime parametra možemo opciono ograditi vitičastim zgradama kako bi eksplicitno naznačili ime parametra (inače, bash će za ime tretirati najduži mogući niz znakova iza znaka '\$' koji gramatički može da predstavlja ime nekog parametra). Bilo gde u komandnoj liniji gde referenciramo neki parametar, umesto te reference će biti ubačena vrednost koju taj parametar pohranjuje. Moguće je i zadati modifikovanje imena parametra ili njegove vrednosti, o čemu će više biti reči u sekciji koja se bavi varijablama.

U našem primeru imamo jedno referenciranje varijable HOME, pa ako pretpostavimo da je njena vrednost '/home/atc/', onda dobijamo:

```
'cp'  
'/home/pera/dev/*.py'  
'/home/pera/prod/*.py'  
'/home/atc/`date +%Y`/`date +%m`/$(($`date +%d`-1`))/"pera backup"/'
```

Zamena komandi

Zamena komandi je mogućnost da se ime komande zameni rezultatom izvršenja te komande. Ovo se može zadati na dva načina:

```
$ (komanda)
```

ili

```
`komanda`
```

Zamena se izvodi tako što se ove komande izvrše u zasebnom procesu i njihov standardni izlaz se upiše umesto celog izraza.

U našem primeru, komanda 'date' je navedena da se izvršava tri puta, sa različitim argumentima. Ova komanda vraća tekući datum, a argumenti koji su joj prosleđeni definišu format ispisa datuma, ako ispred njih stoji znak '+'. Format:

%Y – vraća godinu u obliku YYYY

%m – vraća mesec u obliku MM

%d – vraća dan u obliku DD

Ako je ova komanda izvršena 23. oktobra 2013, onda je rezultat:

```
date +%Y --> 2013
date +%m --> 10
date +%d --> 23
```

pa u zameni dobijamo:

```
`cp`
`/home/pera/dev/*.py`
`/home/pera/prod/*.py`
`/home/atc/2013/10/${(23-1)}/"pera backup"/`
```

Aritmetička ekspanzija

Aritmetička ekspanzija traga za tokenima oblika:

```
${(izraz)}
```

i ceo token menja rezultatom aritmetičkog izraza 'izraz'. O ovim izrazima će više biti reči u narednim poglavljima.

U našem primeru imamo jedan aritmetički izraz

```
${(23-1)} --> 22
```

pa zamenom dobijamo:

```
'cp'  
'/home/pera/dev/*.py'  
'/home/pera/prod/*.py'  
'/home/atc/2013/10/22/"pera backup"/'
```

Podela na reči

Nakon svih gore obavljenih ekspanzija i zamena, komandni interpreter ponovo deli komandnu liniju na tokene, koristeći znake definisane u varijabli IFS kao delimitere. Podrazumevana vrednost ove varijable je '<space>,<tab>,<newline>'. Ovi znaci se zanemaruju i komandna linija se deli u tokene. Podelu na reči možemo blokirati na delu linije ako taj deo ogradimo znacima navoda (jednostrukim ili dvostrukim).

U našem primeru se ništa ne menja jer je jedina belina unutar znakova navoda, pa se na tom mestu token ne prelama.

Ekspanzija putanja

Nakon svih ostalih ekspanzija vrši se ekspanzija putanja, gde komandni interpreter traga za pojavama znakova '*', '?' ili '[' i formira fajl globing izraze koje zatim zamenjuje stvarnim imenima fajlova.

Ako pretpostavimo da korisnik pera u svojim '/home/pera/dev' i '/home/pera/prod' ima sledeće Python fajlove:

```
/home/pera/dev/class.py  
/home/pera/dev/test.py
```

i

```
/home/pera/prod/app.py
```

onda posle ekspanzije putanja dobijamo konačan oblik naše komande:

```
'cp'  
'/home/pera/dev/class.py'  
'/home/pera/dev/test.py'  
'/home/pera/prod/app.py'  
'/home/atc/2013/10/22/"pera backup"/'
```

Na kraju ove ekspanzije dobija se konačan izgled komandne linije. Ukoliko ne postoji redirekcija ulaza i/ili izlaza, komanda se izvršava. Ako pak redirekcija postoji, prvo se otvaraju ti fajlovi za čitanje i/ili pisanje i povezuju sa odgovarajućim fajl deskriptorima i tek se onda komanda izvršava.

Parametri i promenljive

Parametri su entiteti koji pohranjuju neki podatak, kao što smo rekli u prethodnom poglavlju. Svaki parametar ima neko ime preko kojeg ga razlikujemo od drugih parametara. Ako je to ime sastavljeno samo od alfanumeričkih znakova i znaka '_' i počinje slovom ili znakom '_', onda taj parametar nazivamo **promenljiva ili varijabla**. Ovo znači da postoje parametri koji nisu promenljive! No, skoro uvek i promenljive i ostale parametre tretiramo na isti način.

Svaki parametar ima:

- ime
- vrednost

Da bismo dobili vrednost parametra, moramo ga referencirati po imenu. To se radi tako što se ispred imena doda znak '\$'. Da bismo parametru dodelili vrednost, koristimo operator '=':

```
$ ime=vrednost
```

Vodite računa o sledećim stvarima:

- za dodelu vrednosti uvek se koristi ime, nikad referenca
- između imena i operatora '=' ne sme biti belina
- sve iza operatora '=' se uzima kao vrednost koja se dodeljuje - drugim
- rečima, i tu izbegavajte beline.

Kada prvi put dodeljujemo vrednost nekoj promenljivoj, kažemo da tu promenljivu inicijalizujemo. Dobra praksa je da uvek promenljive prvo inicijalizujemo, a tek potom referenciramo. Parametre koji nisu promenljive nikada ne inicijalizujemo sami - to radi sam komandni interpreter. No, moguće je i da neinicijalizovanu promenljivu referenciramo - u tom slučaju promenljiva ima tzv. null vrednost. Ako promenljivu referenciramo u tekstualnom kontekstu, null vrednost će se mapirati u prazan string "", a ako je referenciramo u numeričkom kontekstu, null vrednost će se mapirati u 0. Parametri koji nisu promenljive takođe mogu imati null vrednosti. Iz prethodnog se vidi da promenljive nisu tipizirane kao kod nekih drugih programskih jezika. Naime, jedna ista promenljiva u jednom trenutku može sadržati podatak koji je broj, a u drugom podatak koji je tekst. Jedan te isti podatak možemo tumačiti kao broj i kao tekst, što zavisi od konteksta u kojem referenciramo promenljivu. Ovo, naravno ima smisla samo ako je podatak oblika broja, jer se broj može tretirati i numerički (što i jeste) i kao string koji se sastoji od niza cifara. Primer:

```
#!/bin/bash
#
# Skript koji demonstrira inicijalizovanje, dodeljivanje vrednosti i
# referenciranje promenljivih
#

promenljiva1="Neka vrednost"      # inicijalizovanje promenljive
echo $promenljiva1                # referenciranje promenljive
echo promenljiva1                 # ovo nece prikazati vrednost
                                  # promenljive!

promenljiva1=10                   # nova vrednost je broj
echo $((promenljiva1+3))           # promenljiva1 se tretira kao broj
echo "$promenljiva1 kao tekst"     # promenljiva1 se tretira kao string

echo "Vrednost var2 = '$var2'"     # referenciranje neinicijalizovane
echo "var2 + 2 =" $((var2+2))      # promenljive kao stringa i kao broja
```

Rezultat izvršavanja ovog skripta je:

```
Neka vrednost
promenljiva1
13
10 kao tekst
Vrednost var2 = ''
var2 + 2 = 2
```

Različiti načini referenciranja parametara

Već smo videli da se parametar referencira tako što se ispred njegovog imena doda prefiks '\$'. U tom slučaju, u fazi ekspanzije parametara, umesto znaka '\$' i imena parametra se stavlja vrednost koju taj parametar u tom trenutku ima.

No, to je samo jedan od načina kako možemo referencirati parametar. Relativno često prilikom zadavanja komandi, bilo interaktivno, bilo kroz skript, dolazimo u situaciju da vrednost parametra mora biti praćena nekim tekstom, bez standardnog odvajanja belinama. Shell skriptovi nemaju mogućnost konkatencije stringova na neki specifičan način već je dovoljno nastaviti tekst odmah iza referenciranja parametra. No, tu dolazimo do problema - ako jednostavno iza imena promenljive dopišemo tekst koji treba da stoji uz vrednost promenljive, moguće je da dopisani tekst ili njegov početak, zajedno sa imenom parametra čine ispravno ime promenljive, bez obzira da li ista postoji. Kako smo pokazali u prethodnom poglavlju, komandni interpreter će podeliti komandnu liniju na stringove koji se završavaju belinama ili kontrolnim operatorima, a u našem slučaju to može da obuhvati ime promenljive i tekst dopisan iza. Komandni interpreter će takav tekst smatrati ispravnim imenom promenljive, i zameniće referenciranje te nove promenljive njenom vrednošću. Podsetite se da, ako promenljiva nije prethodno inicijalizovana, prilikom prvog referenciranja će joj biti dodeljena null vrednost, što se u kontekstu teksta pretvara u prazan string i onda ćemo, umesto vrednosti naše promenljive i dopisanog teksta, dobiti samo prazan string. Na primer:

```
$ prefiks="Hadzi-"
$ echo "$prefiksJovanovic"          # ovo NEĆE ISPISATI Hadzi-Jovanovic!
```

Inicijalizovali smo promenljivu 'prefiks' i dodelili joj vrednost 'Hadzi-'. Želimo da ispišemo ovaj prefiks pre prezimena 'Jovanovic' u ispisu, međutim, komandni interpreter će potražiti da li postoji promenljiva čije ime je 'prefixJovanovic'! Ako ova promenljiva ne postoji, biće inicijalizovana i dodeljena joj null-vrednost.

Kako rešiti ovaj problem? Jednostavno, u takvim slučajevima treba ime varijable, odnosno parametra pisati između zagrada '{' i '}':

```
$ echo "${prefiks}Jovanovic"
```

Ovo će signalizirati komandnom interpreteru da je ime varijable samo ono što je unutar vitičastih zagrada, a ostalo je običan tekst.

Pored ovog, postoji još nekoliko načina na koji referenciramo parametre, gde se vrednost varijable dodatno menja, u skladu sa zadatim pravilima.

Napomena

U navedenim načinima referenciranja parametra, parametar zadržava postojeću vrednost, izuzev ukoliko to eksplicitno nije navedeno!

Ispis podrazumevane vrednosti

Ako prilikom referenciranja neke promenljive ili parametra želimo da umesto null-vrednosti bude ispisana neka podrazumevana vrednost, onda koristimo oblik '\${parametar:-podrazumevana vrednost}', kao u primeru:

```
#!/bin/bash
#
# Skript koji demonstrira referenciranje parametra
# ispisom podrazumevane vrednosti
#

a="Ja imam neku vrednost"
echo $a          # ispisuje 'Ja imam neku vrednost'
echo ${a:-default} # ispisuje 'Ja imam neku vrednost'
a=""             # 'a' sada ima null-vrednost
echo ${a:-default} # ispisuje tekst 'default' jer 'a' ima null-vrednost
echo $a          # vrednost 'a' se nije promenila
```

Slično prethodnom, oblik '\${parametar:=podrazumevana vrednost}' će odraditi isto što i prethodni oblik, sa tom razlikom što će, ukoliko 'parametar' ima null-vrednost, 'parametar' dobiti vrednost 'podrazumevana vrednost'. Isti primer od pre će dati nešto drugačiji rezultat:

```
#!/bin/bash
#
# Skript koji demonstrira referenciranje parametra
# dodelom podrazumevane vrednosti
#

a="Ja imam neku vrednost"
echo $a                # ispisuje 'Ja imam neku vrednost'
echo ${a:=default}     # ispisuje 'Ja imam neku vrednost'
a=""                   # 'a' sada ima null-vrednost
echo ${a:=default}     # ispisuje tekst 'default' jer 'a' ima null-vrednost
echo $a                # vrednost 'a' je sad postala 'default'
```

Ispis podstringa

Oblik '\${parametar:ofset:duzina}' ili '\${parametar:ofset}' referenciraju vrednost parametra 'parametar' tako da prikažu samo 'duzina' znakova, počevši od 'ofset' znaka, gde indeks počinje sa 0. Ako je 'ofset' negativan broj, računa se od kraja stringa koji predstavlja vrednost parametra 'parametar'. Takođe, negativan 'ofset' mora da se piše sa jednim blankom ispred znaka '-'. Ako je, pak, 'duzina' negativan broj, onda se ona tretira takođe kao odmak, samo računajući od kraja vrednosti, pa se ispisuje podstring koji počinje 'ofset' znakova od početka a završava se 'duzina' znakova od kraja. Primeri:

```
#!/bin/bash
#
# Skript koji demonstrira referenciranje parametra
# ispisom podstringa
#

var1="Ja sam varijabla"
a=3
b=5
echo ${var1:a:b}        # ispisaće 'sam v'
echo ${var1:2:6}.       # ispisaće ': sam v.'
echo ${var1:-7:b}       # ispisaće 'rijab'
echo ${var1:-7:b}       # ispisaće 'Ja sam varijabla', jer
                        # ispred '-' ne stoji ' '
echo ${var1:2:-6}       # ispisaće 'sam var'
```

Uklanjanje prefiksa i sufiksa

Oblik '\${parametar#prefix}', odnosno '\${parametar##prefix}' uklanjaju tekst 'prefix' sa početka vrednosti parametra 'parametar'. 'prefix' se može zadati kao fajl globing. Razlika između dva oblika je što kod oblika '\${parametar#prefix}' uklanja se najmanja moguća ekspanđovana vrednost fajl globing izraza 'prefix', dok se kod oblika '\${parametar##prefix}' uklanja najveća moguća ekspanđovana vrednost fajl globing izraza 'prefix'. Ovo ćemo jasnije pokazati na primeru:

```
#!/bin/bash
#
# Skript koji demonstrira referenciranje parametra
# uklanjanjem prefiksa
#

tekst="hahaha, smesno"
echo ${tekst#ha*a}      # ukloniće 'haha'
echo ${tekst##ha*a}     # ukloniće 'hahaha'
```

Slično prethodnome, samo za sufikse, služe oblici '\${parametar%sufix}' i '\${parametar%%sufix}'.

Zamena unutar stringa

Oblik '\${parametar/izraz/zamena}' će unutar vrednosti parametra 'parametar' tražiti 'izraz' koji se tretira kao fajn globing izraz. Prvi niz znakova koji odgovara izrazu 'izraz' će biti zamenjen stringom 'zamena'. Ako postoji više nizova znakova koji odgovaraju izrazu 'izraz' unutar vrednosti parametra 'parametar', samo će prvi niz znakova biti zamenjen. Izuzetno, ako 'izraz' počinje sa '/' onda će svi nizovi znakova koji odgovaraju izrazu 'izraz' biti zamenjeni. Ako 'izraz' počinje sa '#' niz znakova koji odgovara izrazu mora biti na početku stringa. ako 'izraz' počinje sa '%' onda niz znakova koji odgovara izrazu mora biti na kraju stringa. Ako je 'zamena' prazan string, onda drugi znak '/' nije potreban, tj. oblik postaje '\${parametar/izraz}' i nizovi znakova koji odgovaraju izrazu 'izraz' će biti obrisani iz stringa. Primer:

```
#!/bin/bash
#
# Skript koji demonstrira referenciranje parametra
# zamenom unutar stringa
#

tekst="param1#param2"
echo ${tekst/param/value} # ispisaće 'value1#param2'
echo ${tekst//param/value} # ispisaće 'value1#value2'
```

Promenljive okruženja

Da li se varijable inicijalizovane u komandnom interpreteru vide u skriptovima koje pokrećemo u tom interpreteru? Probajmo sledeće:

- napravimo skript 'ispis.sh' koji treba da glasi ovako:

```
#!/bin/bash
#
# Skript koji ispisuje vrednosti varijabli
#

echo "Var1 = $var1"
```

- Zatim u komandnom interpreteru dodelimo neku vrednost varijabli 'var1':


```
$ var1="Neka vrednost"
$ ./ispis.sh
Var1 =
```

Kao što se vidi, vrednost varijable se nije prenela u skript, jer se on izvršava u zasebnom procesu - subshell-u.

- Ako bismo skript pokrenuli u tekućem komandnom interpreteru sa:

```
$ source ./ispis.sh
Var1 = Neka vrednost
```

vidimo da promenljive zadržavaju svoju vrednost unutar procesa.

Vrednost promenljive je moguće preneti u podproces tako što se ta promenljiva pretvori u promenljivu okruženja (engl. environment variable).

Napomena

Svaki proces ima svoje okruženje, koje čine različite varijable okruženja sa njihovim vrednostima. Okruženje proces nasleđuje od svog roditelja.

Promenljiva postaje promenljiva okruženja tako što se 'eksportuje':

```
$ export var1...
```

Napomena

Prilikom eksportovanja potrebno je navesti ime varijable, bez referenciranja!

Ako posle prethodne komande izvršimo prethodni skript u podprocesu:

```
$ ./ispis.sh
Var1 = Neka vrednost
```

Važno je istaći da su promenljive okruženja nekog procesa **kopije** promenljivih okruženja njegovog roditelja. To znači da, ako oba procesa izvršavamo paralelno, promena vrednosti neke varijable okruženja u jednom od procesa neće izazvati promenu kod drugog procesa jer svaki radi sa svojom kopijom. Dakle, svaka promena vrednosti varijable okruženja u procesu-detetu važi samo za taj proces i njegove potomke. Izmena vrednosti u procesu-roditelju je vidljiva njegovim potomcima samo ukoliko su oni pokrenuti **nakon** izmene vrednosti.

Promenljive okruženja koje postavlja komandni interpreter

Komandni interpreter sam postavlja neke varijable i parametre okruženja od kojih neke možemo menjati.

- **PATH** – promenljiva koja sadrži listu direktorijuma (razdvojenih „:“) u kojima komandni interpreter traži izvršne programe kada korisnik ne navodi putanju, već samo naziv programa.
- **HOME** – promenljiva koja sadrži početni direktorijum korisnika

- **SHELL** – puno ime komandnog interpretera (sa apsolutnom putanjom)
- **EDITOR** – puno ime podrazumevanog editora teksta
- **VISUAL** – puno ime podrazumevanog editora teksta, programi obično prvo pokušavaju da pozovu \$VISUAL pa tek onda \$EDITOR
- **PWD** – read-only promenljiva koja sadrži tekući direktorijum
- **PPID** – PID roditelja komandnog interpretera
- **PS1** – osnovni prompt
- **PS2** – prompt drugog nivoa (npr. kada se koriste petlje u komandnoj liniji), obično '> '
- **LANG** – promenljiva koja sadrži oznaku 'lokala' (engl. locale), tj. podrazumevanog jezika sistema. Koristi se za one kategorije koje LC_* varijable ne pokrivaju.
- **LC_ALL** - osnovna promenljiva 'lokala', ima prioritet u odnosu na \$LANG i sve ostale \$LC_* promenljive
- **LC_COLLATE** – promenljiva koja definiše redosled alfabetskog sortiranja prema zadatom jeziku
- **LC_NUMERIC** – promenljiva koja definiše prikaz numeričkih vrednosti prema zadatom jeziku
- **TMPDIR** – promenljiva koja sadrži putanju direktorijuma koji služi sa smeštanje privremenih fajlova, obično '/tmp'.
- **IFS** – (Internal Field Separator), sadrži znake koji predstavljaju delimitere tokena prilikom parsiranja koamndne linije, standardno '<space><tab><newline>'.

Za potpun spisak promenljivih koje postavlja bash komandni interpreter, pogledajte njegovu man stranu.

Parametri koje bash postavlja su:

- **?** - izlazni status poslednje izvršene komande ili komandnog skripta
- **\$** - PID komandnog interpretera
- **!** - PID poslednje izvršene komande u pozadini

Pored ovih, bash postavlja i parametre *****, **@** i **#**, o kojima će više reči biti u poglavlju „Prenošenje argumenata u shell skript“.

Standardno, promeljive sadrže tzv. skalarne podatke. Međutim, bash komandni interpreter omogućava da varijable sadrže i jednodimenzionalne nizove. Ti nizovi mogu biti indeksirani ili asocijativni.

Indeksirani nizovi

Indeksirani nizovi su nizovi čiji je indeks cifra. Bash omogućava indeksirane nizove kod kojih indeksi ne moraju biti uzastopni. Indeksirani niz se definiše sa:

```
$ var[n]=vrednost
```

gde su:

- **var** – ime promenljive
- **n** – bročani indeks
- **vrednost** – vrednost koja se dodeljuje elementu niza.

Drugi način je deklarisanje promenljive kao indeksiranog niza korišćenjem ključnih reči 'declare', 'local' ili 'readonly'. Npr:

```
$ declare -a var
```

gde je:

- **var** – ime promenljive

Element indeksirane varijable se referencira sa:

```
${var[n]}
```

gde su vitičaste zagrade obavezne kako bi sprečile da se '[n]' tretira kao fajl globing izraz. Prilikom referenciranja moguće je referencirati istovremeno sve elemente niza tako što se umesto bročanog indeksa stavi '*' ili '@'. Vrednosti asocijirane elementima niza su spojeni u jedan string i razdvojene su prvim znakom iz \$IFS promenljive (standardno, to je blanko znak).

Indeksi počinju od 0. Moguće je kao indeks navesti i negativan broj, u kom slučaju se stvarni indeks računa po formuli:

$$\text{stvarni_indeks} = \text{maks_indeks} + 1 - |\text{negativni_indeks}|$$

tako, indeks -1 označava poslednji element niza ($\text{maks_indeks} + 1 - |-1| = \text{maks_indeks} + 1 - 1 = \text{maks_indeks}$), -2 preposlednji element niza itd.

Ako elementi niza koji imaju dodeljene neke vrednosti nisu uzastopni, podrazumeva se da elementi sa preostalim indeksima takođe postoje i da imaju dodeljenu vrednost nul stringa. Tako npr. ako definišemo:

```
$ niz[3]='treci element'
```

ovim smo automatski definisali i elemente 'niz[0]', 'niz[1]' i 'niz[2]', koji svi imaju vrednost nul stringa.

Indeksiranim nizovima možemo dodeljivati više vrednosti odjednom, komandom:

```
$ ind_niz=( elem1 elem2 elem3... )
```

gde ind_niz[0] dobija vrednost 'elem1', ind_niz[1] dobija vrednot 'elem2' a ind_niz[2] dobija vrednost 'elem3'. Ako želimo da imamo indekse koji nisu uzastopni onda koristimo drugi oblik ove komande:

```
$ ind_niz=( [n1]=elem1 [n2]=elem2 [n3]=elem3 )
```

gde su n1, n2 i n3 brojučani indeksi za elem1, elem2 i elem3, respektivno.

Broj elemenata u nizu dobijamo sa:

```
${#var[@]}
```

Napomena:

Pazite da umesto \${#var[@]} ne napišete \${#var} – ovo će vratiti dužinu stringa u var[0] a ne broj elemenata u nizu 'var'.

Asocijativni nizovi

Asocijativni nizovi se od indeksiranih nizova razlikuju samo po tome što indeksi kod asocijativnih nizova mogu biti proizvoljni nizovi znakova. Ovi indeksi ne moraju da budu unutar znaka navodnika, npr:

```
$ config[pid]=$$
```

će definisati asocijativni niz 'config' koji ima jedan element. Ključ tog elementa je 'pid' a vrednost je PID shell procesa.

Asocijativne nizove takođe možemo deklarisati sa 'declare -A', 'local -A' ili 'readonly -A', npr:

```
$ declare -A asoc_niz
```

gde je 'asoc_niz' ime niza.

Ako želimo da nekom asocijativnom nizu dodelimo više elemenata niza, to možemo uraditi sa:

```
$ asoc_niz=( [kljuc1]=vrednost1 [kljuc2]=vrednost2 ... )
```

Dužinu niza možemo dobiti na isti način kao kod indeksiranih nizova, sa:

```
${#var[@]}.
```

Kontrola toka programa

Komande ne moraju da se izvršavaju jedna iza druge, već možemo praviti alternativne tokove unutar skripta, kao i beskonačne ili konačne petlje, baš kao i u ostalim programskim jezicima.

Kontrola toka: if

Osnovna struktura izmene toka programa je if...then struktura čija je osnovna sintaksa:

```
if cmd; then
    # niz komandi koje se izvršavaju kada je izlazna vrednost 'cmd' 0
    cmd1
    cmd2
    ...
fi
```

Ova sintaksa može biti proširena sa blokom koji se izvršava uslovno kada izlazna vrednost komande 'cmd' nije jednaka 0:

```
if test_cmd; then
    # blok komandi koji se izvršava ako je izl. vrednost test_cmd
    jednaka 0
    cmd1
    cmd2
    ...
else
    # blok komandi koji se izvršava ako izl. vrednost test_cmd nije 0
    cmd3
    cmd4
    ...
fi
```

Više uslova se može kombinovati ovako:

```

if test_cmd1; then
    # blok komandi koji se izvršava ako je izl. vrednost test_cmd1
    jednaka 0
    cmd1
    cmd2
    ...
elif test_cmd2; then
    # blok komandi koji se izvršava ako izl. vrednost test_cmd1 nije 0
i
    # ako je izl. vrednost test_cmd2 jednaka 0
else
    # blok komandi koji se izvršava ako izl. vrednost test_cmd2 nije 0
    cmd3
    cmd4
    ...
fi

```

Petlje: while i for

Petljom 'while' možemo izvršavati niz komandi sve dok je izlazna vrednost test-komade jednaka 0. Test-komanda se izvršava pre svakog prolaza petlje. Sintaksa ove strukture je:

```

while test_cmd; do
    cmd1
    cmd2
    ...
done

```

Petljom 'for' možemo izvršavati niz komandi u petlji dok kontrolna promenljiva preuzima navedene vrednosti jednu po jednu. Ispred svakog prolaza kroz petlju, kontrolna promenljiva preuzima novu vrednost. Ova struktura ima dva oblika, koji se ponašaju isto, ali su vrednosti kontrolne promenljive različite:

```

for kontrolna_promenljiva in vrednost1, vrednost2,...; do
    cmd1
    cmd2
    cmd3
    ...
done

```

U ovom slučaju kontrolna_promenljiva uzima u svakom prolazu kroz petlju vrednosti 'vrednost1', 'vrednost2', itd. koje ne moraju imati isti oblik i značenje.

U ovom obliku for petlje:

```

for ((k_prom=poc_vrednost; k_prom<krajnja_vrednost; k_prom++)); do
    cmd1
    cmd2
    cmd3
    ...
done

```

U ovom slučaju kontrolna promenljiva 'k_prom' uzima numeričke vrednosti redom od poc_vrednost do krajnja_vrednost, uključivo, inkrementirajući se za 1 u svakom prolazu.

U oba slučaja kontrolna promenljiva se može na standardna način referencirati u telu petlje, što je, u stvari, i namena kontrolne promenljive i same petlje.

Dve specifične komande za sve oblike petlji su 'break' i 'continue'. Komanda 'break' će prekinuti izvršavanje petlje i nastaviti izvršavanje skripta od prve komande koja stoji iza petlje. Komanda 'continue' će 'skratiti' izvršavanje komandi u telu petlje, tako što će preskočiti sve komande u telu do 'done' i krenuti sa novim prolazom. Naravno, obe ove komande imaju smisla u if...then kontrolnoj strukturi unutar petlje.

Primeri:

```
#!/bin/bash
#
# Primer 4 A - while petlja
#

echo "Primer za while petlju"
while
    echo -n "Unesi tekst ('kraj' za kraj): " && read a;
do
    if [ "$a" = "kraj" ] ; then
        echo "Završavam..."
        break
    fi
    echo "Procitao $a"
done
```

```
#!/bin/bash
#
# Primer 4 B - for petlja
#

echo "Primer za for petlju"
for i in {1,2,3,4,5,6,7,8,9} ; do
    echo $i
done

echo "Primer 2 za for petlju"
for ((i=1; i<10; i++)); do
    echo $i;
done

echo "Primer 3 za for petlju"
for i in `seq 1 9`; do
    echo $i;
done
```

```
#!/bin/bash
#
# Primer 4 C - for petlja sa if
#

for i in *; do
    if [ -x "$i" ]; then
        echo -n "Fajl $i se moze izvorsavati "
    fi
    if [ -w "$i" ]; then
        echo -n "Fajl $i se moze pisati "
    fi
    if [ -r "$i" ]; then
        echo "i procitati"
    else
        echo
    fi
done
```

Grananje: case

Struktura kontrole toka 'case' izvršava odgovarajući blokkomandi u zavisnosti od vrednosti kontrolne promenljive. Njena sintaksa je:

```
case kontrolna_promenljiva in
vrednost1)
    blok_komandi_1
    ;;
vrednost2)
    blok_komandi_2
    ;;
...
*)
    blok_komandi_za_sve_ostale_vrednosti
    ;;
esac
```

Primer:


```
#!/bin/bash
a="$1"

case $a in
1)
    echo "Argument je 1"
marko)
    echo "Argument je 2 lala"
    ;;
5)
    echo "Argument jeee 5 ...."
    ;;
\*)
    echo "Zvezdica..."
    ;;
*)
    echo "Nesto.."
    ;;
esac
```

Navedene strukture su samo osnovni primeri mogućnosti bash skriptova. Predlažemo vam da pogledate man stranu za 'bash' komandu, kao i brojne primere i tutorijale koje se za šel skripting mogu naći na Internetu.

Funkcije

Ukoliko se unutar skripta pojavljuje niz komandi koje se istim redosledom izvršavaju na više mesta, uz eventualno različite argumente, moguće je taj niz komandi zameniti funkcijom i kasnije tu funkciju pozivati (uz opcione argumente) kao zamenu za takav niz komandi.

Funkcije se deklariraju sa:

```
ime_funkcije () {  
    cmd1  
    cmd2  
    ...  
}
```

Funkcije se referišu preko svog imena.

Funkcije mogu imati argumente koji se u definiciji funkcije pojedinačno referišu kao '\$1'...'9'. Vrednosti argumenata se navode pri referisanju funkcije i kao vrednost se uzima najduži mogući niz znakova ograničen blanko znakovima (ukoliko želimo da neki argument sadrži i blanko znakove koristimo znakove navoda). Argumenata može biti i više od 9 ali se ti argumenti ne mogu referencirati direktno, već se mora koristiti šiftovanje.

Prilikom šiftovanja nekadašnja vrednost \$2 postaje \$1, \$3 postaje \$2 itd. dok se vrednost \$1 gubi, pa ako nam je potrebna treba se sačuvati u nekoj promenljivoj.

Argumenti se takođe mogu grupno referencirati preko '\$*' i '\$@'. Razlika između ova dva se vidi kada se argumenti funkcije tako proslede nekoj komandi. Onda, u slučaju da napišemo 'cmd \$*', komanda 'cmd' će sve argumente videti kao jedan (kao da smo napisali 'cmd „arg1 arg2 ...“'). U slučaju 'cmd \$@' svi argumenti ostaju nezavisni (kao da smo napisali 'cmd „arg1“ „arg2“ „arg3“ ...').

Funkcije mogu imati svoje promenljive, koje neće biti vidljive van funkcije. Ove promenljive se definišu sa:

```
func1 () {  
    local lokalna_promenljiva=vrednost  
    ...  
}
```

Ukoliko se desi slučaj da i lokalna promenljiva i promenljiva interpretera, odnosno okruženja imaju isto ime, onda je unutar funkcije vidljiva samo lokalna promenljiva.

Izmene promenljivih okruženja odnosno promenljivih interpretera u telu funkcije se pamte po završetku izvršavanja funkcije.

Primer:

```
#!/bin/bash
#
# Primer 3
#

func1 () {
    local var1="Lokalna promenljiva"
    echo "Ja sam funkcija!"
    echo "var1 = $var1"
    echo "var2 = $var2"
    var2="Nova vrednost"
    echo Argumenti funkcije: "$1" "$2" "$3"
}

var1="Globalna promenljiva 1"
var2="Globalna promenljiva 2"

func1 argument1 argument2 "$var1"

echo "Posle poziva funkcije:"
echo "var1 = $var1"
echo "var2 = $var2"
```

Prenošenje argumenata u shell skript

Prosleđivanje argumenata skriptu je moguće, kao i bilo kojoj drugoj komandi. Slično kao prosleđivanje argumenata funkciji, i kod prosleđivanja argumenata skriptu koriste se pozicioni parametri \$1, \$2,...\$9, \$* i \$@. Sva pravila, uključujući i šiftovanje parametara važe i za prosleđivanje argumenata samom skriptu (kao da se skript jedna velika funkcija).

Pored navedenih argumenata, postoje još dva parametra koja se prosleđuju skriptu a koja su nam korisna, a to su:

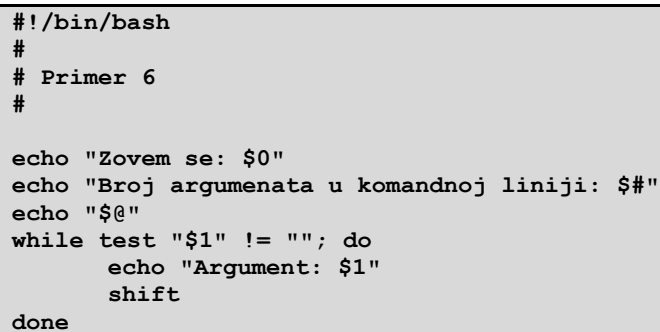
\$0 – ime samog skripta, onako kako je navedeno u komandnoj liniji. Koristiti 'basename \$0' da bi se dobilo samo ime skripta, bez eventualne putanje (ako je ista navođena u komandnoj liniji).

\$# - broj argumenata u komandnoj liniji. Najčešće se koristi u slučajevima kada skript zahteva tačan ili minimalan broj argumenata.

Primer:

```
#!/bin/bash
#
# Primer 6
#

echo "Zovem se: $0"
echo "Broj argumenata u komandnoj liniji: $#"
```



```
echo "$@"
while test "$1" != ""; do
    echo "Argument: $1"
    shift
done
```

Uslovni izrazi

Za uslovne izraze možemo koristiti komandu 'test' ili ugrađene uslovne izraze. Komanda 'test' ima svoj sinonim u obliku '[' što je simbolički link na 'test'. Komandu u tom slučaju možemo pisati kao:

```
test izraz
```

ili

```
[ izraz ]
```

Ugrađene uslovne izraze možemo koristiti umesto 'test' komande od koje se razlikuju samo po tome što se leksičko poređenje vrši po lokalno. Komanda 'test' leksička poređenja uvek vrši po ASCII rasporedu. Druga razlika je ta da se ugrađeni uslovni izrazi kombinuju na sledeći način:

- `[[izraz1 && izraz2]]` - tačan ako su oba izraza tačna
- `[[izraz1 || izraz2]]` - tačan ako je makar jedan od izraza tačan
- `[[! izraz]]` - tačan ako izraz nije tačan

Kod komande 'test', izrazi se kombinuju na ovaj način:

- `[izraz1 -a izraz2]` - tačan ako su oba izraza tačna
- `[izraz1 -o izraz2]` - tačan ako je makar jedan od izraza tačan
- `[! izraz]` - tačan ako izraz nije tačan

'izraz' može biti:

- `-a fajl` – tačan ako 'fajl' postoji (ne postoji kod 'test' komade)
- `-e fajl` – tačan ako 'fajl' postoji
- `-f fajl` – tačan ako je 'fajl' regularan fajl
- `-d fajl` – tačan ako je 'fajl' direktorijum
- `-h fajl` – tačan ako je 'fajl' simbolički link
- `-b fajl` – tačan ako je 'fajl' blok specijalni fajl
- `-c fajl` – tačan ako je 'fajl' karakter specijalni fajl

- -p fajl – tačan ako je 'fajl' imenovani pajp (FIFO)
 - -S fajl – tačan ako je 'fajl' soket
-
- -r fajl – tačan ako korisnik ima pravo 'r' nad 'fajlom'
 - -w fajl – tačan ako korisnik ima pravo 'w' nad 'fajlom'
 - -x fajl – tačan ako je 'fajl' izvršan, odnosno ako je 'fajl' direktorijum i korisnik ima pravo 'x' na njemu
-
- -u fajl – tačan ako 'fajl' ima setovan setUID bit
 - -g fajl – tačan ako 'fajl' ima setovan seGID bit
 - -k fajl – tačan ako 'fajl' ima setovan 'sticky' bit
-
- -s fajl – tačan ako je dužina fajla veća od nule
-
- fajl1 -ef fajl2 – tačan ako su 'fajl1' i 'fajl2' hard linkovi na isti i-nod
 - fajl1 -nt fajl2 – tačan ako je 'fajl1' noviji od 'fajl2'
 - fajl1 -ot fajl2 – tačan ako je 'fajl1' stariji od 'fajl2'
-
- -z string – tačan ako je 'string' prazan string
 - -n string – tačan ako 'string' nije prazan string
 - string1 = string2 – tačan ako je 'string1' jednak 'string2'
 - string1 != string2 – tačan ako 'string1' nije jednak 'string2'
 - string1 > string2 – tačan ako je 'string1' u leksikografskom sort poretku iza 'string2'
 - string1 < string2 – tačan ako je 'string1' u leksikografskom sort poretku ispred 'string2'
-
- arg1 -eq arg2 – tačan ako je 'arg1' aritmetički jednak 'arg2'
 - arg1 -ne arg2 – tačan ako je 'arg1' aritmetički različit od 'arg2'

- `arg1 -lt arg2` – tačan ako je 'arg1' aritmetički manji od 'arg2'
- `arg1 -le arg2` – tačan ako je 'arg1' aritmetički manji ili jednak od 'arg2'
- `arg1 -gt arg2` – tačan ako je 'arg1' aritmetički veći od 'arg2'
- `arg1 -ge arg2` – tačan ako je 'arg1' aritmetički veći ili jednak od 'arg2'

Primer:

```
#!/bin/bash
#
# primeri uslovnih izraza

for fajl in *; do
    echo -n "$fajl je "
    if [[ -f "$fajl" ]]; then
        echo -n "regularan fajl "
    elif [[ -d "$fajl" ]]; then
        echo -n "direktorijum "
    elif [[ -h "$fajl" ]]; then
        echo -n "simbolički link "
    elif [[ -b "$fajl" ]]; then
        echo -n "blok specijalni fajl "
    elif [[ -c "$fajl" ]]; then
        echo -n "karakter specijalni fajl "
    elif [[ -p "$fajl" ]]; then
        echo -n "imenovani pajp "
    else
        echo -n "soket "
    fi

    echo -n "i ja imam pravo "
    [[ -r "$fajl" ]] && echo -n "r"
    [[ -w "$fajl" ]] && echo -n "w"
    [[ -x "$fajl" ]] && echo -n "x"
    echo
done
```

Subshellovi i izvršavanje komandi iz drugog skripta u istom procesu

Ponekad je potrebno 'uhvatiti' ispis neke komande i upotrebiti je dalje u skriptu. Ovo možemo uraditi tako što izlaz komande u skriptu preusmerimo u neki privremeni fajl, a zatim učitati taj fajl. No, ovo često može biti prekomplikovano, sporo i nebezbedno. Iz tog komandni interpreter omogućava da se niz komandi izvrši u zasebnom podprocesu (subshell) a da se rezultujući ispis na standardni izlaz 'uhvari' i, obično dodeli nekoj varijabli. U tom slučaju, ako je ispis bio u više redova, znaci za novi red će biti izostavljeni i vrednost dodeljena varijabli će biti jedna linija.

Postoje dva oblika izvršavanja niza komandi u podprocesu:

```
'niz komandi`  
$(niz komandi)
```

U oba slučaja, ekspanzija parametara se dešava pre izvršavanja niza komandi u podprocesu. U oba slučaja, izlazni status poslednje komande u nizu je izlazni status celog konstrukta.

Primer:

```
#!/bin/bash  
#  
# primer izvršavanja komande u subshell-u  
  
iface="wlan0"  
ipaddr=`ifconfig $iface | grep 'inet addr:' | \  
    sed -re 's/^.*inet addr:([0-9.]+).*/\1/'`  
  
echo „IP adresa interfejsa $iface je $ipaddr“
```

Ukoliko želimo da izvršimo neki eksterni skript u našem skriptu, on će se izvršavati kao i svaka druga komanda, tj. izvršiće se u zasebnom procesu. Ovo nas sprečava da, recimo, postavimo neke vrednosti promenljivih ili definišemo set funkcija. Iz tog razloga, komandni interpreter nam omogućava da učitamo sadržaj nekog eksternog skripta i izvršimo ga unutar procesa u kojem izvršavamo i naš skript. Komanda za učitavanje je:

```
source /putanja/do/skripta
```

ili, jednostavno

```
./putanja/do/skripta
```


Rad sa opcijama

Naši skriptovi, kao i regularni programi mogu imati opcije koje modifikuju izvršavanje skripta. Opcije možemo tretirati kao i bilo koje druge argumente koje prenosimo u skript, ali onda moramo praviti sopstvene rutine za parsiranje opcija. Ta rutina se usložnjava ako koristimo i kratke i duge opcije. Dodatno usložnjavanje se dešava ako neke od opcija koje budemo definisali mogu imati svoje argumente, a posebno ako ti argumenti nisu obavezni. Takođe, ako želimo da korisniku koji izvršava naš skript želimo da omogućimo da meša opcije i stvarne argumente skripta, rutina za analizu ulaznih argumenata će biti veoma kompleksna.

Srećom, ovu obradu ulaznih argumenata možemo prepustiti programu 'getopt' koji će sve argumente analizirati i preurediti tako da prvo navodi opcije a zatim delimiter '-' i onda stvarne argumente skripta. Ako neka od opcija ima argument, taj argument će biti naveden odmah iza te opcije. Na taj način moguće je jednostavno parsirati komandnu liniju i razdvojiti opcije (i njihove argumente) od stvarnih argumenata skripta.

Format kojim pozivamo getopt je:

```
params=`getopt -o kratke_opcije -long duge_opcije - "$@"`
```

Drugim rečima, komanda 'getopt' vraća listu upotrebljenih opcija i njihovih argumenata, delimiter '-' i listu stvarnih argumenata.

Argument 'kratke_opcije' je niz znakova koji definiše kratke opcije. Ako opcija ima obavezan argument, iza slova opcije treba navesti znak '!', a ako je taj argument neobavezan, navodi se niz ':'. Tako, npr. niz 'ab:c:' označava da naš skript prepoznaje opcije:

- -a – bez argumenata
- -b – sa obaveznim argumentom (može se pisati odmah uz opciju ili sa razmakom)
- -c – sa neobaveznim argumentom koji, ako se koristi mora biti napisan odmah uz opciju, bez razmaka

Argument 'duge_opcije' je niz naziva dugih opcija razdvojenih znakom ','. Ako duga opcija ima obavezan argument onda iza njenog imena treba staviti '!', a ako je argument neobavezan, treba iza imena staviti ':'. Tako, npr. niz 'about,block;cont:' označava da naš skript prepoznaje sledeće dugačke opcije:

- --about, bez argumenata
- --block, sa obaveznim argumentom, koji može biti napisan kao '--block block_arg' ili kao '--block=block_arg'
- --cont, sa neobaveznim argumentom koji može biti napisan kao '--cont=cont_arg'

Ako pozovemo naš skript sa izostavljenim obaveznim argumentom opcije, getopt će prijaviti grešku i izlazni status komande će biti 1. U suprotnom, ako je sve ispravno, izlazni status će biti 0.

Konačno, rezultujuću listu možemo parsirati kao u sledećem primeru:

```
#!/bin/bash
#
# primer parsiranja getopt rezultata

parse_args() {
    while true; do
        case "$1" in
            -a|--about) echo "Aktivirana opcija 'about'"
                        shift
                        ;;
            -b|--block) echo "Aktivirana opcija 'block' sa argumentom
$2"
                        shift 2
                        ;;
            -c|--cont) echo -n "Aktivirana opcija 'cont' "
                        if [[ "$2" = "" ]]; then
                            echo "bez argumenta"
                        else
                            echo "sa argumentom $2"
                        fi
                        shift 2
                        ;;
            '--') shift
                   break
                   ;;
            *) echo "Interna greška"
               esac
        done
        while [[ -n "$1" ]]; do
            echo "Stvarni argument: $1"
            shift
        done
    }

params=`getopt -o ab:: --long about,block:,cont:: -n t.sh -- "$@"`
result=$?
if [[ $result -ne 0 ]]; then
    exit 1
fi

parse_args $params
```

Aritmetika

Komandni interpreter bash direktno podržava celobrojnu aritmetiku. Aritmetički izraz se piše kroz konstrukt:

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

u izrazu možemo koristiti varijable bez potrebe da ih referenciramo. Pored varijabli možemo koristiti i konstante, koje zajedno nazivamo identifikatorima.

Operatori koje možemo koristiti su:

- `++var` – pre-inkrementiranje varijable
- `--var` – pre-dekrementiranje varijable
- `var++` – post-inkrementiranje varijable
- `var--` – post-dekrementiranje varijable
- `-var` – vraćanje negativne vrednosti varijable
- `!var` – logička negacija varijable
- `~id` – bitska negacija identifikatora
- `id1**id2` – stepenovanje id1 sa stepenom id2
- `id1*id2` – umnožak id1 i id2
- `id1/id2` – celobrojno deljenje id1 sa id2
- `id1%id2` – ostatak pri celobrojnem deljenju id1 sa id2
- `id1 + id2` – zbir dva identifikatora
- `id1 - id2` – razlika dva identifikatora
- `id1<<id2` – bitsko pomeranje (šiftovanje) udesno id1 za id2 bitova
- `id1>>id2` – bitsko pomeranje (šiftovanje) ulevo id1 za id2 bitova
- `id1<=id2` – logičko poređenje da li je id1 manje ili jednako id2
- `id1<id2` – logičko poređenje da li je id1 manje od id2
- `id1>id2` – logičko poređenje da li je id1 veće od id2
- `id1>=id2` – logičko poređenje da li je id1 veće ili jednako id2

- $id1 != id2$ – logičko poređenje da li je $id1$ različito od $id2$
- $id1 == id2$ – logičko poređenje da li je $id1$ jednako $id2$
- $id1 \& id2$ – bitsko $id1$ AND $id2$
- $id1 \wedge id2$ – bitsko $id1$ XOR $id2$
- $id1 | id2$ – bitsko $id1$ OR $id2$
- $id1 \&\& id2$ – logičko $id1$ AND $id2$
- $id1 || id2$ – logičko $id1$ OR $id2$
- $var = id$ – dodela vrednosti
- $var += id$ – dodela vrednosti sa sabiranjem
- $var -= id$ – dodela vrednosti sa oduzimanjem
- $var *= id$ – dodela vrednosti sa množenjem
- $var /= id$ – dodela vrednosti sa celobrojn timer deljenjem
- $var \% id$ – dodela vrednosti sa ostatkom celobrojn timer deljenja
- $var <=< id$ – dodela vrednosti sa bitskim šiftovanem udesno
- $var >> id$ – dodela vrednosti sa bitskim šiftovanem ulevo
- $var \& id$ – dodela vrednosti sa bitskim AND
- $var | id$ – dodela vrednosti sa bitskim OR
- $var \wedge id$ – dodela vrednosti sa bitskim XOR

U svim ovim slučajevima, konstrukt:

| |
|------------------------------------|
| $\\$((izraz))$ |
|------------------------------------|

vraća aritmetičku vrednost izraza, bez obzira da li se vrednost neke varijable u izrazu menja (dodelom ili inkrementiranjem).

Napredan rad sa terminalima

Standardno, ispis na ekran obavljammo komandom `echo`, koja ispisuje jednu liniju teksta. No, 'echo' može biti instruiran ne dodaje automatski znak za kraj reda prilikom ispisa. Opcija za to je:

```
$ echo -n tekst
```

U ovom slučaju će tekstualni kursor ostati u istoj liniji u kojoj se nalazi ispis 'tekst', pozicioniran na kraju ispisa. Ovo nam omogućava da kreiramo promptove u interaktivnim skriptovima, kao i da ispis linije obavljammo iz više koraka.

Komanda 'echo' nam takođe omogućava da ispisujemo specijalne znakove, koristeći opciju '-e':

```
$ echo -e tekst_sa_specijalnim_znacima
```

U specijalne znake ubrajamo:

- `\a` – alert (zvučni signal)
- `\b` – backspace (brisanje prethodnog znaka)
- `\c` – blokiranje daljeg ispisa
- `\e` ili `\E` – ispis znaka ESC (ASCII kod 27)
- `\f` – form feed
- `\n` – nova linija
- `\r` – ispis znaka CR (carriage return)
- `\t` – horizontalni tab
- `\v` – vertikalni tab
- `\\` - ispis znaka '\'
- `\0nnn` – ispis znaka čiji je ASCII kod oktalna vrednost nnn
- `\xnn` – ispis znaka čiji je ASCII kod heksadekadna vrednost nn
- `\unnnn` – ispis znaka čiji je UTF-8 kod heksadekadna vrednost nnnn

Za napredniji ispis posebno je interesantno što na ovaj način možemo ispisivati tzv. escape sekvence. Escape sekvence su nizovi znakova koji počinju specijalnim znakom ESC (`\e`). Ove sekvence se, u zavisnosti od terminala, za pozicioniranje kursora, učitavanje alternativnog seta znakova ili ispis znaka u drugoj boji.

U sledećem primeru ćemo na Linux konzoli i u xterm-u ispisati rečenicu različitim bojama:

```
#!/bin/bash
#
# Primer ispisa teksta u različitim bojama

res="\e[00;m"

declare -a boje
boje=( [30]=Black [31]=Red [32]=Green [33]=Yellow [34]=Blue \
       [35]=Magenta [36]=Cyan [37]=White )

tekst="Ovo je recenica u boji!"

for ((i=30; i<38; i++)); do
    echo -e "\e[01;${i}m$tekst: ${boje[$i]}$res\n"
done
```

Formatirani ispis, gde vodimo računa o kolonama i širinama ispisa dobijamo upotrebom 'printf' komande. Ova komanda ima format:

```
$ printf fo rmat argumenti
```

Argument 'format' ima isto značenje kao i kod C funkcije 'printf()'.



www.atc.rs