

# Abeceda lupine bash

## 1 Lupina bash

Lupina bash je sestavni del številnih unixovskih operacijskih sistemov, med drugim tudi mnogih distribucij sistema Linux (npr. Ubuntu). **Lupina je program, ki bere in sproti izvršuje besedilne ukaze.** Veliko ukazov je namenjenih delu z datotekami in imeniki, na voljo pa so tudi ukazi za številna druga opravila, na primer iskanje po datotekah, urejanje vsebine datotek itd. Ukaze je mogoče sestavljati v kompleksnejše celote, s katerimi lahko naloge, za katere bi v grafičnih uporabniških vmesnikih potrebovali na tisoče klikov, rešimo v eni vrstici.

Predpogoj za produktivno rabo lupine je dobro obvladovanje tipkovnice. Sledeče tipke imajo v lupini bash (in tudi številnih drugih lupinah) poseben pomen:

- ↑: Potovanje po zgodovini odtipkanih ukazov nazaj. Če enkrat pritisnemo tipko ↑, prikličemo prejšnji ukaz, če tipko pritisnemo še enkrat, dobimo predprejšnji ukaz itd.
- ↓: Potovanje po zgodovini odtipkanih ukazov naprej.
- Tab: Dopolnjevanje parametrov ukazov. Ta funkcija je še posebej uporabna, ko vnašamo relativne ali absolutne poti do datotek ali imenikov.

## 2 Zagon lupine

Okno, v katerem se izvaja lupina, se imenuje *konzola* ali *terminal*. V sistemu Ubuntu konzolo poženemo s kombinacijo **Ctrl-Alt-T**. Po odprtju okna vidimo takšen pozivnik:

```
uporabnik@računalnik:trenutniImenik$
```

Afna, dvopičje in dolar so zgolj ločila. Na računalniku z imenom *računalnik* je prijavljen uporabnik z uporabniškim imenom *uporabnik*, *trenutni imenik* pa je imenik, v kontekstu katerega izvršujemo ukaze. Na primer, če poženemo ukaz **ls** (»izpiši vsebino imenika«) brez parametrov, se izpiše vsebina trenutnega imenika.

Pozivnik od nas pričakuje vnos vrstice z ukazom. Taki vrstici bomo rekli **klic**. Ko svoj vnos potrdimo s tipko Enter, se klic izvrši.

## 3 Navadni uporabniki in superuporabnik

V unixovskih operacijskih sistemih obstaja stroga delitev na navadne uporabnike (ti lahko praviloma delajo samo s svojimi lastnimi datotekami in imeniki) in superuporabnike (ti lahko počnejo karkoli). Tudi če smo sam svoj gospodar na svojem računalniku, delujemo

v vlogi navadnega uporabnika, dokler z ukazom `sudo` izrecno ne zahtevamo izvedbe ukaza v imenu superuporabnika. Na primer, klic

```
mv /home /home1
```

zelo verjetno ne bo uspel, saj je privzeti lastnik imenika `/home` superuporabnik. Če pa se lahko postavimo v vlogo superuporabnika (na svojem računalniku to možnost ponavadi imamo), lahko poskusimo takole:

```
sudo mv /home /home1
```

Lupina nas bo prosila za vnos gesla. Če bo pravilno, se bo klic izvršil v imenu superuporabnika, zato se bo imenik `home` znotraj korenskega imenika uspešno preimenoval v `home1`.

**Pozor!** Superuporabnik ima veliko moč, zato pred vsako izvedbo ukaza `sudo` dobro premislimo, ali klic resnično želimo izvršiti.

## 4 Na splošno o ukazih in klicih

Vsak klic se prične z imenom ukaza, temu pa sledijo morebitni parametri in stikala. Stikala lahko imajo tudi sama svoje parametre. Na primer, pri klicu

```
sortocene.csv -t ';' -k '2,2' -r
```

se ukaz glasi `sort`, njegov parameter je `ocene.csv`, `-t`, `-k` in `-r` pa so stikala. Stikali `-t` in `-k` imata tudi sami svoja parametra.

Če nas zanima zgradba in delovanje določenega ukaza, uporabimo ukaz `man` ali `help`. Na primer, `man sort` podrobno izpiše podatke o ukazu `sort`, `help cd` pa nam obrazloži ukaz `cd`. Ukaz `man` nam pomaga pri samostojnih programih, kot je `sort`, ukaz `help` pa pri ukazih, ki so sestavni del same lupine. Seznam vgrajenih ukazov lupine pridobimo s klicem `man builtins`.

## 5 Datotečni sistem

*Datoteka* je zaokrožen skupek podatkov z določenim imenom, *imenik* (mapa) pa je vsebovalnik datotek in drugih imenikov. Imeniki in datoteke so v unixovskih operacijskih sistemih organizirani v hierarhično strukturo. Slika 1 prikazuje neznaten izsek take hierarhije, ki v praksi lahko vsebuje na milijone imenikov in datotek.

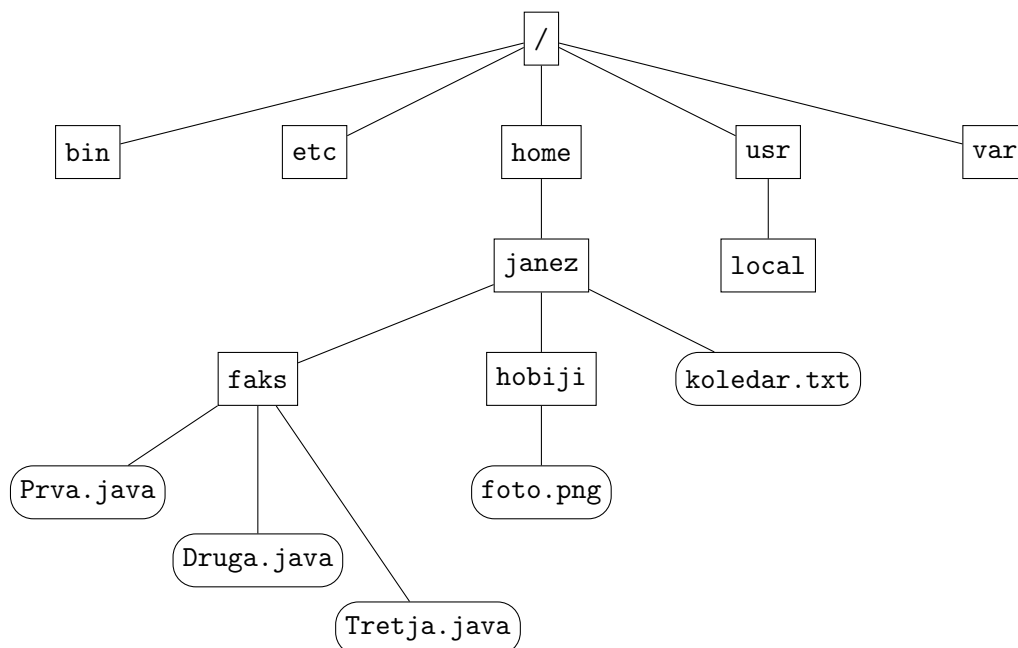
Z malo domišljije si lahko sliko 1 predstavljamo kot na glavo postavljeno drevo. Imenik na vrhu (`/`) je po tej interpretaciji koren(ina) drevesa, zato mu rečemo *korenski imenik*. *Poddrevo* imenika je sestavljeno iz imenika in vsega, kar je pod njim. Na primer, poddrevo imenika `/home/janez` na sliki 1 je sestavljeno iz imenikov `janez`, `faks` in `hobiji` ter datotek `Prva.java`, `Druga.java`, `Tretja.java`, `foto.png` in `koledar.txt`.

Pot od korenskega imenika do podanega elementa datotečnega sistema (datoteke ali imenika) se imenuje absolutna pot tega elementa. Absolutna pot se prične z znakom `/`, nato pa sledi seznam vmesnih elementov, ločen z znakom `/`. Seznam se zaključi s podanim elementom. Na primer, absolutna pot do datoteke `Prva.java` na sliki 1 se glasi `/home/janez/faks/Prva.java`, absolutna pot do imenika `local` pa `/usr/local`.

Imenik `/home/uporabniškoIme` (`/home/janez` na sliki 1) se imenuje *domači imenik* uporabnika. Ta imenik je namenjen hranjenju uporabnikovih imenikov in datotek, zato se večina absolutnih poti, s katerimi ima uporabnik opravka, prične s `/home/ime`. Zaradi tega obstaja v unixovskih sistemih priročna bližnjica: namesto `/home/uporabniškoIme` lahko uporabljamo kar tilde (`~`). Absolutno pot do datoteke `Prva.java` na sliki 1 lahko potemtakem zapišemo tudi kot `~/faks/Prva.java`.

Spomnimo se, da izraz *trenutni imenik* označuje imenik, ki je trenutno zapisan v pozivniku in v kontekstu katerega izvršujemo ukaze. *Relativna pot* do elementa je pot od trenutnega imenika do podanega elementa. Relativna pot se nikoli ne prične z znakom `/` (ta služi le kot ločilo med sestavnimi deli) ali `~`, lahko pa vsebuje znak `.` (trenutni oziroma isti imenik) ali zaporedje `..` (starševski imenik).

Na primer, če je naš trenutni imenik v hierarhiji s slike 1 `/home/janez`, potem se relativna pot do datoteke `Prva.java` glasi `faks/Prva.java` (najprej se pomaknemo v podimenik `faks`, tam pa že najdemo iskano datoteko), relativna pot do imenika `local` pa `../usr/local` (najprej se dvignemo v imenik `home` in zatem v korenski imenik, nato pa se spustimo najprej do imenika `usr` in potem še do imenika `local`). Če je naš trenutni imenik `/home/janez/faks`, pa se relativna pot do datoteke `Prva.java` glasi kar `Prva.java`, relativna pot do imenika `local` pa `../../usr/local`.



Slika 1: Primer hierarhije imenikov (pravokotni okvirji) in datotek (okvirji z zaobljenimi koti).

## 6 Nadomestna znaka ? in \*

Znaka `?` in `*` nam omogočata, da z enim izrazom opišemo veliko elementov datotečnega sistema. Znak `?` lahko nadomesti poljuben znak, znak `*` pa poljubno (tudi prazno) zaporedje znakov. Na primer, izraz `*.java` opredeljuje vse datoteke v trenutnem imeniku, ki se zaključijo z zaporedjem `.java`, izraz `naloge/a?b`, pa vse datoteke v podimeniku `naloge`, pri katerih je ime sestavljeno iz črke `a`, poljubnega znaka in črke `b`.

Ukaz se znakov ? in \* sploh ne zaveda, saj lupina ustrezne zamenjave izvrši še pred njegovo izvedbo. Na primer, če v imeniku ~/faks na sliki 1 izvršimo klic

```
cat *.java
```

potem ga bo lupina še pred zagonom ukaza cat nadomestila s klicem

```
cat Prva.java Druga.java Tretja.java
```

## 7 Osnovni ukazi za delo z datotekami in imeniki

V tem razdelku bomo nanizali nekaj ukazov za delo z elementi datotečnega sistema. Pri kazali bomo samo najbolj tipično rabo teh ukazov; večina od njih namreč ponuja še precej več možnosti. Primeri uporabe ukazov se bodo nanašali na sliko 1.

Oglati oklepaji v prikazu rabe ukaza označujejo neobvezne elemente, tropičja pa poljubno dolga zaporedja elementov. Na primer, pri ukazu

```
cp [-r] [-i] izvor... cilj
```

sta parametra *izvor* in *cilj* obvezna, stikali -r in -i pa nista. Parameter *izvor* lahko nadomestimo s poljubno dolgim zaporedjem elementov. Sledeči klic, denimo, je sintaktično pravilen:

```
cp -r Prva.java Druga.java ../hobiji ../kopije
```

Če bi znotraj domačega imenika uporabnika janez ustvarili podimenik kopije, nato pa se predstavili v imenik ~/faks in izvedli gornji klic, bi se datoteki Prva.java in Druga.java ter celotno poddrevo imenika ../hobiji skopirali v pravkar ustvarjeni imenik.

Ogledali si bomo sledeče ukaze:

- **pwd**

Prikaže absolutno pot do trenutnega imenika.

- **cd *imenik***

Zamenja trenutni imenik. Parameter *imenik* je absolutna ali relativna pot do želenega imenika. Na primer, če je naš trenutni imenik ~/faks, potem se bomo po izvedbi ukaza cd ../hobiji predstavili v imenik ../hobiji.

- **ls [*imenik*] [-l] [-a]**

Prikaže seznam datotek in imenikov v podanem imeniku. Če *imenik* izpustimo, se prikaže vsebina trenutnega imenika. Stikalo -l prikaže seznam z več podrobnostmi. Stikalo -a prikaže vse datoteke, tudi tiste, katerih ime se prične s piko (te so pri običajnem pregledu skrite).

- **mkdir *imenik***

Ustvari prazen imenik.

- **rmdir *imenik***

Izbriše prazen imenik.

- **cp [-r] [-i] *izvor...* *cilj***

Kopira eno ali več izvornih datotek v podani ciljni imenik. Če podamo stikalo `-r`, lahko kopiramo tudi celotna poddrevesa. Na primer, če znotraj imenika `~/hobiji` izvršimo klic

```
cp ../faks/*.java .
```

se bodo vanj (pika označuje trenutni imenik) skopirale vse datoteke s končnico `.java` iz imenika `~/faks`. Klic

```
cp -r ~/hobiji ~/faks/dodatno
```

pa skopira celoten imenik `~/hobiji` v imenik `~/faks/dodatno`.

**Pozor!** Ukaz `cp` bo morebitne istoimenske datoteke v ciljnem imeniku preprosto prepisal in s tem uničil, ne da bi nas na to opozoril. S stikalom `-i` dosežemo, da nas bo lupina pred prepisovanjem ciljne vsebine opozorila.

- `mv [-i] izvor... cilj`

Deluje podobno kot `cp`, le da datotek in imenikov ne kopira, ampak jih v ciljni imenik premakne. Ukaz `mv` v nasprotju z ukazom `cp` v nobenem primeru ne potrebuje stikala `-r`.

- `rm [-r] [-i] element...`

Izbriše podane datoteke, če podamo stikalo `-r`, pa lahko brišemo tudi poddrevesa.

**Pozor!** Ta ukaz je nevaren (še zlasti, če ga izvršimo s `sudo`), saj nas pred brisanjem ne opozori, poleg tega pa izbrisane vsebine ni več mogoče obnoviti. S stikalom `-i` dosežemo, da nas bo lupina pred vsakim brisanjem poprosila za potrditev.

- `touch datoteka`

Ustvari prazno datoteko, če datoteka že obstaja, pa posodobi njen čas zadnje spremembe.

- `cat datoteka...`

Po vrsti izpiše vsebino podanih datotek.

- `less datoteka`

Datoteko izpiše po straneh. S tipkama `PageUp` in `PageDown` se lahko premikamo po straneh naprej oziroma nazaj.

- `ln -s izvor [cilj]`

Ustvari simbolično povezavo (bližnjico). Po izvedbi ukaza bo posebna datoteka `cilj` kazala na datoteko oz. imenik `izvor`. Če cilja ne podamo, se v trenutnem imeniku ustvari bližnjica z imenom izvirne datoteke oziroma imenika. Na primer, če znotraj imenika `~/faks` poženemo

```
ln -s ../koledar.txt
```

dobimo v imeniku `~/faks` datoteko `koledar.txt`, ki služi kot bližnjica do datoteke `~/koledar.txt`.

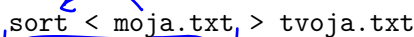
## 8 Vhod in izhod

Veliko ukazov bere podatke s *standardnega vhoda* in jih piše na *standardni izhod*. Privzeti standardni vhod je tipkovnica, privzeti standardni izhod pa zaslon. Na primer, če ukaz `sort` poženemo preprosto kot

```
sort
```

nam bo omogočil, da s tipkovnico vnesemo vrstice besedila, ko vnos zaključimo s kombinacijo tipk `Ctrl-D`, pa bo lupina na zaslon izpisala iste vrstice v leksikografskem vrstnem redu. Standardni vhod in/ali izhod pa lahko tudi *preusmerimo* — denimo na datoteko. To storimo z operatorjema `<` (za standardni vhod) in `>` (za standardni izhod). Na primer, če izvršimo klic

```
sort < moja.txt > tvoja.txt
```



bo ukaz `sort` svoj vhod namesto s tipkovnice prebral iz datoteke `moja.txt`, svoj izhod pa bo namesto na zaslon izpisal v datoteko `tvoja.txt`. Ukaz `sort` se preusmeritve sploh ne zaveda in še naprej bere s standardnega vhoda in piše na standardni izhod. Za preusmeritev v celoti poskrbi lupina `bash`.

Pogosto nam pride prav veriženje vhodov in izhodov. Če ukaza povežemo z operatorjem `|` (*cev*), potem izhod prvega ukaza preusmerimo na vhod drugega ukaza. Na primer, ukaz `cat` izpiše vsebino podane datoteke na standardni izhod, ukaz `wc -l` pa izpiše število vrstic na standardnem vhodu. Če ukaza povežemo s cevjo ...

```
cat Primer.java | wc -l
```

... potem ukaz `cat` svojega izhoda (vsebine datoteke `Primer.java`) ne bo izpisal na zaslon, ampak ga bo posredoval ukazu `wc` kot vhod. Ta ukaz bo svoj izhod (število vrstic datoteke `Primer.java`) izpisal na zaslon.

Tvorimo lahko tudi daljše verige. Na primer, ukaz

```
cat Primer.java | grep 'public' | wc -l
```

izpiše število vrstic v datoteki `Primer.java`, ki vsebujejo podniz `public`. Ukaz `grep` take vrstice izpiše na standardni izhod, ta pa se posreduje ukazu `wc -l`, ki vrstice prešteje in rezultat zapiše na standardni izhod.

Izraz `$(klic)` se nadomesti s standardnim izhodom klica *klic*. Izhod klica lahko tako uporabimo kot sestavni del drugega klica. Na primer, če se prestavimo v imenik `~/faks` in vanj dodamo datoteko `brisi.txt` z vsebino

```
Prva.java Druga.java Tretja.java
```

potem klic

```
rm $(cat brisi.txt)
```

pobriše vse tri datoteke s končnico `.java`. Izraz `$(cat brisi.txt)` se namreč nadomesti z izhodom klica `cat brisi.txt`, torej s seznamom `Prva.java Druga.java Tretja.java`. Celoten klic se tako nadomesti z

```
rm Prva.java Druga.java Tretja.java
```

## 9 Narekovaji

Nekateri znaki, denimo `*` in `?`, imajo v lupini `bash` poseben pomen. Če, na primer, v imeniku `~/faks` izvršimo klic

```
echo *.java
```

ga bo lupina takoj nadomestila z

```
echo Prva.java Druga.java Tretja.java
```

Ker ukaz `echo` enostavno izpiše svoje argumente na standardni izhod, bo gornji klic proizvedel izpis

```
Prva.java Druga.java Tretja.java
```

Z uporabo enojnih ali dvojnih narekovajev pa posebni znaki izgubijo svoj pomen. Klica

```
echo "/*.java"
```

in

```
echo '/*.java'
```

bosta torej izpisala niz `/*.java`.

Pod enojnimi narekovaji svoj posebni pomen izgubijo vsi znaki, pod dvojnimi pa znak `$` ohrani svojo funkcijo. Na primer, klica `echo $(ls)` in `echo "$ (ls)"` izpišeta vsebino trenutnega imenika, klic `echo '$ (ls)'` pa izpiše dobesedno `$(ls)`.

Če želimo posebni pomen odstraniti samo enemu znaku, lahko pred ta znak vstavimo znak `\`. Na primer, klic `echo \*.java` izpiše `/*.java`, klic `echo \$\ (ls\)` pa `$(ls)` (tudi oklepaji imajo v `bashu` posebni pomen).

## 10 Lastništvo datotek in imenikov in pravice dostopa

Unixovski sistemi so po svoji zasnovi večuporabniški, tudi če jih uporablja ena sama oseba. Vsak uporabnik pripada vsaj eni *skupini*. Uporabnik osebnega računalnika tipično pripada sam svoji skupini (npr. uporabnik *miha* pripada skupini *miha*, ki razen njega ne vsebuje nikogar drugega), na večuporabniških sistemih pa nam pogosto pride prav, da lahko uporabnike razvrščamo v skupine.

Vsak element datotečnega sistema pripada nekemu uporabniku (to je *lastnik* elementa) in eni od skupin, v katero je lastnik včlanjen. Lastnika in skupino, ki jima pripada element, lahko razberemo iz izpisa ukaza `ls -l`. Na primer, izpis

```
-rw-rw-r-- 1 jana studentje 5400 jun 15 06:02 porocilo.tex
```

pove, da datoteka `porocilo.tex` pripada uporabnici z uporabniškim imenom `jana` in skupini `studentje`. Uporabnica `jana` mora biti članica skupine `studentje`, lahko pa pripada tudi kateri drugi skupini.

Lastništvo datoteke ali imenika lahko spremenimo z ukazom `chown`:

```
chown [-R] uporabnik[:skupina] poti
```

Na primer, klic

```
chown mojca:administratorji porocilo.tex
```

dodeli lastništvo datoteke `porocilo.tex` uporabnici z uporabniškim imenom `mojca` in skupini `administratorji`. Ukaz `chown` moramo praviloma pognati s predpono `sudo`; navadni uporabniki imajo precej omejene možnosti spreminjanja lastništev. Stikalo `-R` zamenja lastništvo vsem elementom znotraj poddrevesa, zato pred njegovo uporabo dobro premislimo.

Zaporedje znakov od drugega do desetega v posamezni vrstici izpisa ukaza `ls -l` (v gornjem primeru je to `rw-rw-r--`) podaja *pravice dostopa* do datoteke. Zaporedje je razdeljeno na tri trojice. Prva trojica (`rw-`) se nanaša na lastnika, druga (`rw-`) na skupino, tretja (`r--`) pa na ostale navadne uporabnike. Lastnik lahko datoteko bere (`r`) ali spreminja (`w`), ne more pa je izvajati (na tretjem mestu je znak `-` namesto znaka `x`). Skupina ima enake pravice kot lastnik, ostali navadni uporabniki pa lahko datoteko samo berejo. Pravice dostopa ne veljajo za superuporabnika; ta lahko s poljubno datoteko in imenikom počne, karkoli želi.

Pri imenikih so pomeni pravic `r`, `w` in `x` nekoliko drugačni: `r` pomeni pravico do pregledovanja imenika (npr. z ukazom `ls`), `w` pomeni pravico do spreminjanja vsebine imenika (dodajanja in brisanja datotek), `x` pa pravico do vstopa v imenik z ukazom `cd`.

Pravice dostopa lahko spreminjamo z ukazom `chmod`. Ta ukaz lahko uporabljamo na več načinov:

- Stikalo *+pravica* oziroma *-pravica* vsem uporabnikom dodeli oziroma odvzame podano pravico (`r`, `w` ali `x`). Na primer, klic

```
chmod +x porocilo.tex
```

bi datoteko `porocilo.tex` naredil izvršljivo za vse uporabnike. (Če datoteka `porocilo.tex` ni neposredno izvršljiv program, potem ta klic nima smisla.) Klic

```
chmod -w porocilo.tex
```

pa vsem uporabnikom odvzame pravico do spreminjanja datoteke `porocilo.tex`.

- Stikalo *kdo+pravica* oziroma *kdo-pravica* dodeli oziroma odvzame podano pravico samo podani skupini uporabnikov (`u` – lastnik, `g` – skupina, `o` – ostali). Na primer, klic

```
chmod g-w porocilo.tex
```

bi skupini odvzel pravico do spreminjanja datoteke `porocilo.tex`.

- Če želimo spremeniti celoten niz pravic naenkrat, je to najlažje storiti s parametrom *ugo*, kjer so *u*, *g* in *o* števila med 0 in 7. Število *u* se nanaša na lastnika, število *g* na skupino, število *o* pa na ostale uporabnike. Vrednosti števil *u*, *g* in *o* imajo pomen:

Število	0	1	2	3	4	5	6	7
Pomen	---	--x	-w-	-wx	r--	r-x	rw-	rwX

Vrednost 0 pomeni odsotnost vseh pravic, vrednost 1 prisotnost zgolj pravice `x` itd. Na primer, klic

```
chmod 640 porocilo.tex
```

dodeli lastniku pravico do branja in spreminjanja datoteke `porocilo.tex`, skupini dodeli samo pravico do branja, ostalim uporabnikom pa odvzame vse pravice.



## 11 Programi in procesi

*Program* je datoteka z zaporedjem navodil, ki jih računalnik ali nek drug izvajalni sistem (npr. lupina `bash`) lahko neposredno izvede. Če želimo program zagnati, mora biti izvršljiv; uporabnik sistema mora na datoteki imeti dostopno pravico `x`.

Datoteko s programom poženemo enostavno tako, da vnesemo njeno absolutno ali relativno pot. Obstaja pa pomembna izjema: če želimo pognati program (npr. `sestej`) znotraj trenutnega imenika, ni dovolj, da odtipkamo samo `sestej` (kar je najenostavnejša relativna pot do programa), ampak ga moramo pognati takole:

```
./sestej
```

Da bomo lahko razumeli potrebo po takšnem ovinku, moramo poznati nekaj ozadja. V Linuxu in številnih drugih sistemih obstaja sistemska spremenljivka `PATH`. Če s klicem `echo $PATH` izpišemo njeno vsebino, dobimo seznam poti do imenikov, ločenih z dvopičjem. Spremenljivka `PATH` se uporabi, ko poskusimo pognati program samo z navedbo njegovega imena (npr. `sestej` namesto `./sestej`). V takem primeru sistem prične po vrsti pregledovati imenike v spremenljivki `PATH`. Ko najde prvi imenik, ki vsebuje izvršljiv program z odtipkanim imenom, bo ta program izvršil. Na ta način lahko lupina izvrši, denimo, program `ls`, čeprav ga praviloma nimamo v trenutnem imeniku. Program `ls` se namreč nahaja v imeniku `/bin`, ki ga najdemo v spremenljivki `PATH`. Klic `sestej` bi torej uspel samo v primeru, če bi se program `sestej` nahajal v katerem od imenikov, navedenih v spremenljivki `PATH`.

Program v fazi izvajanja se imenuje *proces*. Proces obstaja od trenutka, ko se program zažene, do trenutka, ko se zaključi. Seznam vseh procesov izpišemo s klicem

```
ps -e
```

Vsak proces je enolično določen s številko *PID* (angl. process ID). S klicem

```
kill -9 PID
```

lahko proces s številko *PID* nasilno zaključimo. Sliši se krvoločno, a včasih žal ne moremo drugače ... Ukaz `kill` sicer nima najbolj posrečenega imena, saj je namenjen pošiljanju poljubnih signalov procesom. Le signal s številko 9 neposredno povzroči smrt procesa.

## 12 Razna orodja

Lupina `bash` postane resnično močna šele z orodji. V tem razdelku jih bomo nekaj navedli. Pri vsakem bomo prikazali le tipično uporabo; število vseh možnih parametrov je praviloma še precej večje.

- `echo arg1 arg2 ...`

Po vrsti izpiše svoje argumente.

- `head [-n] [datoteka]`

Izpiše prvih *n* vrstic podane datoteke. Če datoteka ni podana, izpiše prvih *n* vrstic besedila na standardnem vhodu. Ukaz

```
head -5 primer.txt
```

je potemtakem enakovreden ukazu

```
head -5 < primer.txt
```

in tudi ukazu

```
cat primer.txt | head -5
```

V bashu nam veliko ukazov ponuja obe možnosti: datoteko, na kateri naj ukaz deluje, lahko podamo kot argument ukaza, če argument izpustimo, pa bo ukaz deloval nad standardnim vhodom. Če standardni vhod preusmerimo na datoteko, bo ukaz deloval na datoteki, ne da bi se tega zavedal.

- **tail** *[-n]* *[datoteka]*

Deluje podobno kot **head**, le da izpiše zadnjih *n* vrstic podane datoteke oziroma besedila na standardnem vhodu.

- **wc** *[-c]* *[-w]* *[-l]* *[-L]* *[datoteka]*

Izpiše število znakov (*-c*), število besed (*-w*), število vrstic (*-l*) oziroma dolžino najdaljše vrstice (*-L*) v podani datoteki ali na standardnem vhodu.

- **tr** *znak<sub>1</sub> znak<sub>2</sub>*  
**tr** *-s znak*

Prva oblika zamenja vse pojavitve znaka *znak<sub>1</sub>* z znakom *znak<sub>2</sub>*, druga pa nadomesti vsa zaporedja znaka *znak* z enim samim znakom *znak*. Ukaz bere standardni vhod in piše na standardni izhod. Na primer, klic

```
cat primer.txt | tr ' ' '\n' | tr -s '\n' > rezultat.txt
```

prepiše vsako besedo datoteke **primer.txt** v svojo vrstico in nastalo besedilo shrani v datoteko **rezultat.txt**. Najprej se vsi presledki zamenjajo s prelomi vrstic (zaporedje *'\n'* predstavlja znak za prelom vrstice), nato pa se morebitni večkratni prelomi vrstic zamenjajo z enim samim prelomom.

- **cut** *-c p[-q]* *[datoteka]*  
**cut** *[-d znak] -f p[-q]* *[datoteka]*

Prva oblika iz vsake vrstice podane datoteke ali standardnega vhoda izlušči znake od *p*-tega do vključno *q*-tega. Druga oblika pa izlušči stolpce od *p*-tega do vključno *q*-tega, pri čemer so stolpci ločeni z znakom *znak*. Na primer, datoteka **ocene.csv** na sliki 2 vsebuje tri stolpce, ločene s podpičjem, zato bi klic

```
cat ocene.csv | cut -d ';' -f 2
```

izpisal vse priimke oseb.

- **paste** *[-d znak]* *datoteka...*

Izpiše vse datoteke eno ob drugi (prva datoteka postane prvi stolpec, druga drugi itd.), pri čemer kot ločilo med tako nastalimi stolpci uporabi znak *znak*.

- **grep** *vzorec* *[datoteka...]*

Izpiše tiste vrstice podane datoteke oziroma standardnega vhoda, ki vsebujejo podniz, skladen z regularnim izrazom *vzorec*. Na primer, klic

```
grep ';' '$' ocene.csv
```

izpiše vse vrstice datoteke **ocene.csv**, ki se končajo (\$) s podpičjem in poljubnim znakom (.), torej vse študente z enomestno oceno.

Jože;Gorišek;75
Ana;Bevc;80
Peter;Gorenc;67
Marko;Debeljak;93
Maja;Petrič;33
Ivan;Kavčič;75
Janez;Gorenc;9
Petra;Malik;96
Bojan;Gorenc;56
Eva;Gorenc;64
Ana;Gorenc;89

Slika 2: Vsebina datoteke `ocene.csv`.

Regularni izrazi so preobširni, da bi jih obravnavali na tem mestu. Bralec si lahko njihovo sintakso in pomen ogleda s klicem `man grep`.

- `sed s/poisci/zamenjaj/g`

Ukaz `sed` (angl. stream editor) bi si zaslužil vsaj nekaj strani, najpogosteje pa se uporablja za zamenjevanje besedila na standardnem vhodu. V obliki, ki smo jo navedli, ukaz `sed` v vseh vrsticah zamenja vse podnize, skladne z regularnim izrazom `poisci`, z nizom `zamenjaj`. Na primer, klic

```
sed s/a/b/g < ocene.csv
```

zamenja vse pojavitve črke `a` v datoteki `ocene.csv` s črko `b` in rezultat izpiše na standardni izhod, klic

```
sed 's/^\([[:alnum:]]*\); \([[:alnum:]]*\); /\2; \1; /g' < ocene.csv
```

pa med seboj zamenja prvi in drugi stolpec datoteke `ocene.csv` (ime in priimek) in rezultat izpiše na standardni izhod. Znak `^` se nanaša na začetek vrstice, izraz `[[:alnum:]]*` pa na poljubno zaporedje alfanumeričnih znakov (črk in števk). Zapis `\1` se nanaša na prvi, zapis `\2` pa na drugi izraz, obdan s parom oklepajev `\(` in `\)`.

- `sort [-t ločilo] [-r] [-k ključ]... [-n] [datoteka]`

Uredi vrstice datoteke oziroma standardnega vhoda v naraščajočem leksikografskem vrstnem redu. Stikalo `-r` obrne vrstni red. Stikali `-t` in `-k` se uporabljata za urejanje glede na posamezne stolpce; stikalo `-t` podaja znak, s katerim so stolpci med seboj ločeni, stikala `-k` pa stolpce, po katerih naj se vhod uredi. Stikalo `-n` ima smisel samo, če vhod oziroma stolpec vsebuje izključno števila, pomeni pa numerično (namesto leksikografsko) urejanje.

V primeru datoteke `ocene.csv` s slike 2 bi klic

```
sort -t ';' -k '3,3' -n -r ocene.csv
```

uredil datoteko po padajočih ocenah (levi del slike 3), klic

```
sort -t ';' -k '2,2' -k '1,1' ocene.csv
```

pa bi jo uredil primarno po priimkih, v primeru enakih priimkov pa še po imenih (desni del slike 3). Stolpci so med seboj ločeni s podpičjem, parameter `'i,i'` pa predstavlja *i*-ti stolpec (»od *i*-tega do *i*-tega stolpca«).

- `uniq [datoteka]`

Petra;Malik;96	Ana;Bevc;80
Marko;Debeljak;93	Marko;Debeljak;93
Ana;Gorenc;89	Ana;Gorenc;89
Ana;Bevc;80	Bojan;Gorenc;56
Jože;Gorišek;75	Eva;Gorenc;64
Ivan;Kavčič;75	Janez;Gorenc;9
Peter;Gorenc;67	Peter;Gorenc;67
Eva;Gorenc;64	Jože;Gorišek;75
Bojan;Gorenc;56	Ivan;Kavčič;75
Maja;Petrič;33	Petra;Malik;96
Janez;Gorenc;9	Maja;Petrič;33

Slika 3: Levo: vrstice datoteke *ocene.csv*, padajoče urejene po točkah. Desno: iste vrstice, urejene primarno po priimkih in sekundarno po imenih.

Odstrani vse zaporedne podvojene vrstice datoteke oziroma standardnega vhoda. Ta ukaz se pogosto uporablja skupaj z ukazom `sort`. Na primer, če datoteka `imena.txt` vsebuje imena Ivan, Eva, Mojca, Eva, Ivan, Mojca in Ivan (vsako v svoji vrstici), bo klic

```
cat imena.txt | sort | uniq
```

izpisal imena Eva, Ivan in Mojca (seveda spet vsako v svoji vrstici).

Kako bi ugotovili število različnih besed v datoteki `esej.txt`, če so besede med seboj ločene samo s presledki in prelomi vrstic? Takole:

```
cat esej.txt | tr ' ' '\n' | tr -s '\n' | sort | uniq | wc -l
```

- `find imenik [-name 'vzorec']`

V poddrevesu podanega imenika poišče vse datoteke, katerih imena se skladajo s podanim vzorcem, in izpiše poti do njih. Na primer, klic

```
find . -name '*.java'
```

izpiše relativne poti vseh datotek s končnico `.java` v poddrevesu trenutnega imenika. Vzorec `*.java` moramo podati znotraj enojnih ali dvojnih narekovajev, saj bi ga sicer lupina še pred zagonom ukaza `find` zamenjala s seznamom datotek s končnico `.java` v trenutnem imeniku.

Kako v poddrevesu imenika `~/faks/razno` izbrisemo vse datoteke s končnico `.class`? Takole:

```
rm $(find ~/faks/razno -name '*.class')
```

Kot že vemo, se celoten izraz `$(...)` nadomesti z izhodom ukaza `find`. To pomeni, da bo seznam poti do datotek s končnico `.class` postal seznam parametrov ukaza `rm`.

**Pozor!** Verjetno ni treba posebej opominjati, da je takšno brisanje datotek potencialno nevarno. Najprej izvedimo ukaz `find`, nato seznam podrobno preglejmo, in šele ko smo prepričani, da res želimo izbrisati vse datoteke na seznamu, dodamo še dolar, par oklepajev in ukaz `rm`.

Ukaz `find` ponuja bogat nabor možnosti, saj lahko datoteke iščemo tudi po uporabniku, času zadnje spremembe itd. Na primer, sledeči klic prešteje vse datoteke

lastnika *miha* v poddrevesu imenika */home*:

```
find /home -user miha | wc -l
```

- **bc**

Zažene preprost kalkulator. Ta ukaz se pogosto uporablja v kombinaciji z ukazom *echo*. Na primer, klic

```
echo '1 + 2' | bc
```

pošlje izraz  $1 + 2$  na vhod programa *bc*, ta pa na standardni izhod izpiše rezultat, torej 3.

Sledeča veriga izpiše skupno vsoto velikosti datotek v trenutnem imeniku:

```
echo 0$(ls -l | tr -s ' ' | cut -f 5 -d ' ' | tr '\n' '+')0 | bc
```

Pojdimo po vrsti. Klic *ls -l* izpiše seznam datotek v trenutnem imeniku. Klic *tr -s ' '* zamenja vsa zaporedja presledkov z enim samim presledkom. Stolpci so sedaj ločeni z enim presledkom; klic *cut -f5 -d ' '* tako izlušči peti stolpec, ki vsebuje velikosti datotek. Klic *tr '\n' '+'* nato zamenja prelome vrstic z znaki *+*. Na tem mestu dobimo izhod

```
+v1+v2+. . .+vk+
```

pri čemer so  $v_1, \dots, v_k$  velikosti posameznih datotek. Tega izraza še ne moremo posredovati programu *bc*, saj bi ga zmotila začetni in končni plus. Lahko pa izhod na obeh straneh obdamo z ničlo (da dobimo  $0+v_1+. . .+v_k+0$ ), dobljeni izraz podamo ukazu *echo* kot parameter (v ta namen si pomagamo z dolarjem in parom oklepajev) in vse skupaj posredujemo na vhod ukaza *bc*.

## 13 Programiranje

V lupini *bash* lahko pišemo prave pravcate skripte (programe). O bashevskih programskih konstrukcijah bi lahko napisali vsaj krepko poglavje v knjigi, na tem mestu bomo predstavili samo osnovno obliko zanke *for*. Skripto zapišemo v datoteko, nato pa datoteko opredelimo kot izvršljivo in jo poženemo. Recimo, če smo skripto zapisali v datoteko *skripta.sh* znotraj trenutnega imenika, jo poženemo takole:

```
chmod +x skripta.sh
./skripta.sh
```

Ukaz *chmod* seveda izvršimo samo enkrat.

Osnovna oblika zanke *for* izgleda takole:

```
for spremenljivka in besede; do
    ukazi
done
```

Telo zanke (*ukazi*) se izvrši tolikokrat, kolikor je besed v seznamu *besede*. V prvem obhodu se spremenljivki priredi prva beseda s seznama, v drugem druga beseda itd. Vrednost spremenljivke pridobimo z izrazom *\$spremenljivka*. Na primer, skripta

```
#!/bin/bash
for dat in $(ls); do
```

```
    echo $dat
done
```

izpiše vse datoteke v trenutnem imeniku. Izraz `$(ls)` se nadomesti z izhodom ukaza `ls` (seznamom datotek v trenutnem imeniku). Vrednost spremenljivke `dat` bo v prvem obhodu zanke torej enaka imenu prve datoteke, v drugem obhodu imenu druge datoteke itd. V vsakem obhodu zanke se izpiše trenutna vrednost spremenljivke `dat`, torej trenutno ime datoteke.

Vrstica `#!/bin/bash` pove lupini, da gre za skripto lupine bash, ne pa, recimo, za pythonovo skripto. Tako bo lupina vedela, s katerim programom naj skripto zažene. V tem primeru bo skripto zagnala sama. Pred vrstico `#!/bin/bash` ne sme biti ničesar, niti prazne vrstice!

Gornja skripta seveda nima pravega smisla, saj bi isti rezultat lahko dobili kar z ukazom `ls`. V sledeči skripti, ki za vsako datoteko s končnico `.txt` v poddrevesu trenutnega imenika izdela rezervno kopijo s končnico `.txt.bak`, pa bi brez zanke težko shajali:

```
#!/bin/bash
for dat in $(find . -name '*.txt'); do
    cp $dat $dat.bak
done
```