Turinys

Komandų susiejimas

Žemiau pateiktų komandų operatoriai gali būti naudojami ir ne po 1 arba skirtingų tipų.

Operatorius	Sintaksė	Aprašymas	Pavyzdys
;	command1; command2	Komandos yra įvykdomos nuosekliai.	Pavyzdžiui turime: **date ; pwd** **date** komanda bus įvykdoma pirmoji ir tada bus vykdoma sekanti komanda **pwd.**
&	command arg &	Shell vykdo komandą antrajame plane (angl. background) & - operatorius paleidžia komandą antrajame plane, taigi terminalo langas yra atlaisvintas kitų komandų vykdymui	Komanda find yra įvykdoma antrajame plane, galiausiai yra išvedami surastų pdf failų pavadinimai į output.txt find / -iname "*.pdf" >/tmp/output.txt &
&&	command1 && command2	command2 yra vykdoma jeigu ir tik tada jeigu komanda command1 gražina užbaigimo būseną (angl. exit) gražina 0. Pvz: command2 yra paleidžiama jeigu command1 sėkmingai paleista.	[! -d /backup] && mkdir -p /backup Loginis IR (angl. AND)
||	command1 || command2	command2 yra įvykdoma jeigu command1 gražina ne nulį. Command2 paleidžiama jeigu command1 nepavyko įvykdyti.	tar cvf /dev/st0 /home || mail -s 'Backup failed' you@example.com </dev/null

			Loginis arba (angl. OR)
|	command1 | command2	Linux „shell pipes“ (apvalkalo konvejeriai) apjungia standartinės išvesties (angl. output) **komanda-pirma** rezultatą perduodama komandos **komanda-antra** įvesčiai (angl. input)	Komandos ps išvestis yra perduodama komandai grep kaip įvestis: **ps** aux | **grep** httpd

Sudėtinės komandos-operatorius ;

Galima sukurti seką komandų naudojant šį operatorių ; kurio sintaksė yra:

command1 ; command2 ; commandN

ARBA

{ command1; command2 }

Šiuo būdu galima paleisti komandas viena po kitos. Žemiau pateiktame pavyzdyje, shell skriptas atvaizduoja klaidos pranešimą jeigu komandinės eilutės argumentai netinkamo formato pagal math.sh skriptą:

```
#!/bin/bash

a=$1

b=$3

op=$2

ans=0

# display usage

# run commands one after the other using ; chracter

[ $# -eq 0 ] && { echo -e "Usage: $0 num1 op num2\n\t $0 1 + 5"; exit 1; }


case $op in
```

```
        +)
                ans=$(( a+b ));;
        -)
                ans=$(( a-b ));;
        /)
                ans=$(( a/b ));;
        \*|x)
                ans=$(( a*b ));;
        *)
                echo "Unknown operator."
                exit 2;;
esac
echo "$a $op $b = $ans"
```

Išsaugojus ši failą įvykdyti šiuos žingsnius, kad paleisti math.sh scenarijų:

```
chmod +x math.sh
./math.sh
./math.sh 1 + 5
./math.sh 10 \* 5
```

Be šio operatoriaus `;` taipogi galima naudoti ir kitą operatorių `&&` kuris apjungia keletą komandų vienoje eilutėje:

```
[ $# -eq 0 ] && { echo -e "Usage: $0 num1 op num2\n\t $0 1 + 5"; exit 1; }
```

Analogiškos komandos užrašas nepritaikius operatoriaus `&&`:

```
if [ $# -eq 0 ]
then
   echo -e "Usage: $0 num1 op num2\n\t $0 1 + 5"
   exit 1;
fi
```

Laikinojo failo stebėsena

Naudojant watch komandą skirtą stebėti /tmp failą kas 5 sekundes:

```
watch -n 5 'df /tmp; ls -lASFt /tmp'
```

Užduočių perkėlimas į antrąjį planą

Linux palaiko kelių procesų vykdymą lygiagrečiai arba paeiliui.

Daugelis linux komandų tokios kaip failų redagavimas, laiko ir datos nustatymai, prisijungti prie vartotojo paskyros ir kt. nustatymai gali būti atlikti kelias būdais, taikant įvairiausias Linux komandas.

Rašant komandas terminalo lange (shell. Prompt). Šios programos izoliuoja jūsų ekraną šiuo atveju ekranas tai terminalo langas, ir kai baigiame vykdyti komandų seką, galiausiai sugrįžtame į pradinę būsena, sekančių komandų vykdymui.

Kartais norime, kad mūsų vykdomos komandos terminalo lango neperimtu ir būtų vykdomos antrajame plane (angl. background). Pavyzdžiui norime surasti visų mp3 formato failus esančius diske, kol mes rašome programą c programavimo kalba ir panašiai.

Darbų kontrolė

Bash shell'as leidžia paleisti užduotis ar komandas antrajame plane naudojant **job control** priemonę.

Job control turi galimybes: pasirinktino proceso sustabdymas, proceso vykdymo užlaikymas ir proceso pratęsimas vėliau.

Jobs

Procesai job control'e yra vadinami jobs.

Kiekvienas procesas (job) turi unikalų id, kuris yra vadinamas job numeris.

Background process (Antrojo plano procesas)

Komandai kuriai yra sudarytos instrukcijos ir yra vykdoma nenuosekliai yra vadinama antrojo plano procesas (angl. background process).

Antrojo plano procesai nėra matomi ekrane. Vienas iš pavyzdžių Apache httpd serveris yra paleidžiamas antrajame plane, apdoroti internetinius puslapius. Taip pat bet kuris shell scenarijus ar komanda gali būti vykdoma antrajame plane.

Foreground process (Priekinio plano procesas)

Komanda kurią įvykdžius galima matyti komandos vykdymo seką / turinį ekrane yra vadinamas priekinio plano procesu (angl. foreground process).

Kaip aš galiu vykdyti komandas antrajame plane?

Žemiau yra pateikta sintaksė skirta nurodyti, kad šis procesas būtu vykdomas antrajame plane:

```
command &

command arg1 arg2 &

command1 | command2 arg1 &

command1 | command2 arg1 > output &
```

Operatorius **&** nukreipia komadą veikti antrajame plane ir yra atlaisvinamas terminalo langas.

Komanda, kuri yra paleidžiama antrajame plane yra vadinama job.

Taip pat galima įvykdyti kitas komandas kol antrojo plano komanda yra paleista.

Pavyzdys

Pavyzdžiui įvykdžius šią komandą su operatoriumi & nurodo jog ši komanda bus vykdoma antrajame plane:

```
find /nas -name "*.mp3" > /tmp/filelist.txt &
```

Komandos išvestis:

```
[1] 1307
```

Komanda **find** yra paleidžiama antrajame plane. Kai bash startuoja naują job antrajame plane yra išvedama eilutė rodanti job numerį [1] ir proceso identifikacijos numeris (PID – 1307). Job siunčia žinutę terminalo lange atvaizduoja job numerį ir kad darbas yra atliktas pagal jo id:

```
 [1]+  Done                    find /share/ -name "*.mp3" >
/tmp/filelist
```

Linux ir UNIX job control komandų suvestinė

Komanda	Aprašymas	Pavyzdžiai
&	Nurodo jog užduotis(angl. job) bus vykdoma antrajame plane.	command &
%n	Priskiria užduočiai jūsų nurodytą užduoties numerį n.	command %1
%Word	Kreiptis į užduotis, kurios prasideda žodžio šaknimi.	command %yum
%?Word	Kreiptis į visas užduotis atitinkančias konkretų žodį.	command %?ping
%% **%+**	Kreiptis į einamą užduotį.	kill %% kill %+
%-	Kreiptis į praėjusią užduotį.	bg %-
CTRL-Z **kill -s stop jobID**	Užlaikyti arba sustabdyti užduotį.	kill -s stop %ping
jobs **jobs -l**	Atspausdinti visų užduočių sąrašą.	jobs -l
bg	Perkelti užduotis į antrąjį planą.	bg %1 bg %ping
fg	Perkelti užduotį į priekinį planą.	fg %2 fg %apt-get

Unix / Linux PIPES kas tai ?

Shell konvejeris (angl. pipe) yra būdas apjungti vienos programos išvesti perduodant kitai programai kaip įvestį nesukuriant laikinojo failo.

Sintaksė:

```
command1 | command2

command1 | command2 | commandN

command1 arg1 | command2 arg1 arg2

get_data_command | verify_data_command | process_data_command | format_data_command > output.data.file

get_data_command < input.data.file | verify_data_command | process_data_command | format_data_command > output.data.file
```

Kaip matote aukščiau pateiktuose pavyzdžiuose jūs gali sujungti 2 komandas arba daugiau vienu metu.

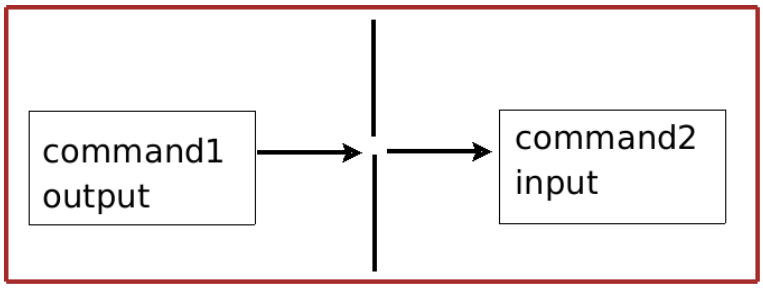
Duomenų kelias, kuris apjungia 2 programas yra vadinmas **konvejeris arba (angl. pipe).**

Vertikalus ir/arba Ortogonalus brūkšnys | yra konvejerio simbolis.

Shell konvejeris palaiko UNIX filosofija programų apjungimas į grandinę atlikti sudėtingoms užduotims.

Nukreipiamieji operatoriai taip pat yra leistini linux konvejeriuose.

Duomenų kelias yra vienakryptis.



Linux / Unix konvejerio (angl. pipe) 2 komandų atvaizdavimas grafiškai.

Operatorius |.arba UNIX / Linux konvejeriai (angl. pipes)

10 naujausiai sukurtų failų atspausdinimas

Šiame pavyzdyje **ls -lt** komandos išvestis (angl. output) yra perduodama kitai komandai **head**, kuri atspausdina 10 naujausiai sukurtų failų einamojoje direktorijoje.

```
ls -lt | head
```

Konkretaus failo egzistavimo nustatymas

Įvykdžius komandą **ls** yra perduodama šios komandos išvestis komandai **grep** „data.txt" ir jeigu nurodytas failas „data.txt" egzistuoja tada rezultatas yra išvedamas komandinėje eilutėje.

```
ls | grep „data.txt"
```

Operatoriaus | panaudojimo pastabos

Tai pat operatorius **|** skiriantis komandas gali neturėti tarpo simbolio tarp komandų pasirinktinai pagal situaciją jeigu komandų aibė nėra labai plati:

```
ls -al|grep „data.txt"
```

Pastaba! Yra rekomenduojama komandas atskirti tarpo simboliais, pagerinti aprašytųjų komandų skaitomumui, kadangi gali būti situacija, kai komandų sąrašas platus.

Komanda **ps** išveda einamojo proceso informaciją, dažnai tenka informaciją talpinti failuose taigi, kad tai atlikti **grep httpd** reikia nurodyti nukreipiamąjį operatorių **>** ir failą į kurį talpinsime informaciją.

```
ps aux | grep httpd > /tmp/ps.output.log
```

Komandos ls išvesties sustabdymas

Siunčiamas komandos ls išvestis komandai more kaip įvestis, taigi išvestis bus pateikiama būtent pilnai užpildyto lango tekstine informacija. Pavyzdžiui langas yra 20 eilučių ilgio taigi jeigu išvestis yra perduoda komandai more 60 eilučių, bus atvaizduojamos tik 20 eilučių būtent užpildydamos pilną ekraną tekstine informacija, na ir kaip įprasta linux terminalo languose dažnai yra pateikiama ne pilnas tekstas ir norint jį peržiūrėti gale būna pridėtas užrašas „more", kad peržiūrėti daugiau informacijos.

```
ls -l | more
```

Prisijungusių vartotojų surūšiuotas sąrašas

Komandos **who** išvestis perduodama komandai sort kaip įvestis, surūšiuoja sąrašą ir tada galiausiai pritaikomas nukreipiamasis operatorius >, kuris nukreipią išvestį į sorted_list.txt (įrašo informaciją į šį failą).

```
who | sort
```

```
who | sort > sorted_list.txt
```

Apskaičiuoja prisijungusių vartotojų skaičių

Apskaičiuoja prisijungusius vartotojus.

```
who | wc -l
```

Suranda ar yra prisijungęs konkretus vartotojas

Tikrina ar yra prisijungęs vartotojas „vivek".

```
who | grep -i vivek
```

Apskaičiuoja failų skaičių esančių einamojoje direktorijoje

```
ls -l | wc  -l
```

Įvykdoma komanda kompiuterio išjungimui nurodytu laiku

```
echo "shutdown -h now" | at 12am tomorrow
```

Formatuojamas mount komandos išvedimas

Atvaizduoje mount komandos vizualiai gražesniu / patogesniu formatu.

```
mount | column -t
```

Kopija (angl. Backup) tar formatu naudojant ssh sesija

Use tar command over secure ssh session to backup local /home file system:

Naudojama tar komanda skirta saugiai ssh sesijai padaryti kopiją lokalioje direktorijoje /home.

```
tar zcvf - /home | ssh user@server "cat > /backup/home_fs.workstation_sep_21_09.tar.gz"
```

Raidžių konvertavimas iš mažųjų į didžiąsias ir atvirkščiai

```
v="Unix Philosophy"; echo $v | tr '[:lower:]' '[:upper:]'
```

```
echo 'tHIs IS A TeSt' | tr '[:upper:]' '[:lower:]'
```

Gimimo dienos proga el. pašto priminimas

```
echo "/usr/bin/mail -s 'Birthday gift for Julia' vivek@gite.in < /dev/null" | at 17:45
```

Sukuriamas ISO CD image failas

Sukuriamas ISO CD image failas iš nurodyto turinio esančio /home/vivek/photos direktorijoje.

```
mkisofs -V Photos -r /home/vivek/photos | gzip -9 > /tmp/photos.iso.cd.gz
```

Taip pat galima įrašyti į diskasukį (angl. cd-rom) nurodant ISO CD image failą:

```
gzip -dc /tmp/photos.iso.cd.gz | cdrecord -v dev=/dev/dvdrw -
```

Taip pat yra galimybė sukurti ISO CD image (failą) ir įrašyti tiesiai į nurodytą diskasukį (angl. cd-rom).

```
mkisofs -V Photos -r /home/vivek/photos | cdrecord -v dev=/dev/dvdrw -
```

Sukurti atsitiktinį slaptažodį

```
tr -dc A-Za-z0-9_ < /dev/urandom | head -c12 | xargs
```

Įvesties nukreipimo panaudojimas konvejeryje (angl. Pipe)

Įvestis < nukreipimas gali būti pritaikomas konvejeriuose gauti informaciją iš failo:

```
command1 < input.txt | command2
```

```
command1 < input.txt | command2 arg1 | command3
```

Pavyzdžiui, sort komanda gauna įvesties informaciją iš /etc/passwd failo, kurio rezultatas yra pritaikomas tolimesniai konvejerio komandai grep:

```
sort < input.txt | grep something
```

```
sort < input.txt | uniq | grep something
```

Išvesties panaudojimas konvejeryje

Kaip ir praėjusiose komandose standartinio išvedimo komandos gali naudoti nukreipiamuosius operatorius > arba >> žemiau esančiuose pavyzdžiuose yra pritaikomi nukreipiamieji operatoriai.

```
command1 | command2 > output.txt
```

```
command1 | command2 arg1 > output.txt

command1 < input.txt | command2 > output.txt

command1 < input.txt | command2 arg1 arg2 | command3 arg1 > output.txt
```

Pavyzdžiui, norime surūšiuoti visus atminties (angl. memory wise) procesus ir išsaugoti išvestį į failą memory.txt:

```
ps -e -orss=,args= | sort -b -k1,1n > memory.txt
```

Arba tiesiogiai nusiųsti į el. paštą pagal jūsų nurodytą paskyros adresą: vivek@gite.in:

```
ps -e -orss=,args= | sort -b -k1,1n | mail -s 'Memory process' vivek@gite.in
```

Kodėl yra naudojami konvejeriai?

Šiame pavyzdyje, mysqldump yra duomenų bazės kopijų sukūrimo programa pritaikyta duomenų bazei, kurios pavadinimas yra **wiki**:

```
mysqldump -u root -p'passWord' wiki > /tmp/wikidb.backup

gzip -9 /tmp/wikidb.backup

scp /tmp/wikidb.backup user@secure.backupserver.com:~/mysql
```

mysqldump komanda naudojama duomenų bazės wiki duomenų kopijos failo /tmp/wikidb.backup sukūrimui.

gzip komanda yra skirta suspausti (angl. compress) didelio duomenų bazės failui taip sutaupant nemažai standžiojo disko atminties.

scp komanda yra naudojama perkelti failą į serverio sub-domeną secure.backupserver.com .

Visos 3 komandos yra įvykdomos viena po kitos.

Laikinas failas yra sukuriamas lokaliame diske /tmp.

Tačiau, naudojant konvejerius galima apjungti standartinės išvesties **mysqldump** komandos ir perduoti kaip įvesti komandai **gzip** nesukuriant failo /tmp/wikidb.backup:

```
mysqldump -u root -p'passWord' wiki | gzip -9 > /tmp/wikidb.backup

scp /tmp/wikidb.backup user@secure.backupserver.com:~/mysql
```

Galima išvengti laikinojo failo sukūrimo paleidžiant visas komandas tuo pačiu metu:

```
mysqldump -u root -p'passWord' wiki | gzip -9 | ssh user@secure.backupserver.com "cat > /home/user/mysql/wikidb.gz" (1)
```

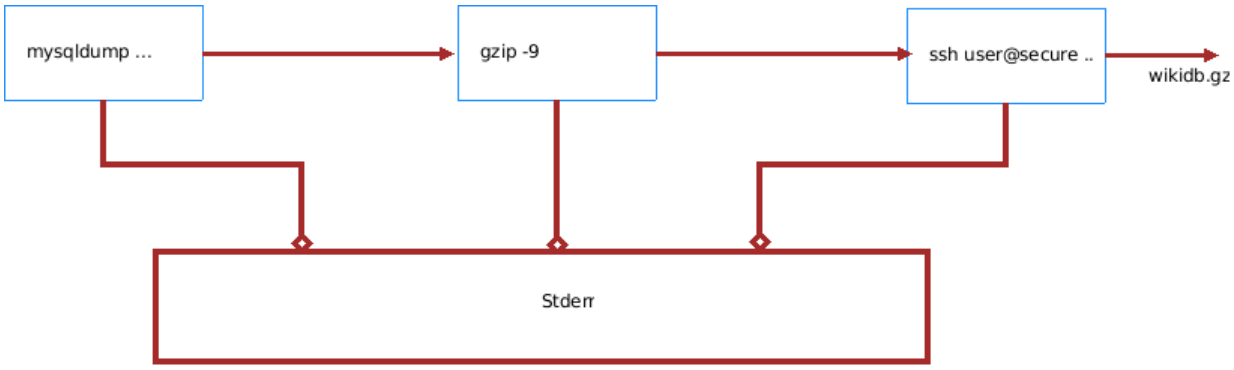
Aukščiau pateikta **(1)** sintaksė kompaktiškesnė ir lengvai pritaikoma (naudojama) kadangi yra apjungiamos kelios komandos iš karto, o nereikia įvykdyti 3 atskirų komandų:

Aukščiau pateiktame pavyzdyje yra įvykdomos 3 programos kartu **(1)**, skirtą įvykdyti sudėtingą užduotį, nuotoliniui „mysql" duomenų bazės, duomenų kopijų sudarymui, pritaikius konvejerius.

Duomenų filtravimas tai kita iš priežasčių kam yra naudojami konvejeriai.

Standartinis klaidų išvedimas yra pritaikytas naudojant konvejerius (žr. 1 schema).

(1)- pavyzdžio, konvejerinė programų vykdymo **1 schema.**



Jeigu Linux komanda priima įvesties duomenis iš standartinės įvesties ir išveda rezultatą standartinei išvesčiai tai yra vadinama filtru (angl. filter).

Filtrai paprastai veikia ir su konvejeriais

Sintaksė:

```
command1 | command2
```

```
command1 file.txt | command2
```

```
command1 args < input.txt | command2
```

Komanda command2 yra filtras command1.

Pavyzdžiai

Šiame pavyzdyje, grep komanda vaidina filtro vaidmenį. (yra išrenkamas vardas vivek iš šio filtro įvesties (angl. input).

```
cut -d: -f1 /etc/passwd | sort | uniq | grep vivek
```

Filtruojama ps komandos išvestis pritaikant grep komandą:

```
ps aux | grep php-cgi
```

Komanda **uniq** yra filtras, kuris paima įvestį iš **sort** komandos ir perduoda išvestį kaip įvestį komandai **uniq**; Tada **uniq** komanda išvestį nukreipia į „u_sname" failą. Komanda **grep** yra nutarta jog ji atlieką didžiausią vaidmenį ir yra iš dažniausiai naudojamo filtrų sąraše Linux ir UNIX operacinėse sistemose.

```
sort < sname | uniq > u_sname
```

Paprastai naudojamos filtrų komandos:

awk	cut	grep	gzip	head
paste	perl	sed	sort	split
strings	tac	tail	tee	tr
uniq	wc			

Filtrai egrep, fgrep, grep.

Filtravimo priemonės: **egrep**, **fgrep**, **grep**. Trumpai apie **awk** komandą ir jos struktūrą.

Kaip išrinkti įrašus apie Jonaitį iš visų jūsų turimų ***stud (studl,*** ...) failų?

Kaip išrinkti visus, gimusius 5-tą dieną?

Kaip išrinkti įrašus, kurių eilutės pradžioje būtų A, B, C, D raidės?

Kaip su ***awk*** komanda išrinkti tas įrašų eilutes, kurios tenkina nurodytą šabloną?

Kaip ***awk*** komandoje nurodyti tai, kad įrašų laukai turi nestandartinį skyriklį?

Kaip surūšiuoti ***awk*** programos rezultatus, neliečiant antraštinės eilutės?

Filtrai yra naudojami reikalingos informacijos išrinkimui iš duomenų srauto. Tam tarnauja ***egrep, fgrep, grep***. Jos leidžia skanuoti failą ir atmesti tas failo eilutes, kurios neatitinka šablonui. Visi šie filtrai iš esmės yra panašūs ir skiriasi tik naudojamais šablonais. Komandų sintaksė:

grep [-hilnvw] šablonas [*filename*]

fgrep [-hilnvwx] string [filename]

egrep [-hilnvw] šablonas [filename]

Komanda ***grep*** leidžia ieškoti sutapimo su šablonu nurodytame failų sąraše. Jei failai nenurodyti, tai paieška yra atliekama duomenų sraute, kuris eina iš standartinio įėjimo.

Eilutės, sutampančios su šablonu yra išvedamos į standartinį išvedimą. Jei yra nurodytas ne vienas failas, tai prieš eilutę, sutapusią su šablonu yra išvedamas failo vardas (jei nėra panaudota ***-h*** opcija).

Opcija ***-n*** liepia nurodyti eilutės numerį.

Opcija ***-i*** liepia ignoruoti šablonus.

Opcija ***-l*** liepia nurodyti sąrašą failų, kuriuose rastas sutapimas su šablonu.

Opcija ***-v*** liepia ***grep*** nurodyti sąrašą failų, kuriuose nerastas sutapimas su šablonu.

Opcija ***-w*** liepia apsiriboti tik tais sutapimais, kai pilnai sutampa visas žodis.

fgrep yra greitesnė ***grep*** komandos versija, kuri apsiriboja tik eilutės tipo šablonais. ***egrep*** yra išplėsta ***grep*** versija, kuri palaiko reguliarias išraiškas.

fgrep palaiko papildomą opciją ***-x,*** kuri liepia išvesti tik tas eilutes, kurios tiksliai sutampa su nurodyta seka.

Reguliariose išraiškose galima naudoti šiuos specialius ženklus:

. reiškia sutapimą su vienu bet kokiu simboliu,

******* reiškia sutapimą su nulinio ilgio arba bet kokio ilgio simbolių seka,

^ reiškia, kad paieška bus atliekama nuo eilutės pradžios,

$ reiškia, kad paieška bus atliekama eilutės gale,

[a-z] ieškos sutapimo su bent vienu simboliu iš nurodyto intervalo,

[abc] ieškos sutapimo su bent vienu iš nurodytų simbolių.

Komanda awk

Tai sudėtinga komanda, turinti programavimo kalboms būdingų bruožų ir elementų. Komanda "filtruoja" failų eilutes, kitaip tariant, ieško ir išskiria tas, kuriose atrandamas vartotojo užduotas šablonas. Komanda ***awk*** išskiria eilutėse laukus, kurie atskirti bent vienu tarpo ar tabuliacijos simboliu, ir gali atlikti šablonų palyginimo su tais laukais veiksmus. Laukai identifikuojami savo numeriais: ***$I.*** Čia ***I*** reikštų lauko numerį, o ***0*** žymi visą įrašo eilutę. Komanda ***awk*** iš įėjimo srauto galima išrinkti tenkinančias šabloną eilutes ir atlikti su jomis ar jų dalimis veiksmus. Šiuo atveju visi atliekami veiksmai yra įrašomi tarp kabučių.

Sintaksė:

```
$ awk pattern{action} [file]
```

Įvedimas yra suskirstytas į įrašus – eilutės ir laukus – simbolių grupės atskirtos tarpu arba TAB'u. Laukų skirtukus galima keisti su kintamuoju NF. Kintamasis $n žymi n-tąjį lauką, o $0 žymi visą įrašą. Eilutės BEGIN ir END žymi viso įvedimo pradžią ir pabaigą. Spausdinimas atliekamas su print ir formatuotu printf komandomis (kaip ir C kalboje). Ieškomą pavyzdį gali sudaryti keli reguliarūs išsireiškimai ir kombinuojami naudojant skirtukus: || - arba, && - ir, ! – ne. Kabutėmis atskirti išsireiškimai žymi paieškos pradžią ir pabaigą, pvz. /pirmas/,/paskutinis/. Norint parinkti eilutes nuo 15 iki 20 reikia rašyti: NR=15, NR=20. Atitikimas

reguliariam išsireiškimui yra išreiškiamas ~ - atitinka išsireiškimą, !~ - neatitinka išsireiškimo, pvz.:

$1 ~ /[Ll]abas/ programa yra true, jei pirmajame lauke bet kurioje vietoje yra žodis "labas". Jei pirmasis laukas turi turi tiesiogiai atitikti žodį 'Labas':

$1 ~ /^[Ll]abas$/

Kai kurios built-in funkcijos:

index(s,t) – grąžina t poziciją s eilutėje;

length(s) – grąžina s ilgį;

substr(s,m,n) – grąžina eilutės s (m,n) dalį.

Valdymo sakiniai (C stiliumi):

for(i=1;i<=$1;i++){ veiksmai }

while(i<=$1){ veiksmai }

if(i<10){ veiksmai }

Sisteminiai kintamieji:

FILENAME – failo vardas;

FS – lauko skirtukas;

NF – laukų skaičius įraše;

NR – einamojo įrašo numeris;

OFS – išvedimo lauko skirtukas;

ORS – išvedimo įrašų skirtukas;

$0 – visas įrašas;

$n – n-tasis laukas.

Pavyzdžiai:

- Atspausdina ilgiausios įvestos eilutės ilgį:

```
awk '{ if (length($0) > max) max = length($0) }
     END { print max }' data
```

- Atspausdinta visas eilutes, kurių ilgis yra didesnis, nei 80 simbolių:

  ```
  awk 'length($0) > 80' data
  ```

- Atspausdina ilgiausios eilutės ilgį iš duomenų srauto:

```
expand data | awk '{ if (x < length()) x = length() }
              END { print "maximum line length is " x
}'
```

- Atspausdina kiekvieną eilutę, kuri turi nors vieną lauką:

```
awk 'NF > 0' data
```

  Šis būdas yra naudingas tuo atveju jeigu mes norime pašalinti visas tuščias eilutes iš failo ir pavyzdžiui galime turinį perkelti į naują failą be tuščių eilučių.

- Atspausdins 7 atsitiktinius skaitmenis iš intervalo 0 to 100:

```
awk 'BEGIN { for (i = 1; i <= 7; i++)
                 print int(101 * rand()) }'
```

- Atspausdina sumą visų failų baitais:

```
ls -l files | awk '{ x += $5 }
                   END { print "total bytes: " x }'
```

- Atspausdina sumą visų failų kilobaitais:

```
ls -l files | awk '{ x += $5 }
   END { print "total K-bytes:", x / 1024 }'
```

- Atspausdina visų vartotojų prisijungimo vardų surūšiuotą sąrašą:

```
awk -F: '{ print $1 }' /etc/passwd | sort
```

- Suskaičiuoja eilutes faile:

```
awk 'END { print NR }' data
```

- Atspaudina lygines eilutes esančias duomenų faile:

```
awk 'NR % 2 == 0' data
```

Jeigu pakeisite išraišką ‘NR % 2 == 1’ programa išspausdins nelygines eilutes.

Užduotys (Atlikti šias komandas ir išsiaiškinti ką atliks)

```
$ awk {print $1}
```

```
$ awk /labas/ {print $0}
```

```
$ awk NF=2 {print $1+$2 }

$ awk BEGIN { FS="\n"; RS=""}

$ awk $1~/labas/ {print $3, $2}

$ awk –F:{printf("Eilutes Nr.%s suma yra %d\n",NR,$1+$2)}

$ awk /labas/ {++x}; END {print x}

$ awk {total+=$2}; END {print "viso",total}

$ awk length<20

$ awk NF=7 && /^Name:/

$ awk { for(i=NF;i>=1;i--) print $i }
```

Pavyzdys su paaiškinimu:

```
awk /Julius/ stud1

awk '$2 == "Julius" ' studn
```

Išrinkime iš failo ***stud1*** tuos, kurių gimimo metai >= 1970 ir patalpinkime į failą ***astud1.***

```
awk '$4 >= 1970' stud1 > astud1
```

Čia 4-ojo lauko turinys bus lyginamas su 1970.

print operatorius ***awk*** komandoje nurodo standartinį išvedimo įtaisą. Nenurodžius komandoje šablono priimama, kad visi įrašai tenkina šabloną.

```
awk '{print}' stud4 awk '{print $1}' stud4
```

awk komandos pagalba iš ***stud1*** failo išrinkite pavardes ir gimimo metus ir surašykite juos į failą ***bstud1.***

Komandoje ***awk*** gali būti naudojami specialūs valdantys žodžiai ***BEGIN*** ir ***END.*** Sakinys po žodžio ***BEGIN*** yra įvykdomas prieš pirmos failo eilutės nuskaitymą, o sakinys po žodžio ***END,*** - tik apdorojus paskutinę failo eilutę.

Įvykdykite sudėtingą komandą:

```
awk 'BEGIN {print "Vardas Pavardė Gim.metai";} {print $2, $3, $4}

END {print "sąrašo pabaiga"}' stud1
```

Šiame pavyzdyje ***awk*** komanda turi visas tris sudėtines dalis (kiekviena iš jų apskliausta {} skliaustais). ***awk*** komandoje gali būti naudojamas tik vienas iš specialių valdančių žodžių ***BEGIN*** ir ***END,*** jei reikia atlikti veiksmus tik prieš failo įrašų apdorojimą arba tik po failo įrašų apdorojimo.

Pavyzdžiui,

awk '{print $2, $3, $4}

END {print "sąrašo pabaiga"}' stud1

";" naudojamas operatorių atskyrimui atitinkamose ***awk*** komandos dalyse (pradinėje, darbo, arba galinėje), kai keletas operatorių turi būti atliekami šiose dalyse. Sudėtingos ***awk*** komandos tekstas gali būt pakankamai ilgas. Tuomet jį patogiau surašyti į failą, kuris tada suprantamas, kaip ***awk*** programa.

vi redaktoriaus pagalba kataloge ***lab6*** sukurkite failą ***awkprog:***

BEGIN {print "Vardas Pavardė Gim.metai";}

{print $2, $3, $4}

END {print "viso įrašų =" NR}

Šią programą galima įvykdyti naudojant ***awk*** komandą su opcija ***-f:***
awk -f awkprog stud1 | tee cstud1

awk programų tekstuose gali būti sutinkami rezervuoti kintamieji. Tai, pavyzdžiui, ***NR*** - perskaitytų failo eilučių kiekis faile, ***NF*** - laukų skaičius einamojoje eilutėje. Programą nutraukti galima panaudojus komandą ***exit.***

Bendra if sakinio konstrukcija yra tokia:

*if (*reiškinys*)* operatoriusl *else* operatorius2

Sudarykime programą ***awkprog1,*** kuri skaičiuoja faile ***stud*** kiekį žmonių, gimusių iki 1973 ir kiekį gimusių vėliau.

BEGIN {print "IF -ELSE pavyzdys"; print "";} {if ($4 < 1973) n1=n1+1 else n2=n2+1} END {print n1,"gimė iki 1973"; print ""; print n2,"gimė vėliau";}

Čia buvo panaudoti laisvai pasirinkti kintamieji ***n1*** ir ***n2*** (jų nereikia skelbti, pagal nutylėjimą jie turi pradines nulines reikšmes, o norint užduoti kitokias pradines reikšmes, jas reikia užduoti ***BEGIN*** dalyje).

Panaudokite šią programą failui ***stud.***

Šią sudarytą programą ***awkprog1*** galima padaryti vykdomuoju failu, įrašius į jį pačią ***awk*** komandą. Atlikite pataisymus ***awkprog1*** ir įrašykite į failą ***awkprog2:***

awk 'BEGIN {print...

...vėliau";}' stud

ls komanda patikrinkite, ką galima atlikti su šiuo failu.

Failą galima padaryti vykdomuoju, panaudojus **chmod** komandą. Susipažinkite su šios komandos veikimo principais. Panaudokite **chmod** komandą ir padarykite failą **awkprog2** vykdomuoju. ***ls*** komanda patikrinkite, ką galima atlikti su šiuo failu.

Programa **awkprog2** bus vykdoma surinkus jos vardą. Duokite komandą **awkprog2.**

END