Contents

L2/1

Komandų interpretatorius (shell)

Pavyzdžiui, jungiantis prie sistemos SSH protokolu, po to kai sėkmingai autentifikuojatės, SSH serviso procesas (`sshd`) paleidžia Jūsų loginui priskirtą programą ir jai perleidžia sesijos terminalo valdymą. Kai ši programa baigia darbą - paprastai baigiama ir sesija.

- **`sh`** (*Bourne shell*) ir su ja suderinamos: `bash` (*Bourne Again shell*), `mksh` (MirBSD Korn Shell - *Free implementation of Korn shell*), `zsh`, `ash`, `dash`, ...
- **`csh`** (*C shell*) ir su ja suderinamos: `tcsh` (*TENEX C shell*)...
- Prisijungus prie sistemos jūs būsite aptarnaujami šio *shell*'o. Norint pereiti į kito *shell*'o aplinką (naudotis kitu *shell*'u) reikia įvykdyti atitinkamą komandą:

```
$ sh
$ tcsh
$ mksh
$ bash
```

- Grįžtama į buvusio *shell*'o aplinką naudojant komandą:

```
$ exit
```

- **interaktyvus** - *shell*'as "bendrauja" su vartotoju, t.y. laukia komandų, jas vykdo, atvaizduoja rezultatus ir kitą informaciją.
- **paketinis** - *shell*'as paeiliui vykdo komandas iš nurodyto failo (*shell* skripto).

Specialūs simboliai

Kai kurie simboliai *shell*'ui turi specialią prasmę: `;`, `>`, `<`, `|`, `\`, `&`, `$`, `` ` ``, `"`, `'`, `(`, `)`, nauja eilutė, *tab*, tarpas, o kai kada ir `*`, `?`, `[`, `#`, `~`, `=`, `%`.

Norint nuimti vieno simbolio specialią prasmę, prieš šį simbolį rašomas `\`, pavyzdžiui, `?` specialios prasmės išjungimas:

```
$ echo Are you sure you want to remove these files\?
```

Komandų seka

Nuosekliai vykdomos komandos *shell*'e atskiriamos nauja eilute arba `;`, pvz.:

```
$ komanda1 argumentai

$ komanda2
```

arba:

```
$ komanda1 argumentai ; komanda2
```

Įvedimo ir išvedimo perskirstymas

Susijus informacija: UNIX komandų sintaksė, http://wiki.bash-hackers.org/howto/redirection_tutorial

Paprastai su kiekvienu procesu yra surišti trys standartiniai failai (srautai):

- *stdin* (0) – standartinis įvedimo failas (iš jo procesas gali skaityti duomenis/komandas/...);
- *stdout* (1) – standartinis išvedimo failas (į jį procesas gali rašyti rezultatus);
- *stderr* (2) – standartinis klaidų išvedimo srautas (į jį procesas gali rašyti pranešimus apie klaidas ar pan.);

Interaktyvioms komandoms šie srautai susiejami su terminalu (pvz.: klaviatūra ir displėjumi; SSH klientu). Standartiniai srautai gali būti perskirstyti (pakeisti kitais failais). Perskirstymas nurodomas `sh` komandoje simboliais (supaprastinta):

- `> failas` – *stdout* nukreipimas į failą, failas sukuriamas arba jo turinys perrašomas;
- `>> failas` – *stdout* nukreipimas į failą, failas sukuriamas arba papildomas (*append*);
- `< failas` – *stdin* duomenys įvedami iš failo;
- `2> failas`, `2>> failas` – *stderr* perskirstymas, atitinkamai perrašant ir papildant failą;
- `2>&1` – *stderr* peradresavimas į *stdout* (*stderr* priskiriamas tas pats failas, kuris priskirtas *stdout*);
- `> failas 2>&1` – *stdout* ir *stderr* peradresavimas į failą (SVARBU: `2>&1 > failas` daro ne tą patį).

Naudojimo pavyzdžiai

```
$ cat
```

```
$ cat < failas
```

Čia pirmu atveju `cat` komanda skaitys eilutes įvedamas klaviatūra (iki bus paspaustas ^D) ir išvedinės jas į ekraną, o antru atveju – įvedinės eilutes iš failo ir išvedinės į ekraną.

```
$ cat aaa > aab
```

Čia `cat` komanda failo `aaa` turinio neišves į ekraną, o įrašys į failą `aab` (perrašys failą, jei toks jau yra).

```
$ cat antras.txt >> pirmas.txt
```

Čia `cat` komanda failo `antras.txt` turinį prirašo į failo `pirmas.txt` pabaigą.

```
$ ls -l kazkoks_failas > log.txt 2>&1
```

Jei failo `kazkoks_failas` einamajame kataloge nėra, tai pranešimas apie klaidą bus įrašytas į failą `log.txt`, o jei failas yra, tai į `log.txt` bus įrašyta `ls -l` išvesta informacija apie šį failą.

Vienu metu gali būti naudojamos kelios perskirstymo operacijos, pvz.:

```
$ cat < failas1 >> failas2
```

Kanalo mechanizmas

Susijus informacija: UNIX komandų sintaksė

Kanalas (*pipe*) komandos eilutėje žymimas `|` ir sujungia vienos komandos *stdout* srautą su kitos *stdin*. Srautas *stderr* niekur neperadresuojamas, todėl norint ir jį perduoti kitai komandai reikia *stderr* peradresuoti į *stdout*.

```
$ ls -l /bet_kas | grep 'rwx'

ls: cannot access '/bet_kas': No such file or directory
```

Šiuo atveju komanda `ls` neradus katalogo `/bet_kas` išves klaidos pranešimą, ir šio klaidos pranešimo eilutė keliaus į *stderr* (ne *stdout*) todėl `grep` komanda jo negaus ir neapdoros.

Pavyzdžiui:

```
$ ls -l /bet_kas 2>&1 | grep 'rwx'
```

Ši komanda nieko neišves į ekraną, nes *stderr* yra peradresuotas į *stdout*, kuris kanalu sujungtas su `grep` *stdin*, o `grep` eilutėje `/bet_kas: No such file or directory` neranda simbolių sekos `rwx`.

Failų vardų šablonai

Keli simboliai *shell* komandose turi specialią prasmę ir naudojami failų vardų šablonams aprašyti, t.y. prieš vykdant komandą šablonas pakeičiamas visais jį atitinkančiais failų vardais (iš einamojo katalogo, jei šablone nėra kelių).

Metasimboliai:

- `*` – atitinka bet kokią simbolių seką, tame tarpe ir tuščią (nulinio ilgio);
- `?` – atitinka vieną simbolį;
- `[]` – atitinka bet kurį vieną simbolį, nurodytą tarp šių skliaustų.
- Pavyzdžiui, jei einamajame kataloge turime failus ar katalogus `kat, la, lab, lab1, lab2, laba, laba1`, tai šablonas `lab*` komandoje:

```
$ ls lab*
```

- bus pakeistas vardais `lab lab1 lab2 laba laba1`.

```
$ ls lab?
```

- Šablonas `lab?` šioje komandoje bus pakeistas vardais `lab1 lab2 laba`.

```
$ ls lab[1-2]
```

- Šablonas `lab[1-2]` šioje komandoje bus pakeistas vardais `lab1 lab2`.

```
$ ls la*/ff?
```

- Pavyzdžiui jei `la` ir `lab2` yra katalogai ir kiekviename jų yra failai kurių vardas sudarytas iš trijų simbolių ir prasideda `ff`, šablonas `la*/ff?` bus pakeistas pvz.: `la/ff1 la/ffX lab2/ff. lab2/ffa`.

Kabutės

shell'e naudojamos trijų tipų kabutės:

1. `'` – specialią prasmę turintys *shell* simboliai esantys eilutėje tarp šių kabučių praranda specialią prasmę (*shell* juos supranta kaip paprastus simbolius);
2. `` ` `` – šiomis kabutėmis apskliausta eilutė vykdoma kaip komanda ir jos išvedami duomenys įstatomi vietoj šios eilutės;
3. `"` – prasmė panaši į `'`, bet dalis spec. simbolių išlaiko specialią prasmę (`$`, `` ` ``, `\`).
4. Išvedami *shell* spec. simboliai:

```
$ echo 'Spec simboliai: $><%`"\'

Spec simboliai: $><%`"\
```

7. Tarkim faile `sarasas` yra tik viena eilutė, kurioje surašyti failų vardai:

```
$ cat sarasas

file1 file2 file3
```

10. tada komanda:

```
$ ls -l `cat sarasas`
```

12. vietoj `` `cat sarasas` `` bus įstatytas failo `sarasas` turinys (taip, kaip jį išveda `cat` komanda) ir `ls -l` išves informaciją apie tuos failus.

13. Čia `"` apskliaustos eilutės viduje naudojamas `\` nuima specialią `"` ir `` ` `` prasmę:

```
14. $ echo "Naudojamos kabutes: \"\`'"

15. Naudojamos kabutes: "`'
```

Reguliarios išraiškos

1. **`.`** – bet koks vienas simbolis;
2. **`[`** ir **`]`** – bet kuris vienas simbolis iš esančių tarp šių skliaustų, nurodant simbolių aibę galima naudoti:
    1. intervalus, pvz.: **`a-z`**;
    2. klases: `[:alnum:]`, `[:alpha:]`, `[:blank:]`, `[:cntrl:]`, `[:digit:]`, `[:graph:]`, `[:lower:]`, `[:print:]`, `[:punct:]`, `[:space:]`, `[:upper:]`, `[:xdigit:]`.
3. **`[^`** ir **`]`** – bet kuris vienas simbolis iš nesančių tarp šių skliaustų;
4. **`^`** – eilutės pradžia (t.y. atitikimas šablonui tikrinamas nuo eilutės pradžios);
5. **`$`** – eilutės pabaiga (t.y. tikrinama ar šablonas atitinka eilutę iki jos pabaigos);
6. **`\`** – po jo einančiam simboliui nuima specialią prasmę (t.y. panaudojus \ prieš metasimbolį, metasimbolis laikomas paprastu simboliu);
7. **`*`** – bet koks prieš šį simbolį esančios reguliarios išraiškos pasikartojimų skaičius (nuo nulio);
8. **`\{N\}`** – prieš šią išraišką esančios reguliarios išraiškos pasikartojimas `N` kartų;
9. **`\{N,\}`** – prieš šią išraišką esančios reguliarios išraiškos pasikartojimas mažiausiai `N` kartų;
10. **`\{N,M\}`** – prieš šią išraišką esančios reguliarios išraiškos pasikartojimas nuo `N` iki `M` kartų;
11. **`\(`** ir **`\)`** – grupavimas (pvz. visai apskliaustai simbolių sekai galima taikyti `*` ar `{}`);
12. **`\N`** – prieš šią išraišką esančios `()` apskliaustos išraiškos reikšmė (t.y. eilutės dalis, kuri atitiko N-ąjį `()` apskliaustą fragmentą), `N=1..9` (apskliaustos išraiškos numeris).

"Išplėstose" reguliariose išraiškose (*extended regular expression*) specialią prasmę papildomai turi metasimboliai:

1. **`+`** – bent vienas prieš šį simbolį esančios reguliarios išraiškos pasikartojimų skaičius;
2. **`?`** – nulis arba vienas prieš šį simbolį esančios reguliarios išraiškos pasikartojimas;
3. **`|`** – alternatyva, t.y. `|` sujungus dvi reguliarias išraiškas, gauta išraiška atitiks teksto fragmentą, jei jį atitiks bent viena iš apjungtų reguliarių išraiškų;
4. **`(`** ir **`)`** – tas pats, kaip *basic* `\(\)`;
5. **`{`** ir **`}`** – tas pats, kaip *basic* `\{\}`.

reguliarios išraiškos (*basic*)			
šablonas	**atitinka**	**neatitinka**	**šablono prasmė**
`a.b`	arb ssavbs aa ss a b	ab bac .ab	eilutė turi simbolį a, po kurio seka bet koks simbolis, o po jo simbolis b
a[rv]b	arb ssavbs		ieškoma seka, susidedanti iš trijų simbolių, kurių pirmasis yra a, trečiasis yra b, o antrasis yra arba r, arba v
a[^rv]b	aa ss a b		ieškoma seka, susidedanti iš trijų simbolių, kurių pirmasis yra a, trečiasis yra b, o antrasis nėra nei r, nei v
^s	ssavbs		ieškoma eilutė, kurios pirmas simbolis s
s$	ssavbs		ieškoma eilutė, kurios paskutinis simbolis s

\^	aa^xx	abc	ieškoma simbolio ^
ab\{3\}c	abbbc	abc	po a turi būti 3 b, po to c
abc\([1-5]\)x\1y	abc3x3y	abc3x2y	seka turi prasidėti abc, paskui vienas skaitmuo nuo 1 iki 5 (tas skaitmuo įsimenamas kaip \1), po to x, po to \1 įsiminta reikšmė, po to y
"išplėstos" reguliarios išraiškos (*extended*)			
ab+c	abbbc	ac	pirmas sekos simbolis turi būti a, paskui bent viena b, o po to c
vienas|du	du	su	atitinka simbolių sekas `vienas` ir du, bet ne vieną iš simbolių s arba d
ab?c	ac	abbc	pirmas sekos simbolis turi būti a, paskui viena b, bet nebūtinai, o po to c

grep

Susijusi informacija: grep pagrindai
***PASTABA:** Kai kuriose Unix šeimos operacinėse sistemose gali būti kelios `grep` komandos.*

Komanda grep (*global regular expression print*) – tai filtras atrenkantis šabloną (reguliarią išraišką) tenkinančias teksto eilutes.

```
grep [OPTION...] PATTERNS [FILE...]

grep [OPTION...] -e PATTERNS ... [FILE...]

grep [OPTION...] -f PATTERN_FILE ... [FILE...]
```

Parinktys:

- `-E` – šablonas aprašytas "išplėsta" reguliaria išraiška;
- `-F` – šablonas aprašytas nenaudojant reguliarių išraiškų (šablono metasimboliai neturi jokios specialios prasmės);
- `-c` – išvesti tik šabloną atitinkančių eilučių skaičių (neišvesti pačių eilučių);
- `-e pattern_list` – ieškoti pagal `pattern_list` sąrašu nurodytus šablonus, šablonai atskiriami nauja eilute;
- `-f pattern_file` – ieškoti pagal `pattern_file` faile esančius šablonus (kiekvienas šablonas naujoje eilutėje);
- `-i` – ieškant neskirti didžiųjų ir mažųjų raidžių;
- `-l` – išvesti tik failų vardus, kuriuose buvo rasta šablono atitikimų;
- `-n` – prieš kiekvieną išvedamą eilutę rašyti eilutės numerį faile;
- `-q` – nieko neišvesti (skiriasi tik pabaigos kodas, apie kurį bus kalbama vėliau);
- `-s` – nespausdinti klaidų pranešimų;
- `-v` – ieškoti eilučių, neatitinkančių šablonų;
- `-x` – ieškoti tik visą šabloną atitinkančių eilučių (t.y. šablonas turi atitikti eilutę nuo pradžios iki pabaigos).

Jei nenurodytas joks duomenų failas `file` – ieško per *stdin* įvedamuose duomenyse.

Naudojimo pavyzdžiai

```
$ cat failas1

4as Poska

1as Lecius

2-as Lenkutis

3-as Zigmutis iskelta i a-5


$ grep '^[0-5]as' failas1

4as Poska

1as Lecius


$ grep '[af].[4-6]$' failas1

3-as Zigmutis iskelta i a-5


$ grep -vi L failas1

4as Poska


$ grep  'a[^0-9]b' failas1

$ grep -E 'Dana|Rita' pav

$ grep -E 'a(ir)?b'

$ grep -E 'a(ir)+b'

$ grep -E 'a(ir)*b'
```

awk

PASTABA: *Unix šeimos operacinėse sistemose yra kelios awk, patartina naudoti awk arba gawk. Aprašymas sutrumpintas/supaprastintas. Pilną aprašymą galite rasti man, info arba POSIX standarte.*

awk – tai kalba ir šios kalbos interpretatorius, skirti šablonais aprašomoms manipuliacijoms su tekstu. Nuskaitomus duomenis awk suskaldo įrašais (pagal nutylėjimą "įrašas" – teksto eilutė). awk programa sudaryta iš šablonų ir veiksmų, kuriuos reikia atlikti su šabloną atitinkančiais įrašais. Su kiekvienu įrašu atliekami visi jį atitinkantiem šablonams priklausantys veiksmai.

```
awk [-F ERE] [-v var=value]... program [argument...]

awk [-F ERE] -f progfile [-f progfile]... [-v var=value]... [argument...]

gawk [ POSIX or GNU style options ] -f program-file [ -- ] file ...
```

```
gawk [ POSIX or GNU style options ] [ -- ] program-text file ...
```

Parinktys:

- `-F ERE` – duomenų įrašų laukai skiriami ne tarpu, o `ERE` nurodyta išplėstine reguliaria išraiška;
- `-f progfile` – vykdoma *awk* programa esanti faile `progfile`;
- `-v var=value` – reikšmės *awk* kintamąjam priskyrimas.

`argument` gali būti apdorojamo failo vardas, arba kintamojo priskyrimas. Failo vardas `-` nurodo, kad apdorojami duomenys skaitomi iš *stdin*. Jei duomenų failas nenurodytas – apdorojami per *stdin* paduodami duomenys.

Bendra *awk* programos struktūra:

```
išraiška { veiksmai }
```

Veiksmų aprašymo sintaksė labai artima C.

Išraiška gali būti:

1. `BEGIN` – su ja susiję veiksmai atliekami prieš pradedant duomenų apdorojimą;
2. `END` – su ja susieti veiksmai atliekami baigus apdorojimą (pasibaigus duomenims);
3. *aritmetinė_išraiška* – išraiška iš kintamųjų ir operacijų su jais (sintaksė labai artima C/C++);
4. `/` *reguliari išraiška* `/` – išplėsta reguliari išraiška.
5. *išraiška1* `,` *išraiška2* – susiję veiksmai atliekami su eilutėmis pradedant nuo tos, kuriai teisinga *išraiška1* iki tos, kuriai teisinga *išraiška2* imtinai, po to vėl laukiama *išraiška1* atitinkančios eilutės. Abi išraiškos gali būti ir aritmetinės, ir reguliarios.

Jei išraiška nenurodyta – veiksmai vykdomi su kiekvienu įrašu (eilute). Jei nenurodyti veiksmai – kiekvienam išraišką atitinkančiam įrašui vykdoma `{ print }` (t.y. išvedamas visas įrašas).

Kintamieji

Kintamieji sukuriami pirmą kartą į juos kreipiantis, jų deklaruoti nereikia. Priklausomai nuo konteksto, kintamieji automatiškai konvertuojami tarp eilutės ir skaitmeninių (sveiko ir slankaus taško) tipų. Yra specialią prasmę turinčių kintamųjų (sąrašas sutrumpintas):

- `FILENAME` – apdorojamo duomenų failo vardas;
- `NR` – apdorojamo įrašo numeris;
- `FNR` – apdorojamo įrašo dabartiniame duomenų faile numeris;
- `NF` – apdorojamo įrašo laukų skaičius;
- `FS` – įrašo laukų skirtukas (g.b. reguliari išraiška);
- `RS` – įrašų skirtukas (pagal nutylėjimą nauja eilutė).
- `OFS` – išvedamų laukų skirtukas `print` komandai;
- `ORS` – išvedamų įrašų skirtukas;

Operacijos ir komandos

awk naudojamos komandos (`if`, `while`, `do..while`, `for`, `break`, `continue` ...) ir operacijos (+, -, *, /, <, >...) artimos C kalbos analogams.

Specifinės išraiškos:

- `$expr` – kreipinys į apdorojamo įrašo lauką (lauko kintamasis), `$0` – visas įrašas, `$1` .. `$NF` – atskiri laukai, galima ir skaityti laukų kintamuosius, ir jiems priskirti reikšmes;
- `str ~ regexp` – eilutės `str` atitikimas reguliariai išraiškai `regexp`;
- `str !~ regexp` – `str` neatitikimas `regexp`;

- `print expr [, expr]` – išvedimas, jei išvedama keletas išraiškų, jos atskiriamos `OFS` reikšme, po kiekvieno `print` išvedama `ORS` reikšmė (nauja eilutė);
- `expr expr` – specialaus eilučių sujungimo operatoriaus (*concatenation*) nėra, tai pasiekiama išraiškas rašant vieną prie kitos (kartais galima neatskirti).

Kitos awk galimybes: aritmetinės ir eilučių apdorojimo funkcijos, naudotojo funkcijos, išvedimo peradresavimas, kanalai ...

Pavyzdžiai

Pavyzdžiui, turime failą `a1`.

```
$ cat a1

01 asd 8211 abcd

02 das 8211 fsdf
```

Tada:

```
$ awk '{print $1,$2}' a1

01 asd

02 das
```

Tas pats, bet išraiškų neskiriant kableliais (eilučių sujungimas):

```
$ awk '{print $1 $2}' a1

01asd

02das
```

Laukų išvedimo tvarką galime sukeisti:

```
$ awk '{print $4, $3}' a1

abcd 8211

fsdf 8211
```

Išvedamos failo eilutės, kurių ketvirtas laukas ≥ 8. (Jei tarp lyginamų reikšmių yra skaičių, *awk* bando visus operandus paversti į skaičius. Jei nepavyksta - operandai lyginami kaip eilutės.
Todėl `Informatika ≥ 8` ir pirma eilutė išvedama):

```
$ cat studentai

Pavarde    Matematika   Fizika  Informatika

Jonaitis        7          8         8

Petraitis       8          9         9

Juska           9          6         7
```

```
$ awk '{if ( $4 >= 8 ) print $0}' studentai

Pavarde    Matematika   Fizika  Informatika

Jonaitis        7          8         8

Petraitis       8          9         9

$ cat awkf2

{ if ($2 >= 8)

  if ($3 >= 8)

   if ($4 >= 8) print $0 }

$ awk -f awkf2 studentai

Pavarde    Matematika   Fizika  Informatika

Petraitis       8          9         9

$ cat awkf3

BEGIN {print "Gerai besimokantys studentai" }

{ if ($2 >= 8)

  if ($3 >= 8)

   if ($4 >= 8) print $0 }


$ awk -f awkf3 studentai

Gerai besimokantys studentai

Pavarde    Matematika   Fizika  Informatika

Petraitis       8          9          9
```

Duomenų laukai faile atskirti ne tarpais, o `:`, išvedant tarp laukų papildomai įterpiamas *TAB* simbolis (`\t`, C sintaksė):

```
$ awk -F : '{print $1,"\t",$3}' /etc/passwd | tail -3

vaickaro         1497

vaitgyti         1498

vaivtoma         1499
```

Keli analogiški variantai, kaip galima iš `/etc/passwd` failo atrinkti eilutes su žodžiu `Tomas` 5-ame lauke:

```
$ awk -F : '{if ( $5 ~ /Tomas/) print $0}' /etc/passwd | tail -1

narbtoma:x:1452:101:Tomas Narbutaitis ? IFN-11 2010-09-23:/home/narbtoma:/bin/bash
```

```
$ awk -F : '$5 ~ /Tomas/{ print }' /etc/passwd | tail -1

narbtoma:x:1452:101:Tomas Narbutaitis ? IFN-11 2010-09-23:/home/narbtoma:/bin/bash



$ awk -F : '$5 ~ /Tomas/' /etc/passwd | tail -1

narbtoma:x:1452:101:Tomas Narbutaitis ? IFN-11 2010-09-23:/home/narbtoma:/bin/bash
```

Puslapių į kuriuos kreipiamasi išskyrimas iš www serverio log failo:

```
$ awk -v FS=' "|" ' '{print $2}' /data/ld/Solaris_access_log
```

`iostat` išvedamų rezultatų apdorojimo pavyzdys (Solaris OS, vidutinis visų diskų apkrovimas %):

```
$ iostat -x 1 | awk 'BEGIN{dt=1; ORS=" ";} /^Device/{print bt/dt; bt=0; dt=0;}
/^xvdap/{bt+=$14; dt++;}'
```

Užduotys

shell pagrindai

- Koks *shell* Jums priskirtas?

Mums yra priskirtas bash shell

- Kaip paleisti `ls / /neratokio` komandą, kad jos rezultatai būtų spausdinami į vieną failą, o klaidos į kitą?

ls / /neratokio > output.txt 2> errors.txt

- Kaip paleisti `ls / /neratokio` komandą, kad jos rezultatai ir klaidos būtų spausdinami į vieną failą?

ls / /neratokio >> failas 2>&1

- Kaip paleisti `ls / /neratokio` komandą, kad jos rezultatai būtų perduodami komandai `cksum`, o klaidos būtų išvedamos į failą?

ls / /neratokis 2> errors.txt | cksum

- Kaip paleisti `ls / /neratokio` komandą, kad jos klaidos būtų perduodamos komandai `cksum`, o rezultatai būtų išvedami į failą?

  (ls / /neratokis 2> errors.txt || true) | cksum > checksum.txt 2>> errors.txt

  Kaip paleisti `ls / /neratokio` komandą, kad jos rezultatai ir klaidos būtų perduodami komandai `cksum`?

  (ls / /neratokis 2>&1 | cksum)

- kaip suskaičiuoti bendrą kelių failų kontrolinę sumą (`cksum`) nekuriant laikino/tarpinio failo?

  ls | cksum

- iš katalogo `/usr/sbin` išrinkite failus, kurių vardai prasideda `re` ir surūšiuokite juos atvirkščia tvarka;

  ls -l /usr/sbin/ | grep "^re" | sort -r

- iš katalogo `/usr/bin` išrinkite failus, kurių vardai sudaryti iš dviejų simbolių;
  - iš jų atrinkite tuos, kurių pavadinime yra skaičių.

  ls -l /usr/bin | grep -E '^..[0-9].*$'

grep

- iš failo `/data/ld/ld2/stud2001` išrinkit eilutes, kurių gale yra skaičius `7`

  grep '7$' /data/ld/ld2/stud2001

- iš failo `/data/ld/ld2/stud2001` išrinkit eilutes, kurių pradžioje yra skaičius `2`

  grep '^2' /data/ld/ld2/stud2001

- iš failo `stud2001` išrinkit eilutes, kurių gale yra raidė `n` arba `d` arba raidės nuo `r` iki `v`.

  grep '[ndr-v]$' /data/ld/ld2/stud2001

- iš failo `stud2001` išrinkit eilutes, kuriose yra įrašas apie studentus, kurių vardai sutampa su Jūsų vardu.

  grep 'Lukas' /data/ld/ld2/stud2001

- iš failo `stud2001` išrinkit eilutes, kuriose nėra `IF 9/1` grupės studentų.

grep '[^IF 9/1]' /data/ld/ld2/stud2001

- užrašykite filtrą, kuris ieškotų eilučių, kuriose nėra jokio skaičiaus.

  grep [^0-9] /data/ld/ld2/stud2001

- užrašykite filtrą, kuris ieškotų eilutėse bet kokios raidės(tiek didžiosios, tiek mažosios).

  grep -i [a-z] /data/ld/ld2/stud2001

- užrašykite filtrą, kuris iš failo `stud2001` išrinktų eilutes, kuriose po `A` raidės seka raidė `n` arba raidė `u`, o po jų seka raidės `d` ir `r`.

  grep 'A[nu]dr' /data/ld/ld2/stud2001

- užrašykite filtrą, kuris iš failo `stud2001` išrinktų eilutes, kuriose po `A` raidės seka raidė `n` arba raidė `u`, o po jų seka raidės `d` ir `r`, o po raidės `r` nėra raidės `e`.

  grep 'A[nu]dr[^e]' /data/ld/ld2/stud2001

- užrašykite filtrą, kuris faile `stud2001` ieškotų eilučių su įrašais apie `9/1` arba `9/2` grupių studentus.

  grep '9/[12]' /data/ld/ld2/stud2001

- užrašykite filtrą, kuris ieškotų eilučių besibaigiančių seka `andr` arba `mari`.

  grep '[andr|mari]$' /data/ld/ld2/stud2001

- Ką reiškia išplėstos reguliarios išraiškos: `a(i)+b`, `a(i)*b`, `a(i)?b`. Duomenų pavyzdys: source:shell|duom

a(i)+b – prasideda a po to eina vienas arba daugiau i ir baigiasi su b

a(i)*b – prasideda a po to eina 0 arba daugiau i ir baigiasi su b

a(i)?b – prasideda a po to eina 0 arba 1 i ir baigiasi su b

```
1 Abccc
2 Abbbbc
3 Aiiibc
4 Aibbc
5 Acb
6 Bca
```

awk

- Nusikopijuokite, išsiaiškinkite ir išbandykite kataloge `/data/ld/ld2/awk/` esančias `awk` programas `awk1`, `awk2` ir `awk3`, skirtas failo `/data/ld/ld2/stud2001` apdorojimui.

- Sudarykite programą `awk11`, kuri iš failo `/data/ld/ld2/stud2001`, atrinktų studentus, kurių vardai sutampa su Jūsų vardu ir atspausdintų eilutės numerius kuriose jie rasti, jų pavardes bei vardus.

```
#!/usr/bin/awk -f



BEGIN {

   # Set your name here

   name = "Lukas"

   # Set the field separator to a comma

   FS = ","

}



# For each line in the file, check if the name matches yours

$5 == name {

   # Print the line number, surname, and first name

   printf("%d: %s %s\n", NR, $4, $5)

}
```

- Sudarykite programą `awk12`, kuri iš failo `stud2001` išrinktų tas eilutes, kuriose yra `IF 9/2` grupės studentai ir gale atspausdintų, kiek eilučių rasta.

```
#!/usr/bin/awk -f



BEGIN {

   count = 0;

}



/IF 9\/2/ {

   print;

   count++;
```

```
}


END {

   print "Number of lines found: " count;

}
```

- Sudarykite programą `awk13`, kuri iš failo `/data/ld/ld2/awk/preke` išrinktų tas eilutes, kuriose yra informacija apie tas prekes, kurios visos yra parduotos.

```
Awk13 > {if($2 ~ $3) print $0} /data/ld/ld2/awk/preke



{

if($2 == $3) {print $0}

}
```

- Sudarykite programą `awk14`, kuri iš failo `/data/ld/ld2/awk/preke` išrinktų maksimalias stulpelių reikšmes bei gale jas atspausdintų.

```
awk '{if($0=sort -k 3 /data/ld/ld2/awk/preke | head -7 | tail -1) print $0)}'
/data/ld/ld2/awk/preke



#!/usr/bin/awk -f



NR == 1 {

   for (i = 1; i <= NF; i++) {

      max[i] = $i;

   }

}
```

```
NR > 1 {
   for (i = 1; i <= NF; i++) {
      if ($i > max[i]) {
         max[i] = $i;
      }
   }
}

END {
   for (i = 1; i <= NF; i++) {
      printf("Max value of column %d: %s\n", i, max[i]);
   }
}
```

- Turime failą `/data/ld/ld2/srautai.txt`, kurį generuoja duomenų srautų stebėjimui ir analizei skirta programa Netflow. Eilutėse laukai atskirti skyrikliu `:`. Laukų prasmė tokia:
    1. Kompiuterio, iš kurio siunčiamas duomenų paketas IP adresas.
    2. Kompiuterio, į kurį siunčiamas duomenų paketas IP adresas.
    3. Maršrutizatoriaus, pro kurį eis duomenų paketas IP adresas (jei IP yra `62.40.103.217` – tai rodo, kad duomenų paketas yra siunčiamas į interneto tinklą)
    4. SNMP indeksai (įėjimo ir išėjimo interfeiso)
    5. Paketų skaičius
    6. Baitų skaičius (visuose paketuose).

- Sudarykite programą `awk15`, kuri iš šio failo suskaičiuotų, kiek paketų ir kiek baitų yra iškeliavę į Internetą.

```
awk -F: '{ if ($2 == "62.40.103.217") { total_packets += $4; total_bytes += $5; } } END { print "Total packets sent to the Internet:", total_packets; print "Total bytes sent to the Internet:", total_bytes; }' /data/ld/ld2/srautai.txt
```

Sudarykite awk filtrą, kuris kaip duomenis imtų `ls -l` rezultatus ir suskaičiuotų/atspausdintų bendrą katalogo failų dydį (užtenka vertint `ls` išvedamą dydį, t.y. kietų nuorodų, failų tipų ir kt. vertint nereikia) pvz.:

```
• $ ls -l | awk -f awkdu
• Total = 1456892
```

```
ls -l | awk '{suma+=$5} END {print suma}'


#!/usr/bin/awk -f


BEGIN {

    total = 0;

}


NR > 1 {

    total += $5;

}


END {

    printf("Total = %d\n", total);

}
```

Bendros užduotys

Galite naudoti visas šiame ir ankstesniuose užsiėmimuose nagrinėtas priemones.

`apache_log` failo formatas `(failo vardas Solaris_access_log)` :
`klientoIP - - [data laiko_zona] "HTTP uzklausa" http_atsakymas atsakymo_dydis`
http_atsakymas – skaičius: 200 - ok, 404 - klaida, ...
atsakymo_dydis – serverio išsiųsto atsakymo klientui dydis baitais.

- išrinkite vardus iš `/data/ld/ld2/stud2001` failo ir atspausdinkite juos ir jų pasikartojimų skaičių, surūšiuotus pagal pasikartojimų skaičių;

  awk '{print $5}' /data/ld/ld2/stud2001 | sort | uniq -c | sort -rn

  iš `/data/ld/ld1/Solaris_access_log` išrinkite:

  - daugiausia užklausų į serverį atsiuntusių klientų IP adresus (pirmus 10, surūšiuokite pagal atsiųstų užklausų kiekį);

awk '{print $1}' /data/ld/ld1/Solaris_access_log | sort | uniq -c | sort -nr | head -n 10

awk ‘{print $5}’ /data/ld/ld1/Solaris_access_log | sort -r | head – 10 | awk ‘{print $1}’ /data/ld/ld1/Solaris_access_log

- daugiausia duomenų iš serverio nusiurbusių klientų IP adresus (pirmus 10, surūšiuokite pagal duomenų kiekį);

awk '{ sum[$1] += $10 } END { for (i in sum) { print i, sum[i] } }' /data/ld/ld1/Solaris_access_log | sort -k 2 -rn | head -n 10

- populiariausius puslapius (kurie yra serveryje), surūšiuokite pagal kreipinių skaičių;

awk '{print $7}' /data/ld/ld1/Solaris_access_log | sort | uniq -c | sort -rn

- IP adresus, kurie galbūt bandė laužtis į serverį (lindo į nesamus puslapius);

awk '$9 == 404 {print $1}' /data/ld/ld1/Solaris_access_log

- populiariausius laužimosi būdus (t.y. skirtingi IP, bet bandė taip pat laužtis);

awk '$9 == 404 {print $1}' /data/ld/ld1/Solaris_access_log | sort | uniq -c | awk '$1 > 10 {print $2}'

- pertvarkykite `/data/ld/ld2/srautai.txt` į lengviau žmogui skaitomą formatą (IP adresai be tarpų, laukai skiriami tarpais, ...)

awk -F: '{print $1,$2,$3,$4,$5,$6}' /data/ld/ld2/srautai.txt

- iš `srautai.txt` (originalo, performuoto ar pan.) suformuokite sąrašą:
  - naudojamų maršrutizatorių;

awk -F':' '{print $3}' /data/ld/ld2/srautai.txt | sort -u

  - naudojamų išėjimo interfeisų;

awk -F':' '{print $4}' /data/ld/ld2/srautai.txt | sort -u

  - per kurį maršrutizatorių išsiųsta kiek duomenų.

awk -F':' '{sum[$3] += $6} END {for (i in sum) print i, sum[i]}' /data/ld/ld2/srautai.txt

  - su kuriais išėjimo interfeisais surištas kuris maršrutizatorius;

awk -F':' '{sum[$3 ":" $4] += $6} END {for (i in sum) print i, sum[i]}' /data/ld/ld2/srautai.txt

- iš `strace` išvedamų rezultatų atfiltruokite *syscall*'ams argumentais perduodamas simbolių eilutes (eilutė išskirta)

strace -e trace=open,write -s 64 -f ls 2>&1 | awk '!/"/'

L2/2

Svarbiausi dalykai

Sh – interpretatorius shell scriptam.

Bash – default shell kuri naudojame

Shell sintaksė – negalima tarpo po priskyrimo kintamojo (tarpų nereikia)

Informacijos perdavimas į scriptą – stdinn (ls -l | f1.sh arba nuskaitymas is failo < f.txt)

Aritmetinės išraiškos – teksto apdorojimas, expr (2+2) $((2+2))

Paprastos shell programos

Jei komandų seką tenka dažnai naudoti, galima ją įrašyti į failą. Tokio tipo failas vadinamas *shell* script'u arba *shell* programa. *Shell* programose be komandų sekų galima naudoti ir sudėtingesnes konstrukcijas (sąlyginės komandos, išsišakojimai, ciklai, funkcijos ir pan.). Su šiomis valdymo struktūromis bus supažindinama sekančiuose laboratorinio darbo užsiėmimuose.

Shell programos yra interpretuojamos. **Tai reiškia, kad kompiliuoti jų nereikia**.

Paprastos *shell* programos pavyzdys

Tarkime kursime *shell* programą **test**, sudarytą iš sekančios komandų sekos:

```
date

who

du /etc
```

Shell programa paprastai pradedama antrašte, kurią sudaro speciali simbolių seka `#!`. Po šios simbolių sekos nurodomas absoliutus kelias iki interpretatoriaus bei jo parametrai. Pavyzdžiui, aukščiau pateikta **test** *shell* programa turėtų būti papildyta sekančia antrašte:

```
#!/bin/sh

date

who

du /etc
```

Pastaba. Shell programose, kurių interpretatoriaus vieta skirtingose mašinose gali skirtis, g.b. naudojama antraštė, iškviečianti interpretatorių kokios nors standartinės programos pagalba, pvz. Perl script'ui vietoj `#!/opt/local/bin/perl6` naudoti `#!/bin/env perl6`

Aukščiau pateiktuose *shell* programos pavyzdžiuose (ir su antrašte ir be jos) komandos vykdomos nuosekliai, viena po kitos. Failas sukuriamas redaktoriaus pagalba (pavyzdžiui, nano). Komandos surašomos į failą ir jis užsaugomas.
Užsaugojus failą, reikalinga suteikti jam vykdymo teises. Tai padaryti galima komandos `chmod` pagalba:

```
$ chmod +x test
```

Failui suteikus vykdymo leidimus, *shell* programą galima vykdyti:

- Jei interpretatorius (`sh, bash, perl, python ...`) *shell* programos antraštėje **nenurodytas**, jis iškviečiamas komandinėje eilutėje parametru nurodant *shell* programą:

```
$ sh test

$ bash test

$ perl test

...
```

- Jei interpretatorius *shell* programos antraštėje **nurodytas**, tuomet komandinėje eilutėje jo nurodyti nebereikia. Esant *shell* programai einamajame kataloge ją vykdome:

```
$ ./test
```

*Pastaba. Dažniau naudojamas antrasis shell programos vykdymo būdas, nes nereikia spėlioti, kokiam interpretatoriui skriptas parašytas. Pirmasis būdas naudingas, jei shell programos pradžioje interpretatorius nurodytas neteisingai (pvz. script'as atkeliavo iš kitos mašinos, kur jo interpretatorius buvo kitokiam kataloge) arba jei reikia apeiti OS ribojimus draudžiančius vykdyti konkrečioje failų sistemoje esančius failus (failų sistema primontuota su **noexec** vėliavėle). Jei interpretatorius shell programoje nenurodytas ir jūs programą vykdote nenurodydami jo ir komandinėje eilutėje - programą vykdys `sh` shell'as.*

Komentarai shell *programoje*

Rašant *shell* programą, komandų sekos, loginės programos dalys yra paprastai komentuojamos, prieš komentarą dedant `#` simbolį. Pavyzdžiui:

```
#!/bin/sh
```

```
# Author : ILA

# Script follows here:

date

who

du /etc
```

Kintamųjų vardai

Kintamasis, tai simbolių seka, kuriai priskiriama kokia nors reikšmė (simbolių seka). Priskiriama reikšmė gali vaizduoti skaičių, tekstą, failo vardą, įrenginį ar kitokio tipo duomenis. *Shell* aplinka leidžia kurti, trinti kintamuosius, priskirti jiems reikšmes.

Kintamojo vardas sudaromas tik iš raidžių (`a-z` ir `A-Z`), skaičių (`0-9`) ir simbolio `_`. Kiti simboliai (`!`, `*`, `-` ir kt.) nenaudojami, nes paprastai turi tam tikrą kitą prasmę shell'e.

Teisingi kintamųjų vardai:

```
_ALI

TOKEN_A

VAR_1

variable3
```

Neteisingi kintamųjų vardai:

```
2_VAR

-VARIABLE

VAR1-VAR2

VAR_A!

%variable3
```

Reikšmės priskyrimas kintamajam ir jos naudojimas

Vartotojo sukurtų kintamųjų **specialiai aprašyti nereikia**. *Shell*'e kintamasis sukuriamas ir jam priskiriama atliekant priskyrimo veiksmą (**tarpų prieš lygybės ženklą ir už jo neturi būti**):

```
vardas=reikšmė
```

Pavyzdžiui:

```
VAR1="Zara Ali"

VAR2=100
```

Kintamojo reikšmės įvedimui *shell*'e naudojama `read` komanda, pvz. kintamojo `var1` įvedimas iš *stdin* (klaviatūros ar peradresuoto failo):

```
read var1
```

Kintamojo reikšmė panaudojama prieš kintamojo vardą nurodant `$` ženklą. Tuomet vietoj išraiškos `$var1` įstatoma kintamojo `var1` reikšmė.

Pavyzdžiai

- Turime *shell* programą:

```
#!/bin/sh


# Author : ILA

# Script follows here:


echo "Koks jūsų vardas?"

read PERSON

echo "Sveiki, $PERSON"
```

Įvykdžius *shell* programą, gausime žemiau pateiktą rezultatą:

```
$./test

Koks jūsų vardas?

Zara Ali

Sveiki, Zara Ali
```

- Turime *shell* programą:

```
#!/bin/sh

read a b

read c

echo "a="$a

echo "b="$b

echo "c="$c
```

Bei duomenų failą `failas`:

```
1 2 3 4 5
```

```
6 7 8 9

10 11 12
```

Paleidus vykdyti programą gausime tokius atsakymus:

```
$ ./pav < failas

a=1

b=2 3 4 5

c=6 7 8 9
```

Shell programų poziciniai argumentai

Shell'o programos gali būti paleidžiamos nurodant **argumentus**, kurie dar vadinami **poziciniais argumentais**. Su jais programos viduje galima atlikti norimus veiksmus. Argumentai yra nurodomi komandinėje eilutėje po programos vardo, pavyzdžiui:

```
$ ./proc a b
```

Čia `a` ir `b` bus programai `proc` perduodami poziciniai argumentai. Į juos programos viduje galima kreiptis nurodant numerį `$1` ,`$2`, ... `$9`, t.y. `$1` atitiktų reikšmę `a`, o `$2` atitiktų reikšmę `b`. Pozicinis argumentas `$0` visada yra komandos vardas, šiuo atveju, `proc`.

Pozicinių argumentų prasmė priklauso nuo *shell* programos. Pavyzdžiui, jei sudarome programą vardu `sc`, kuri pertvarko vieną failą į kitą, tai ši programa gali būti paleidžiama su dviem argumentais - failų vardais:

```
$ ./sc fileA fileB
```

o programos veiksmuose pirmas failas galės būti nurodomas naudojant pozicinį argumentą `$1` (vykdant jis atitiks `fileA`). Veiksmai su antru failu galės būti aprašomi naudojant `$2` (vykdant atitiks `fileB`).

Jei reikia sužinoti **pozicinių argumentų kiekį** (pavyzdžiui, norint patikrinti ar vartotojas nurodė jų tiek, kiek reikia), programoje naudojamas `$#` kintamasis.

Jei programoje jums reikia **visų pozicinių argumentų** naudojamas `$*` kintamasis.

Komanda shift

Pozicinių argumentų reikšmės pasiekiamos naudojant kintamuosius `$1` .. `$9`. Jei argumentų yra daugiau, jų reikšmes pasiekti galima pastūmus visą pozicinių argumentų sąrašą į kairę. Tam naudodajama komanda `shift`:

```
shift [N]
```

Išstumti argumentai dingsta. `shift` argumentas (sveikas skaičius), jei yra, nurodo kiek pozicinių argumentų reikia išstumti. Stumiant argumentai iš naujo numeruojami, mažėja jų skaičius.

Pavyzdžiui, turime tokią shell programą `pav3`:

```
#!/bin/sh

echo $*

shift

echo parametrai po shift $*

echo parametru kiekis po shift $#

shift 2

echo parametrai po \"shift 2\" $*

echo po shift 2 parametru kiekis $#
```

Vykdome šią *shell* programą:

```
$ ./pav3 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

parametrai po shift 2 3 4 5 6 7 8 9 10 11 12 13 14 15

parametru kiekis po shift 14

parametrai po "shift 2" 4 5 6 7 8 9 10 11 12 13 14 15

po shift 2 parametru kiekis 12
```

Aritmetinės operacijos

shell'e aritmetinės išraiškos užrašomos specialiu formatu:

```
$((aritmetinė_išraiška))
```

arba galima naudoti papildomas programas (pvz.: `expr`):

```
expr aritmetinė_išraiška
```

Pavyzdys, iliustruojantis dviejų skaičių sudėtį:

```
#!/bin/sh

val=$((2+2))

echo "Galutinis rez : $val"
```

arba naudojant `expr`:

```
#!/bin/sh

val=`expr 2 + 2`
```

```
echo "Galutinis rez : $val"
```

Įvykdžius šias *shell* programas abiem atvejais gausime:

```
Galutinis rez : 4
```

Pastaba. Naudojant `expr` tarp operatorių, kintamųjų ir reikšmių turi būti dedami tarpai. Pavyzdžiui `2+2` bus neteisingas užrašymo būdas. Reiktų rašyti `2 + 2`.

shell'as palaiko tokius aritmetinius operatorius (pavyzdžiai pateikti, kai pradinės kintamųjų `a` ir `b` reikšmės yra `a=10`, `b=20`):

Operatorius	**Aprašas**	**Pavyzdys naudojant `expr`**	**Pavyzdys naudojant `$((..))`**	**Rezultatas (`rez` reikšmė)**
+	sudėtis	`rez=`expr $a + $b``	`rez=$((a+b))`	30
-	atimtis	`rez=`expr $a - $b``	`rez=$((a-b))`	-10
*	daugyba	`rez=`expr $a * $b``	`rez=$((a*b))`	200
/	dalyba	`rez=`expr $b / $a``	`rez=$((b/a))`	2
%	dalybos liekana	`rez=`expr $b % $a``	`rez=$((b%a))`	0
=	priskyrimas	`rez=$b`	`rez=$b`	20

Užduotys

Paprastos shell programos

- Parašykite *shell* programą, kuri išvestų informaciją apie vartotojo procesus (komanda `ps`), prisijungusius prie sistemos vartotojus (komanda `who`) bei einamojo katalogo turinį (komanda `ls`). Komandų rezultatai turėtų būti atskirti "----------" žyme. Programą įvykdykite.

#!/bin/sh

ps

echo "----------"

who

echo "----------"

ls

echo "----------"

shell kintamieji

- Kataloge `lab2` sukurkite žemiau pateiktą *shell* programą `proc1b`, ją įvykdykite, gautus rezultatus paanalizuokite :

```
#!/bin/sh

a1=studentai

a2=' laba diena '

svente='2013  03  11'

echo  $a2 $a1

echo "su svente "  $svente

echo iveskite savo varda

read    vardas

echo "su svente"  $vardas

echo  iveskite tris skaicius

read   a  b

echo "ivesta a= "  $a "ivesta b= "  $b
```

- Sukurkite *shell* programą `proc2b`, kurioje būtų įvedama ir priskiriama kintamiesiems:
    1. jūsų prisijungimo vardas;
    2. vartotojo numeris;
    3. grupė.

    Visa įvesta informacija turi būti atspausdinta vienoje eilutėje.

```
#!/bin/sh

echo "Iveskite prisijungimo varda: "

read prisijungimo_vardas

echo "Iveskite vartotojo numeri: "

read vartotojo_nr

echo "Iveskite grupe: "

read grupe

echo $prisijungimo_vardas $vartotojo_nr $grupe
```

Aritmetiniai veiksmai

- Turime du kintamuosius: `a=40` ir `b=6`. Parašykite *shell* programą `proc3b`, kuri atliktų sekančius veiksmus:
    - a padauginta iš b
    - a padalinta iš b
    - a dalyba iš b gaunant liekanos reikšmę

```
#!/bin/sh

a=40

b=6

daugyba=$(($a*$b))

dalyba=$(($a/$b))

liekana=$(($a%$b))


echo "Daugyba " $daugyba

echo "Dalyba " $dalyba

echo "Liekana " $liekana
```

- Sukurkite *shell* programą `proc4b`, kuri atliktų šiuos veiksmus:
    - Įvestų Jūsų gimimo metus, menesį ir dieną (skaičius) į kintamuosius `s1`, `s2` ir `s3`
    - Įvestas reikšmes atspausdinkite;
    - Padalinkite `s1` iš `s2`, ir `s1` iš `s3` skaičiaus;
    - Gaukite dalybos rezultatų sumą bei liekanų sandaugą.

```
#!/bin/sh

echo "Iveskite gimimo metus: "

read s1

echo "Iveskite gimimo menesi: "

read s2

echo "Iveskite gimimo diena: "

read s3

echo "Gimimo metai: " $s1

echo "Gimimo menuo: " $s2
```

```
echo "Gimimo diena: " $s3

dalyba1=$((s1/s2))

dalyba2=$((s1/s3))

liekana1=$((s1%s2))

liekana2=$((s1%s3))


dalybos_suma=$((dalyba1+dalyba2))

liekanu_sandauga=$((liekana1*liekana2))


echo "Dalybos suma: " $dalybos_suma

echo "Liekanu sandauga: " $liekanu_sandauga
```

Shell programų poziciniai argumentai

- Sukurkite *shell* programą `proc5b`, kuri išvestų komandinėje eilutėje įvestų pozicinių argumentų kiekį ir jų reikšmių sąrašą.

  ```
  #!/bin/sh

  echo "Pozicinių argumentų kiekis: " $#

  echo "Visi poziciniai argumentai: " $*
  ```

- Paanalizuokite ką išves *shell* programa `proc5b`, paleidus ją vykdymui taip:

```
$ ./proc5b 123

$ ./proc5b tai yra taip



lukkuz1@oslinux ~/lab2_2/scriptai $ ./proc5b 123

Pozicinių argumentų kiekis:  1

Visi poziciniai argumentai:  123

lukkuz1@oslinux ~/lab2_2/scriptai $
```

```
lukkuz1@oslinux ~/lab2_2/scriptai $ ./proc5b tai yra taip

Pozicinių argumentų kiekis:  3

Visi poziciniai argumentai:  tai yra taip

lukkuz1@oslinux ~/lab2_2/scriptai $
```

- Sukurkite *shell* programą `proc6b`, į kurią įrašykite tokius veiksmus:

```
#!/bin/sh

echo "poziciniai argumentai="$*

a=$1$2

echo "a=$a"

$a

b=`$3$4`

echo "b=$b"
```

Padarykite *shell* programą vykdomąja ir ją įvykdykite taip:

```
./proc6b    p s i d
```

Paanalizuokite gautus rezultatus. Kaip pasikeistų rezultatai, jei antroje eilutėje tarp `$1` ir `$2` įdėtume tarpą?

lukkuz1@oslinux ~/lab2_2/scriptai $ ./proc6b p s i d

poziciniai argumentai=p s i d

a=ps

PID	TTY	TIME	CMD
560022	pts/4	00:00:00	bash
560576	pts/4	00:00:00	more
560814	pts/4	00:00:00	more
560827	pts/4	00:00:00	awk
561053	pts/4	00:00:00	awk

561101 pts/4 00:00:00 more

561303 pts/4 00:00:00 grep

561308 pts/4 00:00:00 grep

561354 pts/4 00:00:00 uniq

561361 pts/4 00:00:00 uniq

561394 pts/4 00:00:00 uniq

561646 pts/4 00:00:00 more

562424 pts/4 00:00:00 more

565466 pts/4 00:00:00 proc6b

565467 pts/4 00:00:00 ps

b=uid=2221(lukkuz1) gid=100(users) groups=100(users)

Pasikeistu a kintamasis iš ps į p s, nebus išvedami procesai.

- Koks rezultatas būtų išvestas įvykdžius tokius veiksmus:

```
• c=`expr 2 + 2`
• d=`expr 2+2`
• echo $c + 2
• echo $d + 2
• echo $c $d
```

```
lukkuz1@oslinux ~/lab2_2/scriptai $ ./script2
4 + 2
2+2 + 2
4 2+2
lukkuz1@oslinux ~/lab2_2/scriptai $


TARPAI YRA SVARBUSSSSS
```

- Peržiūrėkite *shell* programą `/data/ld/ld2/2/bsh9` ir ją įvykdykite. Kokia `set` prasmė?

  Nustato kompiuterio datą, kuriame dirbame.

- Modifikuokite `proc2b` taip, kad jūsų prisijungimo vardas, vartotojo numeris ir grupė būtų išvedami automatiškai - jums nevedant šių duomenų (informacijai apie prisijungimo vardą,

vartotojo numerį ir grupę(-es) gauti naudokitės šią informaciją teikiančiomis shell'o komandomis). Modifikuotą shell programą užsaugokite `proc2b_mod` vardu.

```
#!/bin/sh

u="$USER"
id="$(id -u)"
group="$(groups)"
echo "Jūsų vartotojo vardas: " $u
echo "Jūsų vartotojo numeris: " $id
echo "Jūsų grupė: " $group
```

L2/2 scriptai

L2/3

Pabaigos kodas

http://teaching.idallen.com/dat2330/04f/notes/exit_status.txt Pabaigos kodas (Exit Status) / Grąžinama reikšmė (Return Code)

Kiekvienas pasibaigęs Unix procesas (komanda) operacinei sistemai grąžina sveiką skaičių - pabaigos kodą. Pagal susitarimą, jei grąžinama reikšmė yra nulis, tai rodo, kad vykdyta komanda baigėsi be klaidų; bet kokia kita reikšmė reiškia, kad vykdymo metu kažkas nepavyko.

SVARBU: *pabaigos kodo skaičiaus prasmė priešinga, nei įprasta C/C++, t.y. 0 = TRUE, 1 = FALSE, 2 = FALSE ir t.t.*

Pavyzdžiui:

```
$ date

Tuesday, January 28, 2014 04:26:41 PM EET

$ echo $?

0
```

Šiuo atveju pabaigos kodas yra 0 ir tai rodo, kad komanda `date` pavyko.

Kitas pavyzdys:

```
$ grep neratokio /etc/passwd
```

```
$ echo $?

1
```

Šiuo atveju `grep` komandai nepavyko rasti nurodyto šablono duomenyse ir pabaigos kodas gavosi 1.

```
$ grep neratokio nera-tokio-failo

grep: can't open nera-tokio-failo

$ echo $?

2
```

Kintamasis `?` saugo tik paskutinės vykdytos komandos grąžintą pabaigos kodą, pavyzdžiui:

```
$ grep neratokio nera-tokio-failo

grep: can't open nera-tokio-failo

$ echo $?

2

$ echo $?

0
```

Čia išvesta 0 reikšmė jau rodo, kad prieš tai vykdyta `echo` komanda pavyko.

Komanda `exit` *shell* programose nutraukia skriptą ir grąžina jai argumentu nurodytą skaičių, kaip skripto pabaigos kodą. Payzdžiui:

```
#!/bin/sh

echo "Hello World!"

exit 147
```

Jei skriptas pasibaigia be komandos exit, tai skripto pabaigos kodas prilyginamas skripto paskutinės vykdytos komandos pabaigos kodui.

Turime skriptą, įrašytą į failą `pav` tokį:

```
#!/bin/sh

grep "$1" /etc/passwd

status=$?         # pabaigos kodas issaugomas kintamajame status

echo "grep programa iesko '$1' ir grazina koda $status"

exit $status      # skriptas baigiamas tokiu paciu kodu, koki grazino grep
```

Vykdome jį:

```
$ ./pav root

root:x:0:0:Super-User:/root:/usr/bin/bash

grep programa iesko 'root' ir grazina koda 0

$ sh pav nera-tokio

grep programa iesko 'nera-tokio' ir grazina koda 1
```

test, [] – sąlygos tikrinimas

Komandos `test` ir `[..]` naudojamos sąlygų tikrinimui. Abiejų šių komandų veikimas analogiškas - komandos suskaičiuoja argumentais duotos sąlygos rezultatą ir atitinkamai nustato pabaigos kodą (0 - kai gaunama *TRUE*, 1 - kai gaunama *FALSE*).

```
test salyga

[ salyga ]
```

Čia `salyga` – sąlygos išraiška (aritmetinė, loginė ir t.t.), apie galimas sąlygos išraiškas bus kalbama toliau.

Pavyzdžiai:

```
$ test a = b

$ echo $?

1

$ [ a = b ]

$ echo $?

1
```

Aritmetinių palyginimų operatoriai

Operatorius	Aprašas	Pavyzdys	$?
-eq	(==) tikrina, ar operandai yra lygūs	[10 -eq 20]	1
-ne	(!=) tikrina, ar operandai nėra lygūs	[10 -ne 20]	0
-gt	(>) tikrina, ar kairėj pusėj esantis operandas yra didesnis nei esantis dešinėje	[10 -gt 20]	1
-lt	(<) tikrina, ar kairėj pusėj esantis operandas yra mažesnis nei esantis dešinėje	[10 -lt 20]	0

`-ge`	(>=) tikrina, ar kairėj pusėj esantis operandas yra didesnis arba lygus operandui, esančiam dešinėje	`[ 10 -ge 20 ]`	1
`-le`	(<=) tikrina, ar kairėj pusėj esantis operandas yra mažesnis arba lygus operandui, esančiam dešinėje	`[ 10 -le 20 ]`	0

PASTABA: visos tikrinamos sąlygos rašomos skliaustų viduje atitraukiant tarpais jas nuo skliaustų, pavyzdžiui `[ $a -le $b ]` yra teisingas užrašas, o `[$a -le $b]` neteisingas.

shell'as palaiko lentelėje pateiktus sąlygos operatorius skaitmeninėms reikšmėms (šie operatoriai netinka eilutėms ar kintamiesiems, kuriems priskirtos eilutės, lyginti), pvz:

```
$ [ 1 -eq 1 ]

$ [ a -eq a ]

-bash: [: a: integer expression expected
```

Eilučių tikrinimo operatoriai

Operatorius	**Aprašas**	**Pavyzdys**	**$?**
str1 = *str2*	lygina, ar eilutės *str1* ir *str2* vienodos	`[ abc = "abc" ]`	0
str1 != *str2*	lygina, ar eilutės *str1* ir *str2* skiriasi	`[ abc != def ]`	0
`-z` *str*	tikrina, ar eilutės *str* ilgis yra nulinis	`[ -z '' ]`	0
`-n` *str* *str*	tikrina, ar eilutės *str* ilgis yra nenulinis (netuščia eilutė)	`[ -n '' ]` `[ '' ]`	1

Failų tikrinimo loginiai operatoriai

Tarkime turime:

```
-r--r--r--   1 kespaul  users        100 Feb  1 18:57 failas
```

Operatorius	**Aprašas**	**Pavyzdys**	**$?**
`-b` *failas*	ar *failas* yra blokinio tipo specialus failas	`[ -b failas ]`	1
`-c` *failas*	ar *failas* yra simbolinio tipo specialus failas	`[ -c failas ]`	1
`-d` *failas*	ar *failas* yra katalogas	`[ -d . ]`	0
`-e` *failas*	ar *failas* egzistuoja	`[ -e / ]`	0

`-f` *failas*	ar *failas* yra paprastas failas (ne katalogas ar specialus failas)	`[ -f failas ]`	0
`-g` *failas*	ar *failas* turi įjungtą SGID bitą	`[ -g failas ]`	1
`-h` *failas* `-L` *failas*	tikrina, ar *failas* simbolinė nuoroda	`[ -h /lib/libc.so ]`	0
`-k` *failas*	ar *failas* turi įjungtą *sticky* bitą	`[ -k /tmp ]`	0
`-p` *failas*	ar *failas* yra kanalas (*pipe*)	`[ -p failas ]`	1
`-t` *N*	ar deskriptorius *N* yra atidarytas ir surištas su terminalu (nesigilinkit)	`[ -t 1 ]`	0
`-u` *failas*	ar *failas* turi įjungtą SUID bitą	`[ -u /bin/su ]`	0
`-r` *failas*	ar *failas* yra skaitomas (ar vykdomas procesas gali jį skaityti)	`[ -r failas ]`	0
`-w` *failas*	ar į failą *failas* galima rašyti	`[ -w failas ]`	1
`-x` *failas*	ar *failas* yra vykdomas	`[ -x /bin/ls ]`	0
`-s` *failas*	ar *failas* netuščias (jo dydis didesnis nei 0)	`[ -s failas ]`	0

Loginių sąlygų operatoriai

shell naudoja loginius operatorius kelių loginių išraiškų rezultatų apjungimui.

Operatorius	Aprašas	Pavyzdys	$?
! *salyga*	loginė inversija (*NOT*), invertuoja *salyga* reikšmę	`[ a = b ]`	1
		`[ ! a = b ]`	0
salyga1 `-o` *salyga2*	loginis ARBA (*OR*)	`[ 10 -lt 20 -o a = b ]`	0
salyga1 `-a` *salyga2*	loginis IR (*AND*)	`[ 10 -lt 20 -a a = b ]`	1

Sąlyginis komandų vykdymas

Komandose naudojami loginiai operatoriai:

- *komanda1* `&&` *komanda2* – loginis IR, *komanda2* vykdoma tik jei *komanda1* grąžino 0.
- *komanda1* `||` *komanda2* – Loginis ARBA (OR), *komanda2* vykdoma tik jei *komanda1* grąžino ne 0.

- `!` *komanda* – Tai yra loginis neigimas (invertuoja *komanda* grąžintą `$?`).

Pavyzdžiai:

```
$ cd /var && ls -l
```

Jei pavyks pereiti į katalogą `/var` , tai bus parodytas jo turinys.

```
$ cat aa || cat bb
```

Jei pavyks parodyti failo `aa` turinį, `bb` turinys nebus rodomas, o jei nepavyks parodyti `aa` - bus bandoma parodyti `bb`

Tie patys pavyzdžiai panaudojant inversiją:

```
$ ! cd /var || ls -l

$ ! cat aa && cat bb
```

if .. then .. elif .. else .. fi sakinys

`if` leidžia pasirinkti vieną iš vykdymo kelių, priklausomai nuo komandoje tikrinamos sąlygos (komandos pabaigos kodo).

```
if komanda

then

    komandų sąrašas

fi
```

Naudojant dvi šakas:

```
if komanda

then

    komandų sąrašas

else

    komandų sąrašas

fi
```

Esant keletui tikrinimų:

```
if komanda

then

    komandų sąrašas
```

```
elif komanda

then

    komandų sąrašas

fi
```

Pavyzdžiui:

```
#!/bin/sh

if grep nera_tokio /etc/passwd

then

    echo "==== nera_tokio surastas faile   /etc/passwd"

    cat /etc/passwd

else

    echo "==== Nerastas nera_tokio faile /etc/passwd"

fi
```

Šiuo atveju `if` komanda tikrina `grep` komandos pabaigos kodą ir priklausomai nuo jo vykdo vieną ar kitą veiksmų grandinėlę

Du vienodai veikiantys `if` sakiniai, tikrinantys aritmetinę sąlygą (tikrinamas `test` arba `[ .. ]` komandos pabaigos kodas):

```
$ a=3; b=2

$ if test $a -gt  $b ;then echo "a>b"; fi

$ if [ $a -gt $b ]; then echo "a>b"; fi
```

Tikrinama sąlyga gali būti suformuojama naudojant atitinkamą bazinį operatorių, pavyzdžiui vykdant veiksmus su aritmetinėmis reikšmėmis:

Šiame pavyzdyje tikrinamos failo charakteristikos:

```
#!/bin/sh

file="$1"

if [ -r $file ]

then

    echo "Failas turi read leidima"

else

    echo "Failas neturi read leidimo"

fi
```

Sekančiame pavyzdyje tikrinamos simbolių eilutės:

```
#!/bin/sh

read a

read b

if test -n "$a"; then

    if [ "$a" = "$b" ]; then

        echo "Eilutės a ir b vienodos ir netuščios."

    fi

fi
```

case sakinys

`case` leidžia palyginti vieną reikšmę su daug kitų reikšmių, ir radus sutapimą, vykdyti sąrašą atitinkamų komandų. Truputį supaprastinta sintaksė:

```
case str in

    str1 | str2) komanda

              komanda;;

    str3) komanda

          komanda;;

    *) komanda;;

esac
```

Čia `strN` – šablonai su kuriais lyginama `str` eilutė. Tai ne reguliarios išraiškos ir nevisai *shell* metasimboliai, tačiau artimesni *shell* metasimboliams:

- `*` – bet kokia simbolių seka;
- `?` – bet koks vienas simbolis;
- `[ ABC ]` – simbolių aibė (bet kuris vienas simbolis iš išvardintų tarp skliaustelių);
- `[! ABC ]` – simbolių aibė (bet kuris vienas simbolis nesantis tarp išvardintų tarp skliaustelių);
- `\` – specialios sekančio simbolio (metasimbolio) reikšmės panaikinimas.

Pavyzdžiui (*yorno.sh*):

```
#!/bin/sh



echo "Ar jus sutinkate? [yes arba no]: "

read yno

case $yno in
```

```
        [yY] | [yY][Ee][Ss] )
                echo "Sutinku"
                ;;
        [nN] | [nN][Oo] )
                echo "Nesutinkate, todel negalite testi";
                exit 1
                ;;
        *) echo "Blogi duomenys"
           exit 255
           ;;
esac
```

Šioje programoje įvesta `$yno` reikšmė lyginama su keliais šablonais ir vykdoma atitinkama veiksmų seka. Jei įvesta reikšmė nesutampa su laukiamomis reikšmėmis – tokį atvejį apdoroja metasimboliu `*` pažymėta šaka.

for var in ... ; do ... ; done

`for` ciklo struktūra:

```
for var [ in wordlist ]
do
   command-list
done
```

čia `var` – kintamojo vardas, `wordlist` tarpais atskirtų žodžių aibė, `command-list` – komandų seka. Ciklas vykdomas tiek kartų, kiek yra žodžių `wordlist` aibėje, kiekvieno vykdymo metu kintamajam `var` priskiriama iš eilės po vieną `wordlist` aibės žodį. Jei `in wordlist` yra praleista, tada `var` priskiriamos pozicinių argumentų reikšmės, t.y.:

```
#!/bin/sh
for v; do echo $v; done
```

yra tas pats, kaip:

```
#!/bin/sh
for v in "$@"; do echo $v; done
```

Pavyzdžiai

```
#!/bin/sh
```

```
for v in 0 1 2 3 nulis vienas du trys "du simtai"

do

   echo $v

done
```

Čia kintamasis `v` įgaus kiekvieną iš nurodytų reikšmių ir vykdant ciklą jos bus spausdinamos.

```
#!/bin/sh

for i in `ls` ; do

   echo $i

done
```

Šiame pavyzdyje bus atspausdinti visi einamojo katalogo failai, tačiau šis skriptas veiks neteisingai, jei kataloge bus failų su tarpais pavadinime.

```
#!/bin/sh

for i in * ; do

   echo $i

done
```

Šiuo atveju bus korektiškai atspausdintas visų katalogo failų sąrašas.

while ... ; do ... ; done

`while` konstrukcija:

```
while command-list1

do

   command-list2

done
```

`while` ciklo komandų seka `command-list2` yra kartojama tol **kol tenkinama sąlyga**, t.y. `command-list1` grąžina `0`.

Pavyzdžiai

```
#!/bin/sh

c=1

while [ $c -le 5 ]

do

   echo "=== c = $c"
```

```
    c=$(( $c + 1 ))

done
```

until ... ; do ... ; done

`until` konstrukcija:

```
until command-list1

do

    command-list2

done
```

`until` cikle komandų sąrašas `command-list2` vykdomas tol **kol netenkinama sąlyga** `command-list1`.

Pavyzdžiai

```
#!/bin/sh

a=x

until [ $a = xxxxxxxxxx ]

do

    echo $a

    a="${a}x"

done
```

L2/3 scriptai

Užduotys

shell loginių išraiškų užduotys

- sukurkite skriptą, kuris:
    - priimtų 2 argumentus (grąžintų klaidą jei per daug ar per mažai argumentų)

#!/bin/sh

```
#!/bin/bash


# Check if the number of arguments is correct

if [ "$#" -ne 2 ]; then

echo "Error: The script requires exactly 2 arguments"

exit 1

else

echo "Geras argumentu kiekis"

fi
```

- patikrintų ar argumentais nurodyti failai ir ar jų savininkas tas pats, atspausdintų atitinkamą pranešimą ir grąžintų `0` jei tas pats ir `1` jei skirtingi.

```
#!/bin/bash


# Check if the number of arguments is correct

if [ "$#" -ne 2 ]; then

  echo "Error: The script requires exactly 2 file arguments"

  exit 1

fi


# Assign the arguments to variables

file1=$1

file2=$2


# Check if both files exist

if [ ! -e "$file1" ] || [ ! -e "$file2" ]; then

  echo "Error: One or both files do not exist"

  exit 1

fi
```

```
# Get the owner of both files
owner1=$(stat -c '%U' "$file1")
owner2=$(stat -c '%U' "$file2")

# Compare the owners
if [ "$owner1" = "$owner2" ]; then
  echo "The owners of $file1 and $file2 are the same: $owner1"
  exit 0
else
  echo "The owners of $file1 and $file2 are different: $owner1 and $owner2"
  exit 1
fi
```

shell sąlygos sakinių užduotys

- turime komandos eilutę:

```
$ cd /def && ls; cd /var/lib && ls
```

  - kaip pasikeistų jos veikimas pakeitus `&&` į `||` ir kodėl (abu variantus paleidžiant iš namų katalogo)?

    && - vykdo abi komandas, nors ir vienos komandos kodas gali būti 0. || pereina I kitos komandos vykdyma kai pirmos komandos kodas grazina ne 0.

- kuo skiriasi žemiau parodytų komandos eilučių veikimas?

```
$ cd; cd /def || cd /var/lib; pwd
$ cd; cd /var/lib || cd /def ; pwd
```

Pirmosios komandos atveju, nera direktorijos /def, todel vykdoma cd /var/lib komanda.

Antrosios komandos atveju, yra direktorija /var/lib, todel vykdoma cd /var/lib komanda.

  - kas būtų, jei abu katalogai egzistuotų (abi `cd` komandos suveikia be klaidų)?

    Išvedama pati pirmoji cd komandos eilutė – kadangi grazinamas 0.

- *[advanced]* pataisykite komandos eilutę, kad `pwd` būtų paleidžiama tik tuo atveju, jei bent vienas katalogas egzistuoja (t.y. nebūtų iškviečiama, jei nė viena `cd` nesuveikė)

```
If cd /def || cd /var/lib; then pwd; else echo bloga;
```

- sukurkite programą kuri:
  - jei paleista be argumentų atspausdintų pranešimą ir grąžintų klaidą
  - jei paleista su argumentais - atspausdintų argumentų skaičių ir argumentų sąrašą

```
#!/bin/sh

if [ $# -gt 0 ]

then

  echo $*

  echo "Argumentu skaičius: " $#

else

  echo "Nera argumentu"

  echo $?

fi
```

- papildykite aukščiau sukurtą programą:
  - jei programai nurodytas vienas argumentas ir jis yra katalogo vardas atspausdintų šio katalogo turinį
  - programa turi teisingai veikti su vardais, kuriuose yra tarpo simbolių

```
#!/bin/sh

if [ $# -gt 0 ]

then

  echo $*

  echo "Argumentu skaičius: " $#

  if [ -d $1 ]

  then

  ls -l "$1"

  else

    echo "Ne katalogas"

  fi
```

```
else
  echo "Nera argumentu"
  echo $?
Fi
```

- papildykite aukščiau sukurtą programą:
  - patikrintų ar programai pateiktas vienas argumentas ir jis yra failo vardas
  - jei tai skaitomas failas - paklaustų ar rodyti jo turinį (T/N) ir jei atsakymas teigiamas - išvestų failo turinį į terminalą

```
#!/bin/sh

if [ $# -eq 1 ] && [ -f "$1" ] && [ -r "$1" ]; then
  echo "Argument: $1"
  read -p "Display contents? [Y/N] " display
  if [ "$display" = "Y" ] || [ "$display" = "y" ]; then
    cat "$1"
  else
    echo "File contents not displayed."
  fi
elif [ $# -gt 0 ]; then
  echo "Arguments: $*"
  echo "Number of arguments: $#"

  if [ -d "$1" ]; then
    echo "Folder contents:"
    ls -l "$1"
  else
    echo "Not a folder or not a readable file."
  fi
else
```

```
    echo "No arguments provided."
    exit 1
  fi
```

- sukurkite programą kuri:
  - patikrintų ar komandinėje eilutėje yra nurodyti du argumentai,
  - nustatytų, ar pirmasis argumentas yra katalogas,
  - nustatytų, ar antras argumentas yra failas,
  - jei abi šios sąlygos tenkinamos – perkeltų nurodytą failą į katalogą
  - jei failo neišeina perkelti (pvz.: dėl teisių) jį nukopijuotų
  - jei kuri nors sąlyga netenkinama – išvestų atitinkamą informaciją ir grąžintų klaidą

```
#!/bin/bash

if [ $# -ne 2 ]; then
  echo "Error: Two arguments are required."
  exit 1
fi

if [ ! -d "$1" ]; then
  echo "Error: First argument is not a directory."
  exit 1
fi

if [ ! -f "$2" ]; then
  echo "Error: Second argument is not a file."
  exit 1
fi
```

```
if [ ! -w "$1" ]; then
  echo "Error: Cannot write to directory."
  exit 1
fi

mv -i "$2" "$1"

exit 0
```

- sukurkite programą kuri:
  - lauktų įvedimo
  - įvedus skaičių arba lietuvišką savaitės dienos pavadinimą išvestų atitinkamą savaitės dienos pavadinimą angliškai
  - įvedus Q arba q baigtų darbą
  - kitais atvejais – išvestų pagalbos pranešimą

```
#!/bin/bash

while true; do
  read -p "Enter a number or the Lithuanian name of the day of the week (or 'q' to quit): " input

  case "$input" in
    1 | "Pirmadienis")
      echo "Monday"
      ;;
    2 | "Antradienis")
      echo "Tuesday"
      ;;
    3 | "Trečiadienis")
      echo "Wednesday"
      ;;
    4 | "Ketvirtadienis")
      echo "Thursday"
      ;;
    5 | "Penktadienis")
      echo "Friday"
      ;;
    6 | "Šeštadienis")
      echo "Saturday"
      ;;
    7 | "Sekmadienis")
      echo "Sunday"
      ;;
    q)
```

```
        exit 0
        ;;
      Q)
        exit 0
        ;;

      *)
        echo "Invalid input. Please enter a number between 1 and 7 or the Lithuanian name of the day of the week (e.g. 'pirmadienis') or 'q' to quit."
        ;;
    esac
done

```

[advanced]: sukurkite programą, kuri patikrintų ar jai argumentu nurodytas failas yra vykdomasis (pagal failo turinį, ne pagal leidimus)

```
#!/bin/bash

if [ $# -ne 1 ]; then
  echo "Usage: $0 <filename>"
  exit 1
fi

filename="$1"

file_type="$(file -b "$filename")"

if [[ "$file_type" == *"executable"* ]]; then
  echo "$filename is executable"
else
  echo "$filename is not executable"
fi
```

shell ciklo sakinių užduotys

- sukurkite programą kuri atspausdintų kiekvieną įvestą argumentą bei jo numerį.

```
 #!/bin/sh

j=0;

for i in $*; do

   echo $i $j

   j=$((j+1))

done
```

- sukurkite programą kuri kiekvienam darbinio katalogo failui (ne katalogui) išvestų komandos `file` rezultatus.

```
#!/bin/bash

for file in *

do

  if [ -f "$file" ]; then

    echo "Results of 'file $file':"

    file "$file"

    echo ""

  fi

done
```

- sukurkite programą kuri įvestų skaičių eilutę (skaičiai atskirti tarpais), po to atskirai atspausdintų skaičius mažesnius už pirmąjį ir didesnius arba lygius pirmąjam.

```
#!/bin/bash


read -p "Enter a string of numbers separated by spaces: " input

numbers=($input)
```

```
first_number=${numbers[0]}


less_than_first=()

greater_than_or_equal_to_first=()


for number in "${numbers[@]:1}"

do

  if (( $number < $first_number )); then

    less_than_first+=($number)

  else

    greater_than_or_equal_to_first+=($number)

  fi

done


echo "Numbers less than $first_number: ${less_than_first[@]}"

echo "Numbers greater than or equal to $first_number:
${greater_than_or_equal_to_first[@]}"
```

- sukurkite programą kuri kiekvienam failui, nurodytam argumentų eilutėje atspausdintų pranešimą formatu: `failo_vardas >> [ne]skaitomas/[ne]vykdomas`.

```
#!/bin/bash


# Loop through all arguments (assumed to be file names)

for file in "$@"

do

  # Check if the file exists

  if [ -e "$file" ]; then

    # Check if the file is readable

    if [ -r "$file" ]; then

      read_status="yes"
```

```
    else
      read_status="no"
    fi

    # Check if the file is executable
    if [ -x "$file" ]; then
      exec_status="yes"
    else
      exec_status="no"
    fi

    # Print the message with the file name and read/executable status
    echo "$file >> read:$read_status/executable:$exec_status"
  else
    echo "$file does not exist."
  fi
done
```

- sukurkite programą kuri cikle su `shift` „suvalgytų" ir atspausdintų po vieną visus argumentus

```
#!/bin/bash
# Loop through all arguments
while [ $# -gt 0 ]
do
  # Print the first argument
  echo "Argument: $1"

  # Shift the arguments to remove the first one
  shift
done
```

- sukurkite programą kuri atspausdintų skaičių seką nuo 0 iki argumentu nurodyto skaičiaus.

```
#!/bin/sh


# Get the number from the first argument

number=$#


# Loop from 0 to the specified number

for (( i=0; i<$number; i++ ))

do

  echo $i

done
```

- sudarykite programą, kuri kas 60 sekundžių tikrintų ar yra prisijungęs bent vienas argumentais nurodytas naudotojas, o jį pastebėjusi išvestų informaciją apie tai ir pasibaigtų.

```
#!/bin/bash


# Check if arguments were provided
if [ $# -eq 0 ]; then
  echo "Usage: $0 username1 username2 ..."
  exit 1
fi


# Loop indefinitely
while true
do
  # Loop through all specified users
  for user in "$@"
  do
    # Check if the user is logged in
    who | grep -q "$user"
    if [ $? -eq 0 ]; then
      # If the user is logged in, print information and exit
```

```
    echo "$user is logged in."
    exit 0
  fi
 done

 # Wait for 60 seconds before checking again
 sleep 60
done
```

L2/4

Aplinkos kintamieji

Pagal prieinamumą, *shell* naudojamus kintamuosius galima skirstyti į dvi grupes:

- lokalūs kintamieji – matomi tik dabartinio *shell* procese;
- aplinkos kintamieji (*environment variables*) – paveldimi iš tėvo proceso ir matomi vaikų procesuose, t.y. vaiko procesas gauna aplinkos kintamųjų kopijas, bet negauna lokalių kintamųjų.

Aplinkos ir lokalių kintamųjų naudojimas *shell* programoje nesiskiria (jų reikšmių skaitymas ir modifikavimas vienodas).

Kai kurie aplinkos kintamieji UNIX aplinkoje turi specialią prasmę (sąrašas nepilnas, be to įvairios programos gali naudoti savo papildomus aplinkos kintamuosius):

Aplinkos kintamasis	Jo reikšmė
`EDITOR`	teksto redaktorius, kuris kviečiamas pagal nutylėjimą, pvz.: `nano`, `vi`, `vim`
`HOME`	naudotojo namų katalogas
`LANG`, `LC_ALL`, `LC_...`	naudojama lokalė (įtakoja programų naudojamą simbolių kodavimą, datos formatą, rūšiavimą ir t.t.)
`LOGNAME`	naudotojo prisijungimo vardas

PATH	dvitaškiais atskirtų katalogų, kuriuose ieškoma vykdomų komandų sąrašas, pvz.: /usr/bin:/usr/bin:/usr/contrib/bin:/usr/local/bin
SHELL	naudotojo *shell*'as (programos failo kelias) paleidžiamas pagal nutylėjimą, pvz.: /bin/sh
TERM	terminalo tipas (pvz.: xterm, vt100), pagal tipą programos gali sužinoti terminalo savybes (pvz.: spalvų palaikymas)
TMPDIR	katalogas, kuriame programos gali kurti laikinus/tarpinius failus
TZ	laiko zona

Kai jūs įvedate komandą, shell'as turi sugebėti surasti katalogą, kuriame yra ši programa (jei ta komanda nėra vidine shell komanda) ir ją paleisti. Aplinkos kintamajame PATH yra katalogų sąrašas, kuriuose *shell* turi ieškoti paleidžiamų komandų, jei nenurodytas paleidžiamos komandos kelias.

export - vidinė shell komanda aplinkos kintamųjų nustatymui

Naudotojas export komanda gali kurti aplinkos kintamuosius (pirma eilutė) ir pamatyti esamų sąrašą (antra eilutė):

```
export kintamasis[=reiksme]...

export -p
```

Pavyzdžiui:

```
$ a=areiksme

$ b=breiksme

$ export c=creiksme b

$ $SHELL

$ echo $a


$ echo $b

breiksme

$ echo $c

creiksme
```

Čia matome, kad naujame ($SHELL komanda paleistame) *shell*'e matomos tik exportuotų b ir c kintamųjų reikšmės, bet nematoma a reikšmė.

env - programa aplinkos kintamųjų nustatymui

Funkcionalumu `env` komanda artima aukščiau aprašytai `export`, tačiau tai visiskai atskira programa (ne *shell* vidinė komanda). Be to `env` skirta paleisti programoms su modifikuotais aplinkos kintamaisiais. Dabartinio *shell* procceso aplinka nemodifikuojama.

```
env [-i] [name=value]... [utility [argument...]]
```

Parinktys:

- `-i` – nustatyti tik nurodytus kintamuosius (kitų neperduoti paleidžiamai programai).

Paleidžiama `utility` programa su `argument...` argumentais ir pagal `env` parinktis ir `name=value` modifikuota aplinka. Nenurodžius argumentų – `env` išveda dabartinius aplinkos kintamuosius ir jų reikšmes.

Pavyzdžiai:

```
$ export LC_TIME=en_US.utf8

$ echo $LC_TIME

en_US.utf8

$ date

Thu 06 Feb 2020 07:43:20 PM EET

$ env LC_TIME=lt_LT.utf8 date

Kt vas.  6 19:43:43 EET 2020

$ echo $LC_TIME

en_US.utf8
```

Čia komanda `date` antrą kartą paleidžiama modifikavus jos aplinkos kintamąjį `LC_TIME` (todėl komanda pradeda "kalbėti lietuviškai"), tačiau originali *shell* aplinka lieka nemodifikuota.

Komandų grupavimas (sudėtinės komandos)

Visur, kur *shell*'as tikisi vienos komandos galima naudoti **sudėtines komandas**, t.y. komandų grupes apskliaustas `{}` arba `()`. Komandos grupėje gali būti atskirtos:

- `;`, nauja eilutė – nuosekliai vykdomos komandos;
- `|` – kanalai;
- `&&`, `||` – loginės išraiškos;
- `&` – fone paleidžiamos programos;

Sudėtinės komandos rezultatas (*exit code*) – paskutinės įvykdytos komandos grupėje rezultatas (fone paleidžiamų komandų rezultatas – 0).

Skirtumas tarp `{}` ir `()` tas, kad antru atveju apskliaustų komandų vykdymui paleidžiamas naujas *shell* procesas (šis procesas gauna savo aplinką, kuri sunaikinama procesui pasibaigus).

Pavyzdžiai

Tik antru atveju paleidžiamas naujas *shell* (PID=781865), kuris ir paleidžia `ps` komandą:

```
$ date; ps -l

Thu 06 Feb 2020 04:11:50 PM EET

F S   UID     PID    PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD

0 S  1446  781549  781548  0  80   0 -  2227 -      pts/14   00:00:00 bash

0 R  1446  781846  781549  0  80   0 -  1993 -      pts/14   00:00:00 ps

$ ( date; ps -l )

Thu 06 Feb 2020 04:12:39 PM EET

F S   UID     PID    PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD

0 S  1446  781549  781548  0  80   0 -  2227 -      pts/14   00:00:00 bash

1 S  1446  781865  781549  0  80   0 -  2227 -      pts/14   00:00:00 bash

0 R  1446  781867  781865  0  80   0 -  1994 -      pts/14   00:00:00 ps

$ { date; ps -l; }

Thu 06 Feb 2020 04:12:50 PM EET

F S   UID     PID    PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD

0 S  1446  781549  781548  0  80   0 -  2227 -      pts/14   00:00:00 bash

0 R  1446  781871  781549  0  80   0 -  1994 -      pts/14   00:00:00 ps
```

Funkcijos shell'e

Kaip ir kitose kalbose, funkcijos leidžia suskaldyti *shell* programos funkcionalumą į logines, mažesnes dalis. Funkcijos aprašas:

```
function_name() sudėtinė_komanda [ I/O_peradresavimas ]
```

Funkcija iškviečiama nurodant jos vardą, kaip komandą. Argumentai funkcijai perduodami lygiai taip pat, kaip ir komandai, t.y. išvardinami po funkcijos vardo atskirti tarpais arba *TAB* simboliais. Perduoti argumentai funkcijos viduje matomi, kaip poziciniai programos argumentai (`$*`, `$1`, `$2` ..., veikia `shift` komanda ir t.t.). Funkcijos matomos tik dabartiniame *shell* procese (į subshellus neperduodamos).

Kaip ir bet kokia kita programa/komanda, funkcija grąžina pabaigos kodą (*exit code*). Pagal nutylėjimą grąžinamas paskutinės vykdytos komandos pabaigos kodas. Pabaigos kodą galima nurodyti ir komanda:

```
return N
```

Čia `N` – sveikas skaičius. Vykdant `return` iš funkcijos išeinama grįžtant į pagrindinę programą ar iškvietusią funkciją ir grąžinant nurodytą kodą.

Pavyzdžiai

Funkcijos aprašymo ir iškvietimo pavyzdys:

```
#!/bin/sh


# Funkcijos aprasymas
Hello(){
   echo "Hello World"
}


# Funkcijos iskvietimas
Hello
```

Funkcijos iškvietimas su argumentais:

```
#!/bin/sh


# Funkcijos aprasymas
Hello () {
   echo "Hello, $1 $2"
}


# Funkcijos iskvietimas
Hello Vardas Pavarde
```

Vykdant šią procedūrą gausim:

```
$ ./procedura
Hello, Vardas Pavarde
```

Funkcijos grąžinama reikšmė:

```
#!/bin/sh


Hello () {
   return 10
}


Hello
```

```
#Paimam reiksmę kurią grąžino funkcija:

ret=$?



echo "grazinta reiksme: $ret"
```

Vykdant šią procedūrą gausim:

```
$ ./procedura

grazinta reiksme: 10
```

alias, unalias

```
alias [alias-name[=string]...]
```

Komanda `alias` kuriamos "pseudokomandos", kurias iškviečiant *alias*'o vardo `alias-name` vietoje įstatoma jam priskirta eilutė `string`. Iškvietus `alias` be argumentų spausdinamas aprašytų *alias*'ų sąrašas. Iškvietus tik su *alias*'o pavadinimu - spausdinamas to *alias*'o aprašas.

Komanda `unalias` sunaikina aprašytą *alias*'ą.

Sukurti *alias*'ai matomi tik dabartiniame *shell*'e (į subshellus neperduodami).

Pavyzdžiai:

```
$ alias ll="ls -l"

$ alias ll

alias ll='ls -l'

$ ll -d .

drwxr-xr-x 3 kespaul users 4096 Feb  6 16:18 .

$ unalias ll

$ alias ll

-bash: ll: command not found
```

set - shell elgsenos nustatymai

`set` komandos pagalba galima šiek tiek keisti *shell* (interaktyvios sesijos arba skripto) veikimą.

```
set [-abCefhmnuvx] [-o option] [argument...]

set [+abCefhmnuvx] [+o option] [argument...]

set -- [argument...]

set -o
```

```
set +o
```

Kai kurios parinktys (likusias galite rasti `sh(1)` arba POSIX standarte):

- `-e` – įvykus klaidai (komanda grąžino pabaigos kodą ne nulį) *shell*'o skriptas iškart pasibaigia. Ignoruoja pabaigos kodus salygose (`if`, `while`, `&&`, `||` ir pan.);
- `-n` – *shell* tik skaito komandas, bet jų nevykdo (tinka *shell* sintaksės tikrinimui);
- `-u` – sutikęs kreipinį į neaprašytą kintamąjį *shell*'as parodo klaidą ir baigia darbą;
- `-x` – trasuoja shell skripto vykdomas komandas (parodo, kas vykdoma).

Šiuos nustatymus galima įrašyti antraštėje, pvz.:

```
#!/bin/sh -e
```

arba nurodyti paleidžiant skriptą:

```
$ sh -x skriptas
```

Pavyzdžiai

Turime skriptą `testas` (paskutinėje eilutėje specialiai padaryta sintaksės klaida):

```
#!/bin/sh -u

false

echo $tokio_nera

echo Pabaiga

) klaida
```

Suveikia `-u` antraštėje, todėl nutrūksta trečioje eilutėje:

```
$ ./testas

./testas: line 3: tokio_nera: unbound variable
```

Sintaksės patikrinimas, bet komandos nevykdomos:

```
$ sh -n testas

testas: line 5: syntax error near unexpected token `)'

testas: line 5: `) klaida'
```

Nustatyta tik `-x` (antraštė ignoruojama), todėl vykdoma visa programa iki sintaksės klaidos:

```
$ sh -x testas

+ false

+ echo
```

```
+ echo Pabaiga

Pabaiga

testas: line 5: syntax error near unexpected token `)'

testas: line 5: `) klaida'
```

Nustatyta tik `-x` ir `-e`, vykdoma iki pirmos ne 0 grąžinančios komandos:

```
$ sh -xe testas

+ false
```

eval

Komanda `eval` įvykdo jai nurodytą eilutę, kaip *shell* komandą. Kitaip sakant, nurodytą eilutę *shell* interpretuoja du kartus, kiekvieną kartą atlikdamas specialiais simboliais nurodytus pakeitimus (įstatydamas kintamųjų reikšmes, aritmetinių operacijų rezultatus, `` ` `` kabutėmis pažymėtų komandų išvestus duomenis ir pan.)

```
eval [argument...]
```

Ši komanda naudojama norint apeiti *shell* interpretatoriaus apribojimus, pvz. kai skriptas suformuoja sudėtingą komandos eilutę (su peradresavimais, kreipiniais į kintamuosius ir pan.) ir reikia ją įvykdyti.

Pavyzdžiui:

Skriptas įvedantis kintamojo vardą iš klaviatūros ir atspausdinantis jo reikšmę:

```
#!/bin/sh

read varname

eval value=\$$varname

echo "$varname = $value"
```

Tarkim paleidus skriptą buvo įvesta `HOME`. Tada pirmas interpretatoriaus praėjimas pakeičia `eval value=\$$varname` į `value=$HOME` (suvalgo `\` ir įstato `$varname`), o antru praėjimu pakeičiama `value=/home/useris`.

Sekančiame pavyzdyje `eval` naudojamas tam, kad kintamasis `redirect` būtų interpretuojamas *shell*, o ne naudojamas kaip `ls` argumentas:

```
$ redirect='| cksum'

$ echo $redirect

| cksum

$ ls $redirect

ls: cannot access '|': No such file or directory
```

```
ls: cannot access 'cksum': No such file or directory

$ eval ls $redirect

4149648340 7
```

trap

Vidinė *shell* komanda `trap` leidžia nustatyti veiksmus, kurie bus vykdomi pasibaigus interaktyviai sesijai ar skriptui, arba gavus signalą.

```
trap n [condition...]

trap [action condition...]
```

Pirma sintaksės eilutė panaikina skaičiais nurodytų *trap*'ų nustatymus (t.y. jei pirmas argumentas skaičius). Antra eilutė nurodo, kad įvykus kuriam nors iš `condition` išvardintų *trap*'ų būtų vykdomas nurodytas veiksmas `action`. *Trap*'ai gali būti nurodyti skaičiais arba vardais:

N	vardas
0	EXIT
1	HUP
2	INT
3	QUIT
6	ABRT
14	ALRM
15	TERM

Įvykus nurodytam įvykiui `condition` vykdoma `eval action`.

Pavyzdys:

Įdėjus tokią eilutę į skriptą, skriptui tvarkingai (ne dėl `kill -9` ar panašiai) baigiantis (nesvarbu kurioj vietoj ir dėl ko jis pasibaigs) bus įvykdyta `env; times` komandos eilutė:

```
trap 'env; times' EXIT
```

Pavyzdžiui turime skriptą `trap.sh` (source:shell|trap.sh):

```
1 #!/bin/sh
2
3 mytrap(){
```

```
4           date
5           id
6   }
7
8   mybreak(){
9           echo "Programa nutraukta"
10  }
11
12  echo Pradzia
13  trap mytrap 0
14  trap mybreak 1 2 3 6 14 15
15  sleep 60
16  echo Pabaiga
```

Šį skriptą paleidus ir nutraukus su Ctrl-C (kol vykdo `sleep`) gauname:

```
$ ./trap.sh

Pradzia

^CPrograma nutraukta

Pabaiga

Thu 06 Feb 2020 04:39:33 PM EET

uid=1446(kespaul) gid=100(users) groups=100(users),101(dest)
```

Matome, kad nutraukiant programą su Ctrl-C buvo iškviestas `mybreak` *trap*'as, o pasibaigus programai - `mytrap` *trap*'as.

`.` - failo vykdymas dabartiniame shell'e

```
. file
```

Komanda `.` skaito ir vykdo komandas iš nurodyto failo `file` dabartiniame *shell* procese. Kitaip sakant, poveikis panašus, kaip kitų programavimo kalbų *include* komandų, t.y. nurodyto failo turinys įstatomas komandos vietoje. Šitaip vykdomas failas gali keisti dabartinio *shell* proceso kintamuosius, funkcijas, *alias*'us, elgsenos nustatymus ir pan.

Pavyzdys:

Turime failą `dottest.sh`:

```
a=testas

test1(){ echo Hello; }

$ . ./dottest.sh
```

```
$ echo $a

testas

$ test1

Hello
```

exec - shell proceso pakeitimas

Komanda `exec` gali atidaryti/uždaryti failus (šitos galimybės čia neaptarsim) arba pakeisti *shell* procesą nauja komanda `command` su argumentais `argument...` .

```
exec command [argument...]
```

Proceso pakeitimas reiškia, kad po šios komandos įvykdymo dabartinio *shell* skripto (ar interaktyvaus *shell*) proceso programa pakeičiama kita ir jai pasibaigus grįžtama nebe į `exec` paleidusį skriptą, o į dabartinio proceso tėvą.

Pavyzdžiui:

Turime failą `exectest.sh`:

```
#!/bin/sh

echo Pradzia

exec ls -ld .

echo Pabaiga

$ ./exectest.sh

Pradzia

drwxr-xr-x 3 kespaul users 4096 Feb  6 16:42 .
```

Atkreipkite dėmesį – paskutinė skripto eilutė, turėjusi spausdinti `Pabaiga` neįvykdoma, nes įvykdžius `exec` vykdomas nebe `exectest.sh` skriptas, o `ls -ld .` programa.

Pradiniai nustatymai

Paleidžiamas *shell*'as gali nuskaityti ir įvykdyti kai kuruos sistemoje esančius failus (pradinių nustatymų failus). POSIX standartas reikalauja, kad *shell*'as startuodamas įvykdytų kintamąjame `ENV` įrašytame failo varde esančias komandas. Kol kas tai retai naudojama, o *shell*'ai turi kietai įrašytus failų vardus, kuriuos įvykdo.

Nustatymų failai paprastai naudojami norint modifikuoti *shell* aplinką: kintamuosius, alias ir pan., o taip pat atlikti spe cifinius veiksmus (pvz.: atspausdinti kokį nors pranešimą ar pan.).

bash pradiniai nustatymai

Čia login *shell*'u vadinamas pirmas shell procesas, kurį paleidžia OS naudotojui prisijungus prie sistemos (sukūrus naują prisijungimo sesiją). `bash` gali elgtis, kaip login *shell*'as, jei paleidžiamas su `--login` parinktimi.

- Startuodams `bash` login *shell*'as ieško ir bando įvykdyti `/etc/profile`, o po to ieško `~/.bash_profile`, `~/.bash_login`, `~/.profile` ir vykdo tą, kurį pirmą suranda. Pasibaigdamas login *shell*'as vykdo `~/.bash_logout`.
- Startuodamas interaktyvus ne login *shell*'as ieško ir vykdo `~/.bashrc`
- Startuodamas neinteraktyvus ne login *shell*'as bando vykdyti `BASH_ENV` kintamajame nurodytą failą.

Užduotys

shell modulinė organizacija

- Ar pakeitus aplinkos kintamojo `HOME` reikšmę keičiasi `cd` komandos veikimas?

  Keičiasi namų katalogas, kuriame galime atsirasti paraše komandą cd, todėl atsiduriame kitame kataloge jeigu keičiame kintamojo HOME reikšmę. Nerastume kai kurių reikšmių mūsų failų sistemoje.

- Kas bus jei ištrinsite `PATH` reikšmę? Kaip dabar paleisti komandas?

  Naudoti komandas darbiniame kataloge nurodant santykinių/absoliutinių kelių iki failo. Pilnas direktorijas nurodyti nebus patogu.

- Modifikuokite `PATH`, kad iškviečiant `grep` būtų naudojama `/data/ld/ld2/4/grep`, o ne `/bin/grep`. Kaip tai patikrinti?

  Naudojame komandą `export, galime pakeisti arba naudoti env komandą.`

  ```
  export PATH=/data/ld/ld2/4:$PATH

  which grep komanda
  ```

- Modifikuokite savo aplinką taip, kad iškviečiant `more` būtų iškviečiama `less` komanda. Atlikite tai dviem būdais:
  - kad pakeitimas galiotų tik esamoje sesijoje (nemodifikuojant jokių failų);
  - kad pakeitimas galiotų visose naujose sesijose (modifikuojant *shell* nustatymų failus).

1 variantas **alias more=less**

2 variantas **/.bashrc >> alias more=less**

- Kaip nenaudojant `alias` pasiekti, kad *shell* įvedus komandą `tst` būtų vykdoma programa `/usr/bin/id` (galite kurti ir modifikuoti failus/katalogus, bet nebūtina kad pakeitimas galiotų visose naujose sesijose)?

  Sukurti nauja scriptą **`/usr/local/bin/id`**

  `#!/bin/sh`

  `export PATH=/usr/local/bin:$PATH`

- Kodėl paleidus sekančias dvi komandų eilutes skiriasi jų rezultatai?

```
• $ cd; pwd; ( cd /; pwd; ); pwd

• $ cd; pwd; { cd /; pwd; }; pwd
```

Viena užrašoma (), kita {} , vienas pilnai aprašo subshella, kita komandų grupę, pirmoje nėra keičiama darbinis katalogas, antrame kataloge keičiamas darbinis katalogas

- Kodėl išvedant a apskliaudus () gaunama priskirta reikšmė, o paleidus naują *shell* ($SHELL) ji dingsta?

```
• $ a=areiksme

• $ ( echo $a )

• areiksme

• $ $SHELL

• $ echo $a

•

• $ exit

• exit

• $ ( echo $a )

• areiksme
```

Mes ne exportuojame šios komandos į mūsų shellą, todėl jinai dingsta, pradedant naują Shell, komanda neprisimena šios reikšmės.

- sudarykite funkciją, kuri formuotų "žodyną": patikrintų, ar jai argumentais nurodytų žodžių dar nėra žodyno faile, ir jei nėra - failą papildytų;

  #!/bin/sh

  # Define the function to form the dictionary

  form_dictionary() {

  dictionary_file="dictionary.txt"

```
# Create the dictionary file if it doesn't exist

if [ ! -f "$dictionary_file" ]; then

touch "$dictionary_file"

fi


# Loop through the arguments and add to the dictionary if not already present

for word in "$@"; do

if ! grep -q "^$word$" "$dictionary_file"; then

echo "$word" >> "$dictionary_file"

fi

done

}


# Test the function

form_dictionary apple banana cherry apple grape


# Display the contents of the dictionary file

cat dictionary.txt
```

- parašykite pagrindinę programą, kuri iš jai per *stdin* paduodamų duomenų formuotų žodyną.

```
#!/bin/bash


# Define the function to form the dictionary

function form_dictionary() {

  dictionary_file="dictionary.txt"


  # Create the dictionary file if it doesn't exist

  if [ ! -f "$dictionary_file" ]; then
```

```
    touch "$dictionary_file"
  fi

  # Loop through the input and add to the dictionary if not already present
  while read -r word; do
    if ! grep -q "^$word$" "$dictionary_file"; then
      echo "$word" >> "$dictionary_file"
    fi
  done
}

# Call the function to form the dictionary from stdin
form_dictionary <&0

# Display the contents of the dictionary file
cat dictionary.txt
```

Paduodame stdin per echo

```
lukkuz1@oslinux ~/lab2_4/scriptai $ >stdin
lukkuz1@oslinux ~/lab2_4/scriptai $ chmod +x stdin
lukkuz1@oslinux ~/lab2_4/scriptai $ nano stdin
lukkuz1@oslinux ~/lab2_4/scriptai $ echo "obuolys"
obuolys
lukkuz1@oslinux ~/lab2_4/scriptai $ echo "obuolys" | ./stdin
apple
banana
cherry
grape
```

obuolys

lukkuz1@oslinux ~/lab2_4/scriptai $

- ką daro ši programa (kaip ja naudotis)?

```
• #!/bin/sh
• dir=`pwd`
• for i in * ; do
•     if test -d $dir/$i ; then
•         cd   $dir/$i
•         while echo “$i:”; read   x; do
•             eval $x
•         done
•         cd ..
•     fi
• Done
```

Apibendrinant galima pasakyti, kad scriptas pereina per visus dabartinio katalogo pakatalogius ir kiekviename pakatalogyje vykdo komandas nuo standartinės įvesties, kol vartotojas išeina iš ciklo. Tai gali būti naudinga atliekant pasikartojančias užduotis keliuose pakatalogiuose, pvz., pervadinant ar perkeliant failus.

- sukurkite programą, kuri einamajame kataloge ir visuose jo pakatalogiuose formuotų `index.txt` failus, juose kiekvienoje eilutėje įrašydama kiekvieno failo vardą ir aprašymą atskirtus `/` . Failo aprašymas įvedamas klaviatūra. Kelis kartus leidžiant programą, jau suvesti failų aprašymai neturi būti pametami (programa turi tik atnaujinti/papildyti `index.txt` failus).

```
#!/bin/bash

# Define the function to create/update index.txt files
function create_index_files() {
  for file in *; do
    if [ -d "$file" ]; then
      # Recurse into subdirectories
      cd "$file"
      create_index_files
      cd ..
    elif [ "$file" != "index.txt" ]; then
      # Create/update index.txt in current directory
```

```
      description=""
      if [ -f "index.txt" ]; then
        # Read existing description from index.txt
        description=$(sed -n '1p' "index.txt" | cut -d'/' -f2)
      fi
      echo "Enter a description for $file:"
      read -r new_description
      echo "$file/$new_description" > "index.txt.tmp"
      if [ -n "$description" ]; then
        # Remove existing description from index.txt
        sed -i '1d' "index.txt"
      fi
      # Move index.txt.tmp to index.txt and append existing description
      cat "index.txt.tmp" "index.txt" > "index.txt.new"
      mv "index.txt.new" "index.txt"
      rm "index.txt.tmp"
    fi
  done
}

# Call the function to create/update index.txt files
create_index_files
```

- papildykite programą, kad surastų ir pašalintų perteklinius failų aprašymus (failo nebėra, o aprašymas vis dar yra).

```
#!/bin/sh


# Define the function to create/update index.txt files and remove redundant descriptions

create_index_files() {

  # Initialize array to hold existing descriptions

  existing_descriptions=()



  for file in *; do

    if [ -d "$file" ]; then

      # Recurse into subdirectories

      cd "$file"

      create_index_files

      cd ..
```

```
  elif [ "$file" != "index.txt" ]; then
    # Create/update index.txt in current directory
    description=""
    if [ -f "index.txt" ]; then
      # Read existing description from index.txt
      description=$(sed -n '1p' "index.txt" | cut -d'/' -f2)
      # Add existing description to array
      existing_descriptions+=("$description")
    fi
    echo "Enter a description for $file:"
    read new_description
    echo "$file/$new_description" > "index.txt.tmp"
    if [ -n "$description" ]; then
      # Remove existing description from array if file still exists
      if [ -f "$file" ]; then
        existing_descriptions=("${existing_descriptions[@]/$description}")
      # Remove existing description from index.txt if file no longer exists
      else
        sed -i '1d' "index.txt"
      fi
    fi
    # Move index.txt.tmp to index.txt and append existing description
    cat "index.txt.tmp" "index.txt" > "index.txt.new"
    mv "index.txt.new" "index.txt"
    rm "index.txt.tmp"
  fi
done


# Remove redundant descriptions from index.txt
```

```
  for description in "${existing_descriptions[@]}"; do
    if [ -n "$description" ]; then
      # Remove redundant description from index.txt
      sed -i "/\/$description\$/d" "index.txt"
    fi
  done
}

# Call the function to create/update index.txt files
create_index_files
```

Kūrybinės užduotis ir kolio pavyzdžiai

SSH prisijungimai

Užduotis: SSH bandymai laužtis

- `/data/ld/ld2/studlog` suraskite IP adresus iš kurių buvo bandyta laužtis (spėlioti ssh loginai/passwordai).

```
#!/bin/sh

grep "Failed password" /data/ld/ld2/studlog | awk '{print $15}' | sort -u
```

- Suraskite pataikymus į naudotojo vardą (t.y. IP adresas jungėsi keliais neegzistuojančiais loginais, bet tarp jų yra bandytas ir sistemoje registruotų naudotojų loginų).

```
#!/bin/sh

grep "Failed password" /data/ld/ld2/studlog | awk '{print $15" "$16}' | sort | uniq -c | awk '$1 > 1 {print $2}' | sort -u
```

- Parodykite tokių loginų sąrašą.

```
#!/bin/sh



# Search for IP addresses with failed login attempts

ip_list=$(grep -oP 'Failed password for \K(.+)(?= from)' /data/ld/ld2/studlog | sort | uniq -d)



# Loop through IP addresses and search for usernames

for ip in $ip_list

do

  user_list=$(grep "$ip" /data/ld/ld2/studlog | grep -oP 'Failed password for \K(.+)(?= from)' | sort | uniq -d)

  echo "IP address: $ip"

  echo "Usernames: $user_list"

done
```

Užduotis: SSH galbūt atspėti passwordai

- `/data/ld/ld2/studlog` suraskite IP adresus, iš kurių buvo ir sėkmingų ir nesėkmingų bandymų jungtis per SSH.

```
#!/bin/bash


# Find all unique IP addresses in the studlog file
ips=$(grep -Eo "[0-9]+\.[0-9]+\.[0-9]+\.[0-9]+" /data/ld/ld2/studlog | sort -u)


# Iterate through each IP address
for ip in $ips
do
  # Check if the IP has both successful and failed attempts
  success=$(grep -c "Accepted" /data/ld/ld2/studlog)
  failed=$(grep -c "Failed" /data/ld/ld2/studlog)


  echo "Success ip's" $success
  echo "Failed ip's" $failed
done
```

- išvesti IP sąrašą ir prie kiekvieno "praėjusių" ir "nepraėjusių" loginų sąrašą.

```
#!/bin/sh

# Get a list of unique IP addresses from the studlog file
ips=$(grep -E -o "([0-9]{1,3}[\.]){3}[0-9]{1,3}" /data/ld/ld2/studlog | sort -u)

# Loop through the list of IP addresses and check for successful and failed attempts
for ip in $ips; do
   # Count the number of successful attempts
   num_passed=$(grep "$ip.*Accepted" /data/ld/ld2/studlog | wc -l)

   # Count the number of failed attempts
```

```
    num_failed=$(grep "$ip.*Failed" /data/ld/ld2/studlog | wc -l)

    # Output the IP address and whether or not there were successful attempts
    if [ $num_passed -gt 0 ]; then
        echo "$ip passed"
    else
        echo "$ip not passed"
    fi
done
```

Užduotis: klasėj dirbantys useriai

- iš `last` komandos rezultatų suraskite nurodytame laiko intervale iš klasės dirbusių userių sąrašą.

```
#!/bin/bash

read -p "Enter the start time (in format YYYY-MM-DD HH:MM): " start_time

read -p "Enter the end time (in format YYYY-MM-DD HH:MM): " end_time

# Get the login history using the last command and filter it based on the given time interval

login_history=$(last | grep -E "(\b${start_time}\b.*\b${end_time}\b)")

# Extract the usernames from the login history

users=$(echo "$login_history" | awk '{print $1}' | sort | uniq)

# Output the list of users who worked during the specified time interval

echo "Users who worked from ${start_time} to ${end_time}:"

echo "$users"
```

- išveskite sąrašą: loginas IP Vardas Pavarde

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

```
# Get user input for time interval

echo "Enter start date in format YYYY-MM-DD:"

read start_date

echo "Enter end date in format YYYY-MM-DD:"

read end_date


# Get all logins during time interval from last command

logins=$(last | grep -E "$start_date|$end_date" | awk '{print $1}' | sort | uniq)


# Get user information from /etc/passwd file and join with logins

while IFS=: read -r login _ uid gid name home shell; do
   if [[ "$logins" == *"$login"* ]]; then
      ip=$(last | grep "$login" | awk 'NR==1{print $3}')
      surname=$(echo "$name" | awk -F' ' '{print $NF}')
      name=$(echo "$name" | awk '{$NF=""; print $0}' | sed 's/ *$//')
      echo "$login,$ip,$name,$surname"
   fi
done < /etc/passwd
```

- [advanced]: jei yra loginų, kurie jungėsi iš kelių klasės IP spausdinkite perspėjimą ir IP adresus.

WWW

Užduotis: populiariausi bandymai "laužtis"

- `/data/ld/ld1/Solaris_access_log` suraskite per kokius puslapius dažniausiai bandoma laužtis

```
#!/bin/bash


# Extract only the URLs from the log file
```

```
urls=$(awk -F\" '{print $2}' /data/ld/ld1/Solaris_access_log)


# Sort the URLs and count how many times each URL appears

sorted_urls=$(echo "$urls" | sort | uniq -c | sort -rn)


# Print the top 10 most frequently hacked pages

echo "$sorted_urls" | head -n 10
```

- išveskite puslapių sąrašą ir kiek kartų bandyta jį pasiekt

```
#!/bin/sh

# extract page access logs and sort them by frequency
awk '{print $7}' /data/ld/ld1/Solaris_access_log | sort | uniq -c | sort -rn
```

- [advanced] prie kiekvieno puslapio išveskite IP adresų sąrašą iš kurių bandyta jį pasiekt.

```
#!/bin/sh


# specify the access log file

access_log="/data/ld/ld1/Solaris_access_log"


# extract the requested data from the access log file

awk '{print $7}' "$access_log" | sort | uniq -c | sort -nr | while read count page

do

  echo "Page accessed $count times: $page"

  echo "-----------------------------------"

  grep "$page" "$access_log" | awk '{print $1}' | sort | uniq -c | sort -nr | awk
'{printf("%8s %s\n", $1, $2)}'

  echo ""

done
```

Užduotis: ėdriausi klientai

- iš `/data/ld/ld1/Solaris_access_log` suraskite kiek kuris IP nusiurbė iš WWW serverio.

```
#!/bin/bash


LOG_FILE="/data/ld/ld1/Solaris_access_log"


# Extract IP addresses and bytes transferred from log file

grep -Eo '([0-9]{1,3}\.){3}[0-9]{1,3}.*/[A-Z]* [0-9]* ' "$LOG_FILE" | awk '{print $1,$NF}' > temp1.txt


# Sum bytes transferred for each unique IP address

awk '{a[$1]+=$2} END {for(i in a) print i, a[i]}' temp1.txt | sort -k2rn


# Remove temporary file

rm temp1.txt
```

- išveskite sąrašą IP adresų ir kiekvieno jų nusiurbtos informacijos kiekį.

  ```
  #!/bin/bash

  # Define log file path
  LOG_FILE="/data/ld/ld1/Solaris_access_log"

  # Extract IP addresses and bytes transferred for each request
  grep -oE "^[0-9]+\.[0-9]+\.[0-9]+\.[0-9]+" "$LOG_FILE" | \
    awk '{ip_count[$0]++;}

        # Extract bytes transferred from each line
        {getline; bytes=$NF;}

        # Sum bytes transferred for each IP
        END {for (ip in ip_count) printf("%s %d\n", ip, bytes*ip_count[ip]);}' | \
    sort -nrk 2
  ```

Užduotis: atkakliausi (bukiausi) įsilaužėliai

- iš `/data/ld/ld1/Solaris_access_log` suraskite 10 IP adresų, kurie daugiausia bando laužtis (kreipiasi į įvairius puslapius ir gauna klaidas).

  ```
  #!/bin/bash


  LOG_FILE="/data/ld/ld1/Solaris_access_log"
  ```

 # filter log for errors and hacking attempts, then count occurrences by IP

**awk '/(404|500|501|502|503|504|user) / {count[$1]++} END {for (ip in count) print ip, count[ip]}' "$LOG_FILE" **

**| sort -rnk 2 **

| head -n 10

- išveskite IP sąrašą ir kiekvienam jų puslapių sąrašą į kuriuos jie kreipėsi (ir kur gavo klaidas, ir kur viskas gerai).

```
#!/bin/bash


# define log file path

LOG_FILE="/data/ld/ld1/Solaris_access_log"


# extract IP addresses and pages accessed

grep -E "([4-5][0-9]{2})" "$LOG_FILE" | awk '{print $1 " " $7}' | sort | uniq -c | sort -nr | head -n 10 | awk '{print $2}' > top_ips.txt


# loop through each IP and output the pages accessed

while read -r ip; do

   echo "IP address: $ip"

   grep "$ip" "$LOG_FILE" | awk '{print $7 " " $9}'

   echo ""

done < top_ips.txt


# remove temporary file

rm top_ips.txt
```