Terminaali tutuksi

Linux ja komentorivin hallinta

Lappeenrannan teknillinen yliopisto 2015
Annika Ikonen, Timo Hynninen ja Erno Vanhala

Tarkastus ja oikoluenta:
Jouni Ikonen, Jussi Kasurinen ja Uolevi Nikula

Lappeenrannan teknillinen yliopisto

LUT School of Business and Management

Innovation and Software

PL 20

53851 Lappeenranta

Lappeenrannan teknillinen yliopisto
Lappeenranta University of Technology

LUT School of Business and Management
Innovation and Software

LUT Scientific and Expertise Publications
Oppimateriaalit – Lecture Notes 7

Annika Ikonen, Timo Hynninen ja Erno Vanhala

Terminaali tutuksi: Linux ja komentorivin hallinta

ISBN 978-952-265-773-2
ISBN 978-952-265-774-9 (PDF)
ISSN-L 2243-3392
ISSN 2243-3392

Lappeenranta 2015

Perustuu teokseen : **UNIX Tutorial for Beginners** (Michael Stonebank) osoitteessa: http://www.ee.surrey.ac.uk/Teaching/Unix/

Table of Contents

Luku 0: Johdanto

“Tämä artikkeli käsittelee Linux-käyttöjärjestelmää. Linux tarkoittaa myös käyttöjärjestelmän ydintä. 9885 Linux on asteroidi.” - Suomenkielinen Wikipedia sivulla Linux.

UNIX on 1960 -luvulla kehitetty käyttöjärjestelmä, joka on sittemmin ollut jatkuvassa kehityksessä. UNIX on vakaa, monelle käyttäjälle ja moniajoon soveltuva järjestelmä palvelimille, pöytäkoneille sekä kannettaville. Linux puolestaan on UNIX-klooni, eli käyttöjärjestelmä, jonka perustoiminnot vastaavat UNIXin toimintoja ja ovat yhteensopivia toistensa kanssa.

Sekä UNIX- että Linux-käyttöjärjestelmät ovat laajalti käytössä. Niiden vahvuus korostuu erityisesti palvelinkäytössä, missä Linux onkin ylivoimaisesti suosituin käyttöjärjestelmä. Linux lieneekin nykyään maailman yleisin käyttöjärjestelmä, sillä palvelimien lisäksi myös useat mobiililaitteet pohjautuvat Linuxiin (Android) samoin kuin useimmat sulautetut järjestelmät – Microsoft Windows on edellä ainoastaan työpöytäkäytössä.

Tämä opas käsittelee komentorivin ja komentorivityökalujen käyttöä UNIX- ja Linux-käyttöjärjestelmissä. Oppaassa käsiteltävien asioiden tiimoilta ei ole merkityksellistä puhutaanko Unixista vai Linuxista, sillä kaikkien esimerkkien pitäisi kumpaan tahansa perheeseen kuuluvassa käyttöjärjestelmässä toimia identtisesti. Varsinaisissa sisältökappaleissa puhutaankin johdonmukaisuuden takia vain Unixista, mutta Unixin tilalle voisi mihin kohtaa vain vaihtaa Linuxin.

Huomionarvoista on myös, ettei nykyään UNIX tai sen kummemmin Linuxkaan ole käyttöjärjestelmä. Vaikka UNIX-käyttöjärjestelmä on/oli kyllä olemassa, Unixiksi nimitettävät käyttöjärjestelmät, kuten esimerkiksi FreeBSD, Solaris tai OS X, ovat alkuperäisestä UNIXista polveutuvia uusia käyttöjärjestelmiä. Linux puolestaan on avoin käyttöjärjestelmän ydin (eli kerneli). Yleisesti puhuttaessa Linux-käyttöjärjestelmästä pitäisi käyttää käyttöjärjestelmäperheen koko nimeä, GNU/Linux, sillä käyttöjärjestelmä muodostuu Linux-ytimestä sekä GNU-projektin varusohjelmista. GNU/Linux -ohjelmistokokoelmia on erilaisia, ja eri kehittäjien omia GNU/Linux -käyttöjärjestelmäkokonaisuuksia nimitetään Linux-jakeluiksi eli distribuutioiksi.

0.1 Unixin tausta

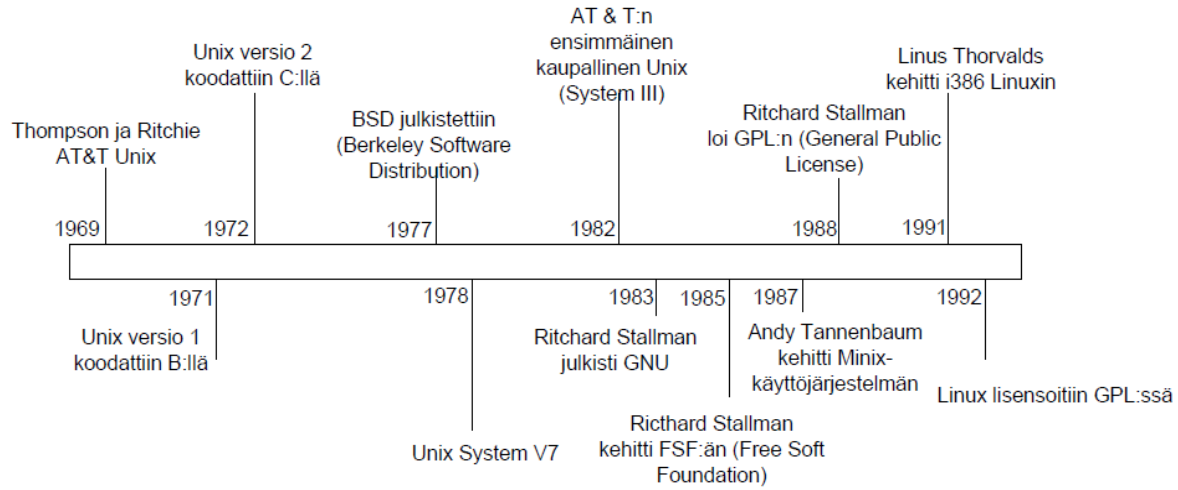
Tietotekniikan aamuhämärässä, kun tietokoneita ei ollut vielä jokaisen ihmisen taskussa vaan lähinnä armeijan ja yliopistojen käytössä, kehitettiin Yhdysvalloissa yleispätevä käyttöjärjestelmä, jonka nimeksi tuli Unix. Samassa yhteydessä aiemmin käytetystä B-ohjelmointikielestä jalostettiin C-kieli. Molemmat olivat parannuksia aiempiin ratkaisuihin nähden. UNIX oli siirrettävissä alustoille, joilla pystyi ohjelmoimaan C:llä ja C oli entistä helpommin omaksuttava ohjelmointikieli.

Unix oli kaupallinen käyttöjärjestelmä, mutta myöhemmin 80-luvulla tietokoneiden jo yleistyttyä Richard Stallman halusi tarjota ihmisille avoimen käyttöjärjestelmän, joka olisi kuitenkin yhtä hyvä kuin Unix ja yhteensopiva sen kanssa. Projekti sai nimekseen GNU (GNU’s not Unix) ja ihmiset työstivät GNU manifeston mukaisesti avoimia ohjelmistoja

korvaamaan kaupallisia suljettuja tuotteita, kuten kääntäjiä, pitkin 80-lukua. Projektilta puuttui kuitenkin se kaikkein tärkein eli käyttöjärjestelmän ydin - kernel.

Myös kernelin rakentaminen aloitettiin Stallmanin ohjauksessa ja se valmistui 90-luvun puolivälissä. Projekti kesti kuitenkin niin pitkään, että muut ehtivät havaita saman ongelman ja mm. Linus Torvalds aloitti oman projektinsa Helsingin yliopiston suojissa. Torvalds oli ollut tekemisissä tietokoneiden kanssa lapsuudestaan lähtien ja yliopistoon päästyään hän keräsi rahaa ostaakseen Intelin 80386-arkkitehtuurilla (nykyään i386) varustetun PC:n. Koneen mukana tuli Microsoftin MS-DOS käyttöjärjestelmä, mutta se oli Torvaldsin mielestä aivan liian rajoittunut ja Torvalds käytti sitä vain Prince of Persia –pelin pelaamiseen. Torvalds tilasi koneeseensa Andrew Tanenbaumin luoman MINIX-käyttöjärjestelmän ja huomasi, että MINIXin ja GNUn välissä olisi tilaa uudelle kernelille. Kernelin tärkeimpiä lähtökohtia oli POSIX-yhteensopivuus, sillä POSIX-standardi määrittelee Unix-käyttöjärjestelmien perustoimintoja kuten tiedosto-operaatiot, signaalit sekä putket ja näin ollen POSIX-yhteensopivia ohjelmistoja pystyisi käyttämään suoraan uudessa kernelissä. Torvaldsille selvisi kuitenkin pian, että standardit ovat maksullisia eikä hänen opintotukeen perustuva elämänvaihe mahdollistanut niiden ostamista. Torvalds ratkaisi tämän ongelman etsimällä käsiinsä yliopiston SunOS Unix-koneen, sen käsikirjan ja käytti näitä referenssinä luodessaan Linuxin eli käyttöjärjestelmän kernelin. Linux ei siis ole teknillisesti Unix, mutta käytännössä Linux tekee kaiken mitä muutkin Unixit, sillä sen määritykset pohjautuvat BSD-Unixiin. Unix-järjestelmistä on saatavana myös kaupallisia versioita (esim. Solaris), mutta ne tuntuvat olevan väistymässä avoimen lähdekoodin Unix-järjestelmien tieltä. Näistä esimerkkinä erilaiset BSD:t: FreeBSD, NetBSD ja OpenBSD. On myös syytä huomata, että Mac OS X perustuu Unixiin.

Käsitteisiin liittyen on hyvä huomata, että Linux viittaa periaatteessa vain kerneliin kun taas GNU/Linux viittaa koko käyttöjärjestelmään. Puhekielessä Linuxilla viitataan koko käyttöjärjestelmään. Kuvassa 1 näkyy Unix-GNU/Linux –järjestelmien tärkeimmät kehitysvaiheet alkaen vuodesta 1969 päättyen Linuxin julkistamiseen.



Kuva 1: Unix GNU/Linux aikajana (Robot Wisdom 2011).

0.2 Ubuntu Linux

Ubuntu on avoimesta lähdekoodista koostuva Linux-käyttöjärjestelmä, joka pohjautuu Debian-jakeluun ja sisältää haluttaessa myös suljetun koodin ohjelmistoja kuten Skypen, Adobe Flashin tai suljetut näytönohjainajurit. Ubuntu sisältää kaikki peruskäyttöön tarvittavat ohjelmat, kuten esim. tekstinkäsittelyn, taulukkolaskennan, sähköpostiohjelman ja nettiselaimen. Ohjelmia on helppo asentaa lisää joko komentoriviltä tai pakettienhallintatyökalulla. Graafisen asennusohjelman avulla käyttöjärjestelmän asennus on ongelmatonta ja sujuu nopeasti. Ubuntu-projekti on sitoutunut noudattamaan vapaan ohjelmistokehityksen periaatteita.

Ubuntu on myös pelaajalle parhaiten sopiva Linux-jakelu, koska sen käyttäjäkunta on niin laaja, että ohjeita ja tukea löytyy. Esimerkiksi Valve tarjoaa Steam-palveluaan Ubuntulle.
Yliopistomme Linux-luokassa on käytössä Ubuntu-käyttöjärjestelmä ja kirjauduttaessa koneelle sisään käyttäjä voi valita työpöytäympäristökseen Unityn, KDE:n, Xfce:n tai LXDE:n.

Luku 1: UNIXia aloittelijoille

1.1 UNIX -käyttöjärjestelmä

Käyttöjärjestelmä muodostuu kolmesta osasta; ytimestä, komentorivistä ja ohjelmista.

Ydin

UNIXin ydin on käyttöjärjestelmän keskus: se jakaa aikaa ja muistia ohjelmille sekä hoitaa tiedostojen varastoinnin ja tiedonsiirron järjestelmän käskyjen mukaan.

Esimerkkinä komentorivin ja ytimen yhteistyöstä oletetaan, että käyttäjä kirjoittaa **rm tiedosto** komentoriville (mikä toimii poistaen tiedoston *tiedosto*). Komentorivi etsii tiedostovaraston tiedostolle, joka sisältää ohjelman **rm**, ja sen jälkeen pyytää järjestelmäkutsujen avulla ydintä panemaan täytäntöön ohjelman **rm** tiedostolle *tiedosto*. Kun prosessi (tiedoston poistaminen) on valmis, komentorivi palauttaa kehotteen $ käyttäjälle osoittaakseen odottavansa uusia käskyjä.

Komentorivi

Komentorivi toimii käyttöliittymänä käyttäjän ja ytimen välillä. Kun käyttäjä kirjautuu sisään, kirjautumisohjelma tarkistaa käyttäjänimen ja salasanan jonka jälkeen se aloittaa seuraavan ohjelman eli komentotulkin. Komentotulkista (CLI, command line interpreter) käytetään usein myös englanninkielistä nimitystä *shell*. Se tulkkaa käyttäjän kirjoittamat komennot ja järjestää ne suoritettavaksi. Komennot ovat itsessään ohjelmia; kun ne niiden suoritus on loppunut, komentotulkki palauttaa käyttäjälle kehotteen.

Myös komentotulkki itse on ohjelma. Se käyttää Unixin sisäänrakennettuja syöttö- ja tulostustoimintoja ottaakseen käyttäjältä vastaan syötteitä ja tulostaakseen merkkejä käyttäjän nähtäville. Komentotulkkia ajetaan *päätteessä*, eli laitteessa, mikä mahdollistaa tietokoneen käyttämisen. Graafisessa ympäristössä päätteen virkaa hoitaa *terminaaliemulaattoriohjelma*, joka mahdollistaa komentorivin käytön graafisen ikkunointiympäristön päällä.

Koska komentotulkkikin on ohjelma, voi käyttäjä ajaa samassa järjestelmässä useita eri komentotulkkeja. Yliopistolla on oletuksena käytössä *bash* (Bourne Again Shell), mutta Linux-koneisiin on asennettuna myös zsh (Z Shell). Muita variantteja komentotulkeista on mm. alkuperäisessä UNIXissa (versiossa 7) käytetty *sh* (Bourne Shell), *csh*, *ash* ja *dash*.

Eri komentotulkit poikkeavat ominaisuuksiltaan, joten Unix-käyttäjä voikin valita itselleen mieleisen komentorivikäyttöliitymän. Komentotulkkia voi myös kustomoida omien tarpeidensa mukaan. Perustoiminnallisuudet, kuten tiedostonimen täydennys tai komentohistorian selaus, toimivat kuitenkin lähestulkoon samalla tavoin kaikissa komentotulkeissa.

Tiedostonimen täydennys - Kirjoittamalla osan komennon, tiedoston tai kansion nimestä ja painamalla sarkainnäppäintä (tabulaattoria, tabia), komentorivi osaa täydentää loput

automaattisesti. Jos komentotulkki löytää useamman kuin yhden samalla merkkijonolla alkavan nimen, se listaa osumat, jolloin lisäämällä muutaman kirjaimen ja painamalla tabia uudelleen saa haun tarkennettua oikeaan kohteeseen.

Historia - Komentotulkki pitää kirjaa aikaisemmin käytetyistä komennoista. Jos komento tarvitsee toistaa, nuolinäppäimiä (nuoli ylöspäin, nuoli alaspäin) käyttämällä voi selata komentohistoriaa ylös- ja alaspäin.

1.2 Tiedostot ja prosessit

Kaikki UNIXissa on joko tiedostoja tai prosesseja. Prosessi on suoritettava ohjelma, joka on tunnistettavissa yksilöllisen PID:n (process identifier) avulla.

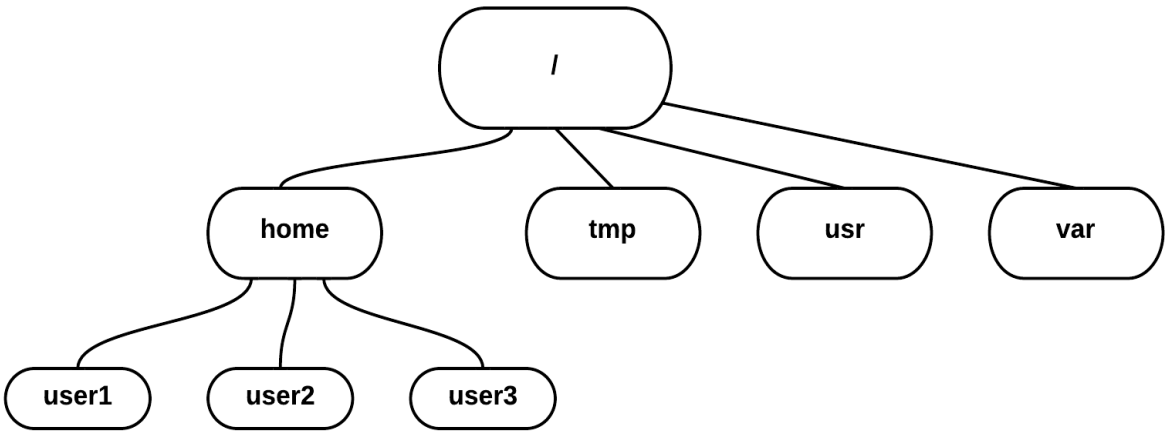
Tiedosto on kokoelma dataa. Käyttäjät luovat niitä käyttäen tekstieditoreja, tulkkeja jne.

Esimerkkejä tiedostoista:

- Asiakirja (raportti, essee jne)
- Tietyn ohjelman koodi
- Hakemisto (hakemistokin todella on vain erikoismääreen saanut tiedosto)

Hakemistojärjestelmä

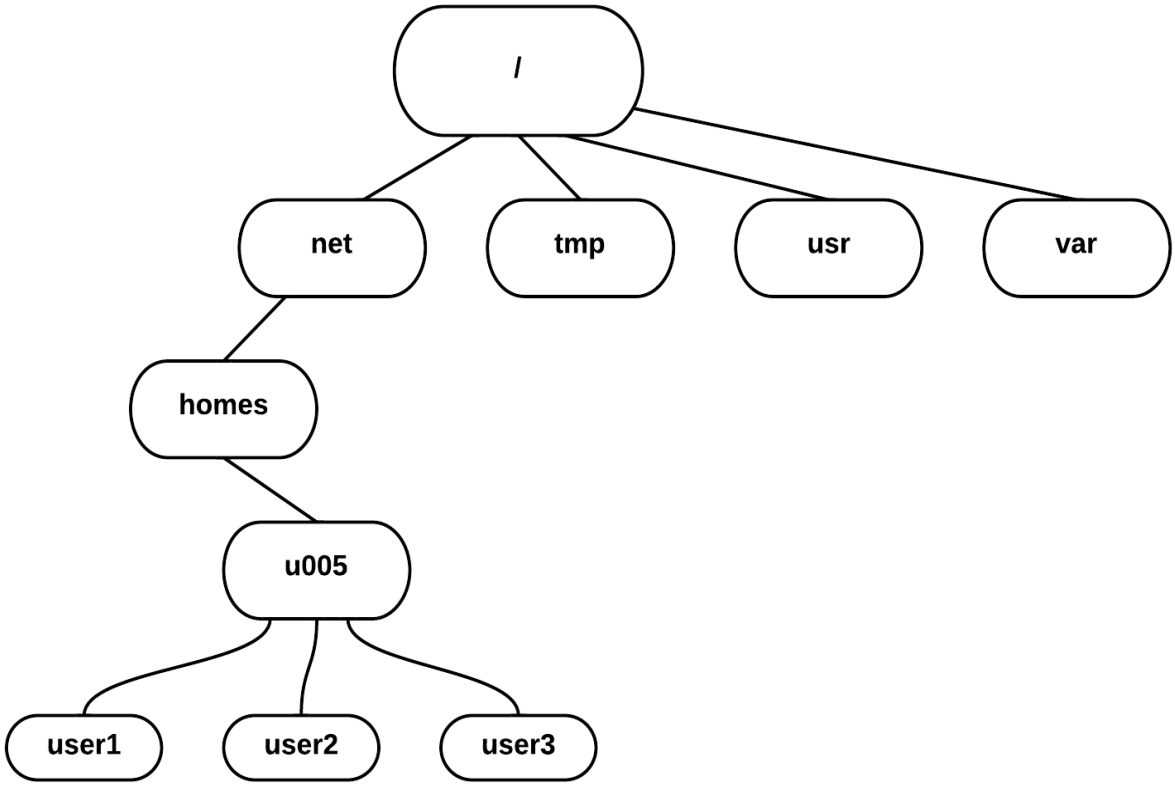
Kaikki tiedostot on ryhmitelty yhteen hakemistojärjestelmään. Tiedostosysteemi on järjestelty arvojärjestykseen kuin väärinpäin olevaan puuhun. Puun huippua on perinteisesti kutsuttu nimellä *juuri* (*root)* ja se usein kirjoitetaan kauttaviivana */*.



Kuva 2: Hakemistojärjestelmä Linux/Unix-järjestelmässä.

Kaaviosta (Kuva 2) näemme, että käyttäjien kotihakemistot sijaitsevat hakemistossa */home/* Käyttäjän tallentaessa kotihakemistoonsa tiedoston, esimerkiksi jos käyttäjä *user1* tallentaa Word-tiedoston *report.doc*, täydellinen polku kyseiseen tiedostoon on */home/user1/report.doc*

LUT:n Linux-luokan koneiden tiedostojärjestelmän rakenne poikkeaa edellä esitetystä hieman. Tämä johtuu siitä, ettei yksittäisen käyttäjän kotihakemisto sijaitse koneilla paikallisesti vaan verkossa. Verkossa sijaitsevat käyttäjien omat levytilat on liitetty tiedostojärjestelmään hakemistoon */net/homes* siten, että *homes*-hakemistossa on vielä alihakemistoja ryhmittäin. (Nämä ryhmät ovat olemassa vain järjestelmänhallinnallisista syistä, ja käyttäjien jaottelu ryhmiin on näennäisen satunnaista.)



Kuva 3: Hakemistojärjestelmä Linux/Unix-järjestelmässä, kun kotihakemistot sijaitsevat verkossa.

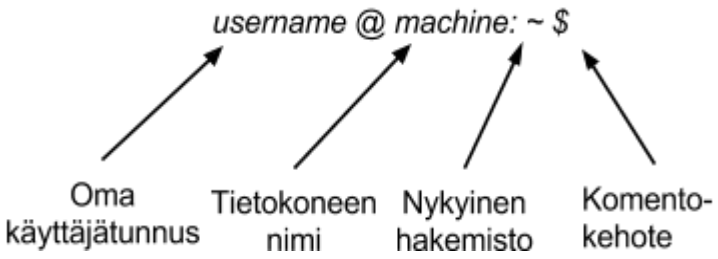
Komentotulkin avaaminen

Avataksesi terminaali-ikkunan, klikkaa “Terminal” (tai esim Pääte, Konsole, Terminaali, riippuen versiosta) -kuvaketta sovellusvalikosta.

Esimerkiksi Ubuntussa Unity-työpöydällä avataan sovellusvalikko, ja haetaan hakukenttään kirjoittamalla “terminaali” tai “terminal”. KDE-työpöytäympäristössä Konsole-terminaaliemulaattori on K-valikossa valitsemalla *Ohjelmat/Applications* ja *Apuohjelmat*/*Accessories*. Xfce:n vastaava Terminal emulator löytyy *Accesories/Apuohjelmat* valikosta.

OS X:ssä terminaali löytyy sovelluskansiosta *ohjelmat -> lisäohjelmat*.

Tämän jälkeen näytölle avautuu terminaali-emulaattori-ikkunassa komentotulkki odottamaan komentoja. Komentokehote näyttää seuraavanlaiselta:



Kuva 4: Komentokehotteen eri osat.

Tässä oppaassa komentotulkissa ajettavat esimerkkikomennot esitetään kursiivifontilla. Komentokehote esitetään typistetyssä muodossa, sisältäen vain $-merkin kehotteen tunnuksena ja jättämällä käyttäjänimen, koneennimen sekä nykyisen hakemiston pois. Esimerkiksi

$ echo “hello world”
hello world

Esimerkkiajon rivi, joka ei ala $-merkillä on ajettavan komennon (ohjelman) tuloste.

Luku 2: Tiedostonselaus

2.1 Tiedostojen ja hakemistojen listaus

ls (list)

Kun kirjaudut ensimmäistä kertaa sisään, senhetkinen hakemisto on kotihakemistosi. Se on nimetty käyttäjänimesi mukaan, ja henkilökohtaiset tiedostot ja alikansiot tallennetaan sinne.

Selvittääksesi mitä kotihakemistosi sisältää, kirjoita

$ ls

ls -komento (pieni L ja S) listaa nykyisen hakemistosi sisällön. Esimerkkiajo:

```
$ ls
Desktop     Dropbox     unixnoob   Public     shakespere.txt   Videos
Documents   Downloads   Music      Pictures   Templates
$
```

Hakemistossa ei välttämättä ole yhtään tiedostoa tai hakemistoa. Vaihtoehtoisesti siellä voi olla joitakin järjestelmänhaltijan asettamia tiedostoja.

ls kuitenkin listaa vain ne tiedostot, joiden nimet eivät ala pisteellä (**.**). Tällaiset tiedostot tunnetaan piilotettuina tiedostoina, ja ne usein sisältävät tärkeää tietoa ohjelmien asetuksiin liittyen. Näitä tiedostoja ei tule muuttaa syyttä, sillä joidenkin ohjelmien toiminta voi mennä sekaisin.

Nähdäksesi kaikki tiedostot hakemistossa (sisältäen myös piilotetut tiedostot), kirjoita

$ ls -la

Tällöin myös piilotetut tiedostot voidaan nähdä:

```
$ ls -la
.     .bash_history   Desktop     Dropbox     unixnoob   Public     shakespere.txt
Videos
..    .cache          Documents   Downloads   Music      Pictures   Templates
$
```

Esimerkkiajossa hakemiston tiedostolistaukseen ilmestyi nyt myös kaksi piilotiedostoa, *.bash_history* eli komentotulkin käskyhistorian sisältävä tiedosto sekä *.cache*, joka on itse asiassa väliaikaistiedostoja sisältävä hakemisto. Lisäksi komennon aikaansaamassa listauksessa on myös kansiot . ja .. joihin palaamme vielä tämän kappaleen aikana.

ls on esimerkki komennosta joka ymmärtää vaihtoehtoja: **-a** on tällaisesta esimerkki. Tällaisia komennoille annettavia parametrejä kutsutaan tuttavallisemmin myös vivuiksi tai

lipuiksi. Parametrit muuttavat komennon käyttäytymismallia. Käyttöohjeet, jotka kertovat mitä vaihtoehtoja mihinkin komentoon voi käyttää, ja kuinka jokainen niistä muokkaa komennon käyttäytymistä löytyvät komennon manuaalisivulta (lisää manuaaleista kappaleessa 4.3). Useissa ohjelmissa parametri **-h**, **-help** tai **--help** antaa lyhyen käyttöohjeen komennon käyttöön sekä listauksen mahdollisista parametreistä.

2.2 Hakemistojen luominen

mkdir (make directory)

Seuraavaksi luodaan kotihakemistoon alihakemisto, johon voidaan tallentaa oppaan esimerkkien läpikäynnin aikana luodut ja käytetyt tiedostot. Luodaksesi alihakemiston nimeltä *unixnoob* nykyiseen hakemistoosi kirjoita

$ mkdir unixnoob

Kansion näkemiseksi listaa tiedostot kuten edellisessä luvussa opimme.

2.3 Toiseen hakemistoon siirtyminen

cd (change directory)

Komento **cd *hakemisto*** käskee komentoriviä muuttamaan nykyiseksi hakemistoksi hakemiston '*hakemisto'.* Tämänhetkistä työhakemistoa voidaan ajatella hakemistona, jonka sisällä olet, eli nykyisenä asemanasi tiedostojen puussa.

Vaihtaaksesi äsken tehtyyn hakemistoon, kirjoita

$ cd unixnoob

Kirjoita **ls** jotta näet kansion sisällön (jonka kuuluisi olla tyhjä).

2.4 Hakemistot . ja ..

Edelleen hakemistossa *unixnoob,* kirjoita

$ ls -a

Kuten voit nähdä hakemistossa (kuten kaikissa muissakin hakemistoissa, poislukien päähakemisto /), listassa on kaksi erityistä kansiota nimeltä (**.**) ja (**..***).*

Nykyinen hakemisto (.)

UNIXissa (.) tarkoittaa nykyistä hakemistoa, joten kirjoittamalla

$ cd .

käsketään komentotulkkia pysymään nykyisessä hakemistossa (joka on tässä vaiheessa *unixnoob).* Tämä ei välttämättä vaikuta kovin hyödylliseltä, mutta pisteen (.) käyttäminen nykyisen hakemiston nimenä säästää paljolta kirjoittamiselta, kuten myöhemmin oppaassa voimme huomata.

Ylähakemisto (..)

(..) tarkoittaa nykyisen hakemiston “vanhempaa”, joten kirjoittamalla

$ cd ..

siirrytään yksi hakemisto ylemmäs järjestyksessä eli lähemmäs kotihakemistoa. Kokeile!

Huom: Pelkkä *cd* vie aina kotihakemistoon.

2.5 Polunnimet

pwd (print working directory)

Polunnimet mahdollistavat nykyisen olinpaikkasi selvittämisen suhteessa koko tiedostojärjestelmään. Esimerkiksi kotihakemiston täydellisen polun nimen selvittääksesi kirjoita **cd** jolloin pääset kotikansioon ja sen jälkeen

$pwd

jolloin komentotulkki tulostaa kotihakemistosi täydellisen polun tiedostojärjestelmässä.

2.6 Lisää kotihakemistoista ja polunnimistä

Polunnimien ymmärtäminen

Mene **cd**:llä kotihakemistoon, ja kotihakemistossa kirjoita

$ ls unixnoob

listataksesi *unixnoob* -hakemiston sisällön.

Luodaan tämän jälkeen *unixnoob*-hakemiston alle uusi hakemisto

$ mkdir unixnoob/varmuuskopiot

Sen jälkeen kirjoita

$ ls varmuuskopiot

Saat tämän kaltaisen viestin:

varmuuskopiot: No such file or directory

Tämä johtuu siitä, ettei hakemisto *varmuuskopiot* sijaitse nykyisessä työhakemistossa. Käyttääksesi komentoa tiedostoon (tai hakemistoon) joka ei ole nykyisessä työhakemistossa (eli hakemistossa jossa juuri olet), sinun täytyy joko mennä **cd**:n avulla oikeaan hakemistoon tai tarkentaa sen koko polunnimi, samaan tapaan kuin aiemmin loimme *varmuuskopiot*-hakemiston. Nähdäksesi *varmuuskopiot* -kansion sisällön, oikea komento olisi

$ ls unixnoob/varmuuskopiot

~ (kotihakemisto)

Kotihakemistoon voidaan myös viitata **~** (tilde) -merkillä. Sitä voidaan käyttää tarkentamaan polkuja jotka alkavat kotihakemistostasi, joten

$ ls ~/unixnoob

listaa aina *unixnoob* -kansion sisällön riippumatta siitä, missä kohtaa tiedostojärjestelmää tällä hetkellä on.

Mitä seuraavat komennot listaavat?

$ ls ~
tai
$ ls ~/..

2.7 Yhteenveto

Taulukko 1: Yhteenveto Linuxin yleisistä komentorivikomennoista.

Komento	**Tarkoitus**
ls	listaa tiedostot ja hakemistot
ls -a	listaa myös piilotetut tiedostot ja hakemistot
mkdir	tee hakemisto
cd *hakemisto*	siirry hakemistoon *hakemisto*
cd	siirry kotihakemistoon
cd ~	siirry kotihakemistoon
cd ..	siirry yksi hakemisto ylöspäin
pwd	näytä polku nykyiseen hakemistoon

Luku 3: Peruskomentoja

3.1 Tiedostojen kopiointi

cp (copy)

cp tiedosto1 tiedosto2 on komento, joka tekee kopion *tiedosto1*:stä nykyisessä työhakemistossa ja kutsuu kopiota *tiedosto2*:ksi

Esimerkiksi, luodaan jollain tekstieditorilla (graafinen tai komentorivipohjainen, gedit, nano, LibreOffice Writer…) tiedosto kotihakemistoon, nimeltään vaikkapa *testitiedosto.txt*. Seuraavaksi kopioidaan se hakemistoon *unixnoob* komennolla *cp*.

$ cp testitiedosto.txt unixnoob/.

Piste hakemistopolun perässä tarkoittaa, että tiedosto kopioidaan saman nimisenä. Piste ei tosin ole pakollinen merkki juuri hakemistopolun perässä - ainoastaan kun haluamme kopioida toista tiedostoa nykyiseen hakemistoon ja cp-komento vaatii kaksi parametria. Vaihtoehtoisesti voisimme tehdä kopiosta erinimisen komentamalla

$ cp testitiedosto.txt unixnoob/tiedostotesti.txt

3.2 Tiedostojen siirtäminen

mv (move)

mv tiedosto1 tiedosto2 siirtää (tai nimeää uudelleen) *tiedosto1*:stä *tiedosto2*:een. Siirtääksesi tiedostoa paikasta toiseen, käytä **mv**-komentoa. Se ei kopioi kuten **cp** vaan siirtää tiedoston uuteen paikkaan joten tiedostoja ei jää ylimääräisiä. Sitä voidaan käyttää myös uudelleennimeämiseen siirtämällä tiedosto samaan kansioon mutta antamalla sille uusi nimi. Seuraavaksi siirretään tiedosto *tiedostotesti.txt varmuuskopiot*-hakemistoon.
Aloita *unixnoob*-hakemistoon siirtymällä. Siellä kirjoita

$ mv tiedostotesti.txt varmuuskopiot/.

Kirjoita **ls** ja **ls varmuuskopiot** selvittääksesi, toimiko komento.

3.3 Tiedostojen ja hakemistojen poistaminen

rm (remove), rmdir (remove directory)

Tiedoston poistamiseen käytetään **rm** -komentoa. Esimerkkinä luodaan kopio tiedostosta *testitiedosto.txt* ja poistetaan se.
Unixnoob -hakemistossa kirjoita

$ cp testitiedosto.txt tempfile.txt
$ ls
$ rm tempfile.txt
$ ls

Hakemiston poistamiseen voi käyttää **rmdir** -komentoa (tarkista ensin että hakemisto on tyhjä). Yritä poistaa *varmuuskopiot* -hakemisto. UNIX/Linux ei anna tehdä sitä, sillä kansiossa on tavaraa.

3.4 Tiedoston sisällön näyttäminen ruudulla

clear (clear screen)

Ennen kuin aloitat seuraavan osion, saatat haluta tyhjentää terminaalin ikkunan edellisistä komennoista, jotta seuraavien komentojen tulosteet voidaan ymmärtää selkeästi.
$ clear

clear-komennolle on useimmissa komentokehotteissa myös näppäinoikotie CTRL+L.

cat (concatenate)

Komentoa **cat** voidaan käyttää tiedoston sisällön esittämiseen ruuduilla. Haetaan tätä tarkoitusta varten esimerkiksi Työaseman käytön perusteet -kurssin kotisivuilta, Noppa-opintoportaalista (http://noppa.lut.fi) pidemmänpuoleinen tekstitiedosto tutkailtavaksi (toki mikä tahansa muukin tekstitiedosto käy). Tentti- ja harjoitusmateriaalipaketti sisältää tiedoston *words-en.txt*, joka sisältää kokoelman englanninkielisiä sanoja aakkosjärjestyksessä. Tallenna tämä tiedosto kotihakemistoosi, ja kirjoita:

$ cat words-en.txt

Kuten voit nähdä, tiedosto on pidempi kuin ikkunan koko, jolloin se vierii ohi tehden tekstistä vaikeasti luettavan.

less

Komento **less** kirjoittaa tiedoston sisällön ruudulle yksi sivu kerrallaan. Kirjoita

$ less words-en.txt

Paina [**space**] jos haluat nähdä seuraavan sivun ja [**q**] jos tahdot lopettaa lukemisen. Kuten huomaamme, **less**iä kannattaa käyttää **cat**in sijasta pitkillä tiedostoilla.

head

head-komento tulostaa tiedoston ensimmäiset 10 riviä ruudulle. Tyhjennä ruutu ja kirjoita sitten

$ head words-en.txt

Ja sen jälkeen

$ head -5 words-en.txt

Mitä lisäys -5 teki komennolle?

tail

tail -komento tulostaa viimeiset 10 riviä tiedostosta ruudulle. Tyhjennä ruutu ja kirjoita

$ tail words-en.txt

Miten saat näkyviin viimeiset 15 riviä tiedostosta?

3.5 Tiedoston sisällön etsiminen

Yksinkertainen etsintä käyttäen less -komentoa

Käyttämällä **less**iä, voit etsiä avainsanaa tekstitiedostosta. Esimerkiksi etsiäksesi *words-en.txt* -tiedostosta sanaa **'**trammel**'**,kirjoita

$ less words-en.txt

kun olet tiedostossa, kirjoita kauttaviiva [**/**] ja sen jälkeen etsittävä sana */trammel*

Less löytää ja korostaa avainsanan. Kirjoita [**n**] etsiäksesi sanaa muualta samasta tiedostosta.

grep (search **g**lobally for a **r**egular **e**xpression and **p**rint)

grep on yksi UNIXin monista perusapuvälineistä. Se etsii tiedostoista tiettyjä sanoja tai merkkijonoja. **Grep**in kokeilemiseksi, hae tentti- ja harjoitusmateriaalipaketista tiedosto *science.txt* ja tallenna se kotihakemistoosi.

Tyhjennä ruutu ja näppäile sitten

$ grep science science.txt

Kuten näet, **grep** on tulostanut jokaisen rivin jolla on sana *science.*
Vai onko?
Kokeile

$ grep Science science.txt

Grep on merkkikoosta riippuvainen, eli sille science ja Science ovat eri sanat.
Jos et halua sen huomioivan merkkien kokoja, lisää vaihtoehto **-i** eli kirjoita

$ grep -i science science.txt

Etsiäksesi lausetta tai merkkijonoa, se täytyy sulkea yksinkertaisiin lainausmerkkeihin (‘ ‘). Esimerkiksi etsiäksesi ilmaisua *spinning top*, kirjoita

$ grep -i ‘spinning top’ science.txt

Muita vaihtoehtoja **grep**issä:

- **-v** näyttää rivit, jotka eivät sovi hakuehtoihin
- **-n** ilmoittaa myös jokaisen osuneen rivin numeron
- **-c** ilmoittaa ainoastaan osumien määrän

Kokeile optioita niin yhdessä kuin erikseenkin. Esimerkiksi niiden rivien, joilla ei ole sanoja science tai Science, laskeminen onnistuu komennolla

$ grep -ivc science science.txt

wc (word count)

Myös **wc** -komento on kätevä. Laskeaksesi sanat tiedostossa *science.txt,* kirjoita

$ wc -w science.txt

Jos haluat sanojen sijasta tietää rivien määrän, kirjoita

$wc -l science.txt

3.6 Yhteenveto

Taulukko 2: Lisää yleisiä komentoja.

Komento	**Tarkoitus**
cp tiedosto1 tiedosto2	kopioi *tiedosto1* ja nimeä se *tiedosto2*:ksi
mv tiedosto1 tiedosto2	siirrä tai nimeä uudelleen *tiedosto1->tiedosto2*
rm tiedosto	poista *tiedosto*
rmdir hakemisto	poista *hakemisto*
cat tiedosto	näytä *tiedosto*n sisältö
less tiedosto	näytä *tiedosto*n sisältö sivu kerrallaan
head tiedosto	näytä *tiedosto*n muutama ensimmäinen rivi
tail tiedosto	näytä *tiedosto*n muutama viimeinen rivi
grep ‘avainsana’ tiedosto	etsi avainsanaa *tiedosto*sta
wc tiedosto	laske *tiedosto*n rivit/sanat/merkit

Luku 4: Terminaalin I/O -käskyt ja uudelleenohjaus

4.1 Uudelleenohjaus

Useimmat UNIX -komentojen käynnistämät prosessit kirjoittavat standarditulosteen (eli kirjoittavat komentorivin ruudulle), ja monet ottavat syötteen standardisyötteestä (eli lukevat sen näppäimistöltä). On olemassa myös standardivirhe, jossa prosessit kirjoittavat oletuksena virheilmoituksensa ruudulle. Olemme aikaisemminkin nähneet yhden käytön **cat** -komennolle, eli kun se kirjoittaa tiedoston sisällön ruudulle. Nyt kirjoita **cat** tarkentamatta luettavaa tiedostoa

$ cat

Sitten kirjoita muutama sana näppäimistöllä ja paina [**enter**]. Lopulta pidä [**Ctrl**]- näppäintä pohjassa ja paina [**d**] (lyhennettynä **^D**) lopettaaksesi syötteen. Mitä tapahtui?

Ajamalla **cat**-komennon ilman luettavaa tiedostoa, se lukee standardisyötteen (näppäimistöltä) ja saavuttaessaan “tiedoston” (eli syötteen) lopun (**^D**), se kopioi syötteen tulosteeseen (eli ruudulle). UNIXissa voi uudelleenohjata sekä komentojen syötteitä että tulosteita.

4.2 Tulosteen uudelleenohjaus

> -symbolia käytetään komennon tulosteen uudelleenohjaamiseen. Esimerkiksi luodaksesi tiedoston nimeltä *lista1* joka sisältää listan hedelmiä, kirjoita

$ cat > lista1

ja kirjoita joidenkin hedelmien nimiä (paina [**enter**] joka hedelmän välissä)

päärynä
banaani
omena
^D

cat lukee syötteen näppäimistöltä ja **>** uudelleenohjaa tulosteen joka normaalisti tulisi näytölle, tiedostoon nimeltä *lista1.* Tarkista tiedoston sisältö kirjoittamalla

$ cat lista1

Tiedostoon lisääminen

Muoto **>>** lisää tulosteen aikaisempaan tiedostoon. Lisätäksesi lisää hedelmiä tiedostoon *lista1*, kirjoita

$ cat >> lista1

ja kirjoita lisää hedelmien nimiä:

persikka
rypäle
appelsiini
^D

Lue tiedoston sisältö.

Nyt sinulla pitäisi olla kaksi tiedostoa, joista toisessa kuusi ja toisessa neljä hedelmää. Seuraavaksi käytetään **cat**-komentoa yhdistämään *lista1* ja *lista2* uudeksi tiedostoksi nimeltä *biglist.* Kirjoita

$ cat lista1 lista2 > biglist

Tämä lukee *lista1* ja *-2*:n sisällöt siirtäen tulostetekstin tiedostoon *biglist.* Lue uuden aiemmin tiedoston sisältö opitulla tavalla.

4.3 Syötteen uudelleenohjaus

< -symbolia käytetään komennon syötteen uudelleenohjaamiseen. Komento **sort** järjestää listan aakkosellisesti tai numeerisesti. Kirjoita

$ sort

Ja näppäile muutamia eläinten nimiä (paina [**enter**] jokaisen välissä)

koira
kissa
lintu
apina
^D

Tuloste on
apina
kissa
koira
lintu

Käyttämällä **<** -merkkiä voit uudelleenohjata syötteen tulemaan myös tiedostosta näppäimistön sijaan. Esimerkiksi järjestääksesi listan hedelmistä, kirjoita

$ sort < biglist

ja järjestelty lista ilmestyy tulosteena ruudulle.

Tulostaaksesi järjestellyn listan tiedostoon, kirjoita

$ sort < biglist > jlista

Lue lopuksi tiedoston *jlista* sisältö **cat**-komennon avulla.

4.4 Putket

Putket mahdollistavat UNIX-komentojen yhdistetyn käytön suorittamalla ensin yhden komennon ja sen jälkeen välittämällä tuotetun tulosteen syötteeksi seuraavalle komennolle.

Tiedämme esimerkiksi entuudestaan, että komento *ls* listaa hakemiston sisällön **ja** komento *wc* laskee sanoja sekä rivejä sille annetusta tiedostosta. Putkittamalla *ls*:n tulosteen edelleen *wc*:lle voimme selvittää hakemistossa olevien tiedostojen ja alihakemistojen lukumäärän:

$ ls | wc -l

Komentoja voi toki putkittaa mielivaltaisen määrän kerrallaan, vain mielikuvitus on rajana. Esimerkkinä kolmen komennon putkesta:

$ echo "whole hole with no holes" | grep "hole" -o | wc -w

Kokeile jokaista komentoa yksitellen ja sen jälkeen yhdessä. Mitä koko komentoputki tekee?

4.5 Yhteenveto

Taulukko 3: Uudelleenohjaus.

Komento	**Tarkoitus**
komento > tiedosto	uudelleenohjaa standardituloste tiedostoon
komento >> tiedosto	lisää standarditulostetta tiedostoon
komento < tiedosto	uudelleenohjaa standardisyöte tiedostosta
komento1 | komento2	putkita komento1:n tuloste komento2:n syötteeksi
cat tiedosto1 tiedosto2 > tiedosto0	liitä tiedostot 1 ja 2 tiedostoon 0
sort	järjestä dataa

Luku 5: Tiedostojen nimet ja jokerimerkit, oppaat ja manuaalit

5.1 Jokerimerkit

Jokerimerkki *

Merkkiä '*' kutsutaan jokerimerkiksi, ja se vastaa mitä tahansa merkkiä tai merkkijonoa tiedoston tai kansion nimessä. Esimerkiksi, *unixnoob* hakemistossa kirjoita

*$ ls list**

Tämä listaa nykyisen hakemiston kaikki tiedostot joiden nimi alkaa *list…*
Kokeile myös

*$ ls *list*

Näin saadaan näkyviin kaikki tiedostot nykyisessä hakemistossa jotka loppuvat *...list*

Jokerimerkki ?

? -merkki vastaa täsmälleen yhtä merkkiä. Näin ollen **?ouse** vastaa sanoja kuten *house* ja *mouse,* muttei sanaa *grouse*. Kokeile

$ ls ?list

5.2 Sopivat nimet tiedostoille

Tässä on huomautettava, että hakemistot ovat vain erikoistapauksia tiedostoista, joten säännöt ja hyvät käytännöt pätevät myös niihin. Tiedostojen nimeämisessä erikoismerkityksellisiä merkkejä kuten **/ * & %** , sekä välilyöntejä tulisi välttää. Turvallisinta on käyttää ainoastaan aakkosnumeerisia merkkejä eli kirjaimia ja numeroita yhdessä alaviivan **_** ja pisteen **.** kanssa.

Taulukko 4: Tiedoston nimiä.

Hyviä tiedoston nimiä	**Huonoja tiedoston nimiä**
projekti.txt	projekti
miun_ohjelma.c	miun ohjelma.c
martti_pertti.odt	martti & pertti.odt

Sovinnonmukaisesti tiedostojen nimet alkavat pienellä kirjaimella ja voivat loppua pisteeseen sekä sitä seuraavaan tiedostopäätteeseen, joka viittaa tiedoston sisältöön. Esimerkiksi kaikki

C-koodia sisältävät tiedostot voidaan nimetä lopulla **.c** (esimerkiksi ohj1.c). Haluttaessa listata kaikki tiedostot joissa on C-koodia kotihakemistossasi, täytyy kirjoittaa ainoastaan **ls *.c** kyseisessä hakemistossa.

5.3 man

Manuaalisivut

UNIXissa sisäänrakennetut online -käsikirjat antavat tietoa useimmista komennoista (eli ohjelmista). Manuaalisivut kertovat, mitkä vaihtoehdot tietylle komennolle sopii, ja kuinka joka vaihtoehto muokkaa komennon käyttäytymistä. Tämän lisäksi manuaalit kattavat myös ohjelmoijalle hyödylliset kirjasto- ja järjestelmäkutsut sekä konfigurointitiedostot ja standardit.

Kirjoita **man *komento*** lukeaksesi manuaalisivun tietystä komennosta. Esimerkiksi saadaksesi lisää tietoa **wc** (word count) -komennosta, kirjoita

$ man wc

Vaihtoehtoisesti voidaan käyttää komentoa

$ whatis wc

joka tarjoaa yksirivisen, manuaalisivua suppeamman kuvauksen komennosta.

Komentojen manuaalisivut on jaoteltu erillisiin manuaaliosaan niiden käyttötarkoituksen mukaisesti. Yleiskomennot, kuten *wc*, ovat osassa 1. *man*-komennon yhteydessä voikin täsmentää, minkä manuaaliosan artikkeli halutaan lukea:

$ man 1 wc

toimii samoin kuin aiempi *man*-komento. Sen sijaan

$ man 2 wc

ei toimi, sillä manuaalien osassa kaksi ei ole sivua komennolle *wc*.

Manuaalien jakaminen eri osiin johtuu edellä mainitusta manuaalisivujen kattavuudesta: Komento voi viitata ohjelmiin, järjestelmäkutsuihin tai standardeihin, jolloin samanniminen manuaalisivu voi esiintyä kahdesti. Esimerkiksi *printf* on sekä komentoriviohjelma, jolla saadaan tulostettua tekstiä ja numeroita ruudulle, että järjestelmäkutsu, joka myös tulostaa ruudulle. Eroavaisuutena näissä on se, että *printf*-ohjelmaa käyttää loppukäyttäjä komentotulkissa, kun *printf*-aliohjelmakutsua käyttää C-ohjelmoija tehdessään ohjelmaa loppukäyttäjille.

Manuaalisivujen osastoja on uusissa GNU/Linux-järjestelmissä yhdeksän. Mikäli käyttäjän antamalla komennolla on manuaalisivuja useammissa manuaaliosissa, näytetään yleiskomennot ennen ohjelmoijalle tarkoitettuja manuaaleja. Tällöin

$ man printf

näyttää manuaalisivun osasta 1, kun ohjelmoijan dokumentaatio on manuaaliosassa 3:

$ man 3 printf

Apropos

Kun komennon nimestä ei ole varma

$ apropos avainsana

antaa komennot, joiden manuaalisivuilla avainsana on (huom. englanniksi) Kokeile esimerkiksi:

$ apropos copy

Lisäksi komento *man -k* toimii identtisesti *apropos*-komennon kanssa.

5.4 Yhteenveto

Taulukko 5: Jokerimerkkejä ja lisää komentoja.

Komento	**Tarkoitus**
*****	vastaa kuinka monta tahansa merkkiä
?	vastaa yhtä merkkiä
man komento	lue komennon online-manuaali
whatis komento	lyhyt kuvaus komennosta
apropos avainsana	etsi komentoja avainsanalla man-sivuilta

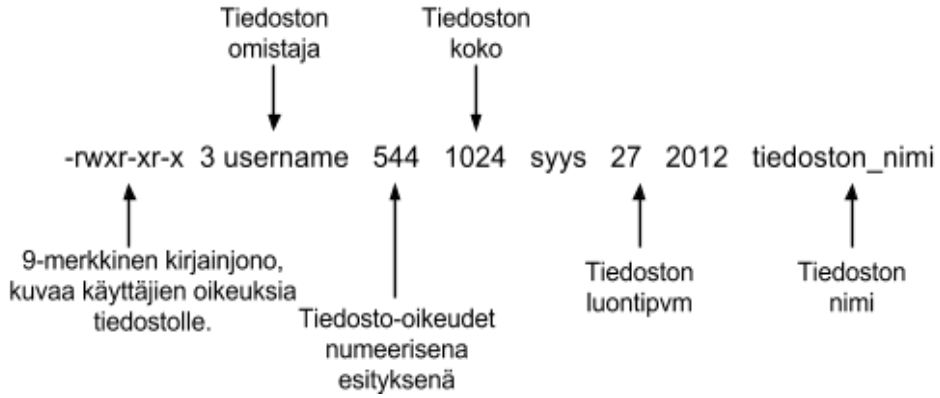
Luku 6: Käyttöoikeudet ja prosessit

6.1 Tiedostojärjestelmän turvallisuus (oikeus päästä käsiksi)

Unixnoob -hakemistossa kirjoita

$ ls -l (l =long listing)

Näet, että nyt hakemiston sisällöstä saa paljon erilaisia tietoja, joita seuraavassa esimerkissä valaistaan:



Kuva 5: tiedostojen metatietojen merkitys.

Jokaisella hakemistolla ja tiedostolla on jonkinlaiset käsittelyoikeudet, jotka nähdään kirjoittamalla **ls -l**. Lisäksi **ls -lg** -komennolla voidaan saada lisätietoa siitä, kuka käyttäjistä tai mikä käyttäjäryhmistä omistaa tiedoston.

Vasemmanpuoleisessa sarakkeessa on 10 symbolin (**d, r, w, x** tai **-**) pituinen merkkijono. Jos ensimmäinen merkki on **d**, kyseessä on hakemisto (directory). Muussa tapauksessa jono alkaa **-** -merkillä.

Ensimmäinen merkki on tavallisella tiedostolla **-**. Mikäli kyseinen tiedosto on hakemisto, näytetään merkki **d** tai **l** mikäli tiedosto on symbolinen linkki, joka johtaa jonnekin muualle.

9 jäljellä olevaa symbolia osoittavat lupia (tai käsiksipääsyoikeuksia) ja niitä käsitellään 3 merkin ryhmissä.

- Vasemmanpuoleinen kolmen ryhmä antaa luvat tiedoston omistavalle käyttäjälle (eli käyttäjälle *username* yläpuolella olevassa esimerkissä
- Keskimmäinen taas antaa luvat tiedoston omistavalle ryhmälle
- Oikeanpuolimmainen ryhmä antaa luvat kaikille muille

Symboleilla on hieman eri merkitykset riippuen siitä, viittaavatko ne tiedostoon vai hakemistoon.

Käsittelyoikeudet tiedostoissa

- **r (**tai **-**) ilmaisee luvan lukea tiedostoa (read -lupa) tai sen puutteen, eli saako tiedostoa lukea ja kopioida
- **w** (tai **-**) ilmaisee luvan kirjoittaa tiedostoon (write -lupa) eli saako tiedostoa muuttaa vai ei
- **x** (tai **-**) ilmaisee suorituslupaa (execute) tai sen puuttetta, eli saako tiedoston suorittaa

Lisäksi käsittelyoikeuksissa voi tulla vastaan myös erikoisemmat merkit **s**, **S** ja **t**, jotka liittyvät tiedoston omistamiseen sekä suorittamiseen toisena käyttäjänä tiedoston omistajan käyttöoikeuksilla.

Käsittelyoikeudet hakemistoissa

- **r** antaa käyttäjän listata tiedostot hakemistossa
- **w** tarkoittaa, että käyttäjät saavat poistaa tiedostoja hakemistosta tai siirtää tiedostoja sinne
- **x** tarkoittaa oikeutta päästä käsiksi hakemiston tiedostoihin. Tämä edellyttää sitä että käyttäjällä on myös itse tiedostoon tarvittavat luvat

Näin ollen lukeaksesi tiedostoa, sinulla täytyy olla suorituslupa hakemistoon, jossa tiedosto on, ja näin ollen jokaiseen hakemistoon sitä aikaisemmin eli puuta ylöspäin.

Esimerkkejä

Taulukko 6: Esimerkkejä tiedosto-oikeuksista.

-rwxrwxrwx	tiedosto, jonka kaikki voivat lukea, kirjoittaa tai suorittaa (sekä poistaa)
-rw-------	tiedosto jonka/johon vain omistaja voi lukea, kirjoittaa tai suorittaa (esimerkiksi tiedosto postilaatikossasi)

6.2 Käsittelyoikeuksien muuttaminen

chmod (changing a file mode)

Vain tiedoston omistaja voi käyttää **chmod** -komentoa muuttaakseen sen lupia. Komennon vaihtoehdot ovat tässä:

Taulukko 7: Tiedosto-oikeuksien muokkausmääreet.

Symboli	**Tarkoitus**
u	käyttäjä (user)
g	ryhmä (group)
o	muut (other)
a	kaikki (all)
r	luku (read)
w	kirjoitus ja poisto (write)
x	suoritus (execute)
+	lisää lupa
-	poista lupa

Esimerkiksi luku -ja suorituslupien poistaminen tiedostosta *biglist* ryhmältä ja muulta maailmalta onnistuu näin:

$ chmod go-rwx biglist

Tämä ei vaikuta muihin lupiin.
Antaaksesi luku- ja kirjoitusluvat kaikille tiedostossa *biglist,* kirjoita

$ chmod a+rw biglist

6.3 Prosessit ja työt

Prosessi on suoritettava ohjelma, joka on tunnistettavissa ainutlaatuisen PID-koodin (“prosessi-id”) avulla. Prosesseista saa tietoa näkyville kirjoittamalla

$ ps

Prosessi voi olla etualalla, taustalla tai pysäytettynä. Yleisesti komentorivi ei palauta kehotetta kunnes senhetkinen prosessi on lopettanut suorittamisen. Tiettyjen prosessien käsittelyt vievät paljon aikaa ja varaavat komentoriviä. Tällaisen prosessin voi laittaa ajamaan taustalle, jolloin UNIXin kehote palautuu välittömästi, ja muita tehtäviä voidaan panna täytäntöön samalla kun edellinen on kesken.

Taustaprosessien ajaminen

Prosessin laittamiseksi taustalle, kirjoita **&** komennon loppuun. Esimerkiksi komento **sleep** odottaa annetun sekuntimäärän ennen jatkamista. Kirjoita

$ sleep 10

Tämä odottaa 10 sekuntia ennen kehotteen *$* palauttamista. Ennen kuin kehote on palautettu, ainoa vaihtoehto on odottaa.
Ajaaksesi **sleep**-komennon taustalla, kirjoita

$ sleep 10 &
[1] 8164

& ajaa ohjelman taustalla ja palauttaa kehotteen heti, jolloin käyttäjää saa ajaa muita ohjelmia odottaessa toisen valmistumista. Ensimmäisen rivin yläpuolisessa esimerkissä käyttäjä kirjoittaa itse; seuraavan rivin, joka ilmaisee työn numeroa ja PID:tä, palauttaa kone. Käyttäjälle ilmoitetaan työn numero (numeroitu 1:stä eteenpäin) joka on suljettu hakasulkuihin, yhdessä PID -numeron kanssa. Komentorivi ilmoittaa myös, kun prosessi on valmistunut. Prosessien ajo taustalla on hyödyllistä etenkin töissä, joiden valmistuminen vie aikaa.

Etualan prosessin siirto taustalle

$ sleep 1000

Voit jäädyttää etualalla ajavan prosessin painamalla **^Z** eli painamalla [**Ctrl**] -näppäintä pohjassa ja samalla [**z**]. Sen jälkeen prosessin voi siirtää taustalle kirjoittamalla

$ bg

Huom: Älä laita taustalle ohjelmia, jotka vaativat vuorovaikutusta (esim tekstieditorit) käyttäjän kanssa!

6.4 Prosessin lopettaminen

kill (lopeta tai anna merkki prosessille)

Joskus on tarpeen pakottaa prosessi kiinni (esimerkiksi kun ohjelman suoritus kestää sietämättömän kauan tai on kokonaan pysähtynyt). Etualalla ajavan prosessin voi tappaa kirjoittamalla **^C** (**Ctrl+C**). Aja esimerkiksi

$ sleep 100
^C

Hiljennetyn tai taustaprosessin voi lopettaa kirjoittamalla

$ kill %työn numero

Esimerkiksi

$ sleep 100 &
$ jobs

Jos työn numero on 4, kirjoita

$ kill %4

Tarkista toimiiko komento tutkimalla töiden listaa (**jobs**)

ps (process status)

Vaihtoehtoisesti prosessit voidaan tappaa löytämällä prosessien numerot (PID:t) ja käyttämällä **kill *PID_number***

$ sleep 1000 &
$ ps
PID TT S TIME COMMAND
20077 pts/5 S 0:05 sleep 1000
21563 pts/5 T 0:00 netscape
21873 pts/5 S 0:25 nedit

Prosessin **sleep 1000** tappaminen onnistuu komennolle

$ kill 20077

Sen jälkeen tarkista **ps**-komennolla, onko prosessia enää jäljellä. Jos prosessi taistelee vastaan, käytä optiota **-9** eli kirjoita

$ kill -9 20077

Huom: On mahdotonta lopettaa toisen käyttäjän prosesseja! ps näyttää ainoastaan käyttäjän samasta komentotulkista käynnistämät prosessit. Nähdäksesi kaikki omat prosessisi kokeile

$ ps -ef | grep oma_kayttajatunnus.

6.5 Yhteenveto

Taulukko 8: Tiedosto-oikeus- ja prosessikomennot.

Komento	**Tarkoitus**
ls -lag	näytä käsittelyoikeudet kaikkiin tiedostoihin
chmod [optiot] tiedosto	muuta käsittelyoikeuksia nimetylle tiedostolle
komento &	aja komento taustalla
^C	lopeta etualalla ajava prosessi
^Z	pysäytä prosessi tilapäisesti
bg	siirrä hiljennetty työ taustalle
jobs	näytä senhetkiset työt
fg %1	siirrä työ numero 1 etualalle
kill %1	tapa työ numero 1
ps	listaa senhetkiset prosessit
kill 26152	tapa prosessi numero 26152

Luku 7: Muita hyödyllisiä UNIX-komentoja

quota

Quota kertoo tiedostojärjestelmään liitetyt levyt (sekä paikalliset että verkkolevyt), levytilan määrän ja käytetyn tilan määrän.

Käytetyn osuuden levytilasta voi tarkistaa käskyllä

$ quota -v

df

Df puolestaan ilmoittaa jäljellä olevan tilan tiedostojärjestelmässä. Esimerkiksi selvittääksesi kuinka paljon tilaa on jäljellä tiedostojärjestelmään liitetyillä levyillä, kirjoita

$ df .

Tällöin näytetään levyllä oleva vapaa tila kilotavuissa. Helpompilukuisen listauksen, jossa tila näytetään mega- ja gigatavuina saa vivulla *-h* (human readable):

$ df -h

du

Komento **du** tulostaa kilobitteinä jokaisen alikansion viemän tilan. Tämä on hyödyllistä jos muistin kiintiö on ylittynyt ja halutaan selvittää, missä hakemistossa on eniten tiedostoja. Kirjoita kotihakemistossa

*$ du -s **

-s -kytkin näyttää ainoastaan summauksen (total size) ja ***** tarkoittaa sekä tiedostoja, että hakemistoja. Myös *du*-komennosta saa human readable -listauksen *-h* -vivulla.

gzip

Gzip on pakkausohjelma, joka pienentää tiedoston kokoa ja näin vapauttaa arvokasta levytilaa. Kirjoita esimerkiksi

$ ls -l science.txt

ja huomaa tiedoston koko minkä **ls -l** näyttää. Pakkaa tiedosto kirjoittamalla

$ gzip science.txt

Gzip puristaa tiedoston pieneen kokoon ja asettaa sen tiedostoon nimeltä **science.txt.gz**. Koon muutos voidaan tarkistaa kirjoittamalla **ls -l** uudestaan. Tiedosto puretaan käyttämällä **gunzip**-komentoa.

$ gunzip science.txt.gz

zcat

Zcat lukee gzipatut tiedostot purkamatta niitä ensin.

$ zcat science.txt.gz

Jos teksti rullaa ohi liian nopeasti, tulosteen voi putkittaa **less**-komennon kautta.

$ zcat science.txt.gz | *less*

file

File luokittelee nimetyt tiedoston niiden sisältämän datatyypin mukaan, esimerkiksi ascii (teksti), kuvat, pakattu data jne. Katsauksen kaikkiin kotikansion tiedostoihin saa kirjoittamalla

*$ file **

diff

Komento vertailee kahden tiedoston sisältöjä ja esittää niiden eroavaisuudet. Oletetaan, että kyseessä on tiedosto nimeltä *tiedosto1*, josta editoidaan joitakin osia ja uusi versio tallennetaan nimellä *tiedosto2.* Eroavaisuudet nähdään kirjoittamalla

$ diff tiedosto1 tiedosto2

Rivit, jotka alkavat **<** -merkillä, viittaavat *tiedosto1*:een, ja taas **>** -merkityt *tiedosto2*:een.

find

Find -komento etsii hakemistoista tiedostoja ja alihakemistoja annettujen hakuehtojen, kuten nimen, päivämäärän, koon tai minkä tahansa määritellyn ominaisuuden mukaan. Yksinkertaiseen komentoon on monia eri optioita - komennon manuaalin voi lukea kirjoittamalla **man find**.
Etsiäksesi kaikki tiedostot nykyisestä hakemistosta (**.**) ja sen alihakemistoista tarkenteella **.txt**, ja sen jälkeen tulostaaksesi tiedostojen nimet ruudulle, kirjoita

$ find . -name “.txt” -print*
Sen sijaan löytääksesi kaikki kooltaan yli 1Mb:n tiedostot ja näyttääksesi tulokset pitkänä listauksena, kirjoita

$ find . -size +1M -ls

history

C-shell ja bash pitävät järjestettyä listaa kaikista komennoista joita käyttäjä on syöttänyt. Jokaiselle on annettu numero sen järjestyksen mukaan, missä komennot on syötetty.

$ history

Huutomerkillä (**!**) voi suorittaa uudelleen komentohistoriaan tallentuneita käskyjä helposti.

$!! (palauta viimeisin komento)
$!-3 (palauta 3. viimeisin komento)
$!5 (palauta 5. komento listassa)
$!grep (palauta viimeisin komento, joka alkaa **grep**illä)

Muistissa olevan komentohistorian kokoa voi muuttaa komennolla *set history=n*, esimerkiksi

$ set history=100

screen

Screen on hyödyllinen ohjelma, jonka avulla käyttäjä voi asettaa ohjelmia jatkamaan suoritustaan taustalle häiritsemättä komentotulkin muuta toimintaa. Screenin hyödyllisyys on erityisesti vuorovaikutusta vaativien ohjelmien siirtämisessä taustalle. Samoin screen pitää ohjelman ajossa taustalla, vaikka käyttäjä kirjautuisi välillä työasemasta ulos!

Screen tukee myös useita sessioita, eli useita ohjelmia voi olla samalla työasemalla tausta-ajossa.

$ screen nano

Tämän jälkeen komentorivipohjainen tekstieditori **nano** käynnistyy, ja avattuun uuteen tiedostoon voi kirjoittaa teksitä. Nanon voi tämän jälkeen asettaa taustalle näppäilemällä **^a** ja sen jälkeen **d** (eli CTRL+a, d). Tällöin käyttää voi jatkaa komentorivin käyttöä ja myöhemmin palata äsken avattuun tekstitiedostoon. Listauksen avoinna olevista screeneistä saa komennolla

$ screen -ls

Mikäli vain yksi screen-sessio on käyninssä, sen voi palauttaa etualalle komennolla *screen -r*. Mikäli useampia screen-sessioita on käynnissä, *screen -ls* näyttää seuraavankaltaisen listauksen:

There are screens on:
4662.pts-10.LUT3631 (06.10.2014 16.04.30) (Detached)
28250.pts-4.LUT3631 (19.08.2014 18.15.26) (Attached)
2 Sockets in /var/run/screen/S-d0358412.

Tällöin on täsmennettävä, minkä session screenin halutaan tuovan takaisin etualalle antamalla komennon lisäksi session nimi:

$ screen -r 4662.pts-10.LUT3631

Huomaa, että tässäkin komennossa automaattista täydennystä voi käyttää apuna (eli painamalla tabulaattoria session nimeä kirjoittaessa).

Ajettaessa useita screenejä voi olla hyödyllistä antaa sessioille oma, selkeyttävämpi nimi:

$ screen -S tekstieditori nano

Lopuksi, screen-sessio sammuu automaattisesti kun ajettava ohjelma lopetetaan. Session voi kuitenkin lopettaa myös screenin avulla. Etualalla oleva sessio lopetetaan painamalla näppäinyhdistelmää **^a** ja kirjoittamalla sen jälkeen **:quit**

Taustalle laitetun prosessin voi tappaa komennolla

$ screen -X -S session_numero_tai_nimi quit

Screenin komennoille voidaan antaa parametrinä session nimi täydellisenä (esim. 4742.tekstieditori), mutta myös pelkkä nimi (“tekstieditori”) tai numero (4742) riittää.

Screenillä ajettava ohjelma pysyy siis taustalla ajossa vaikka käyttäjä kirjautuisi välillä ulos työasemasta. Samoin screenillä ajettavan ohjelman voi palauttaa mille tahansa työasemaan yhteydessä olevalle terminaalille: Voit esimerkiksi käynnistää ohjelman screenillä fyysisesti työasemalla ja myöhemmin palata sen käyttöön etäyhteyden avulla.

Joissain tapauksissa screen-sessio saattaa olla jollakin terminaalilla etualalla (=”attached”), jolloin sitä ei voida tuoda etualalle toisessa terminaalissa. Tällöin tulee *screen*-komentoon lisätä parametri *-d*, joka käskee tuomaan ohjelman etualalle, ja viemään muiden terminaalien etualalla olevat screen-sessiot taustalle. Screen sallii myös usean terminaalin käyttää ohjelmia samanaikaisesti. Komento

$ screen -r -x

tarkoittaa screen-session palauttamista etualalle nykyisessä terminaalissa samalla jättäen muut etualalla olevat screen-sessiot etualalle (ts. voit käyttää samaa ohjelmaa kahden terminaalin kautta samanaikaisesti.)

Luku 8: Verkko-ohjelmat ja etäkäyttö

wget

wget on apuohjelma tiedostojen lataamiseen webistä. Esimerkiksi komento

$ wget http://www.google.fi

lataa Google-hakupalvelun HTML-muotoisen etusivun nykyiseen hakemistoosi nimellä *index.html*. Tiedoston nimen voi määritellä itse vaihtoehdolla *-O*. wget on kätevä työkalu tiedostojen lataamiseen internetistä ja se tukee myös useiden tiedostojen (ja hakemistojen) latausta samalla kertaa.

ssh

Secure shell eli ssh itsessään on salatun tietoliikenteen protokolla. Ohjelmaa *ssh* puolestaan käytetään Unix/Linux -työaseman etäkäyttöön komentoriviltä. (Oletuksena on, että käyttäjällä on pääsy ja käyttäjätunnukset etäkäytettävään tietokoneeseen.)

Esimerkiksi yliopiston sisäverkossa voimme ottaa etäyhteyden Linux-koneesta toiseen[1] komentamalla

$ ssh akseli.pc.lut.fi

Tällöin ssh käyttää käyttäjätunnuksena etäkoneeseen samaa tunnusta, millä olemme kirjautuneena sisään paikalliseen koneeseen. Mikäli haluamme käyttää toista käyttäjätunnusta, komento annetaan muodossa *ssh tunnus@konenimi*.

Onnistuneen kirjautumisen jälkeen ssh palauttaa komentokehotteen, jota ajetaan etäkoneelta. Etäyhteyttä voidaan esimerkiksi tarvita halutessamme ajaa jotakin ohjelmistoa, joka ei ole asennettuna paikalliselle koneelle, mutta joka on saatavilla toisella Linux-työasemalla. ssh:n yli voidaan käyttää myös graafisen käyttöliittymän ohjelmia, jos *ssh*-komentoon liitetään vaihtoehto *-X*.

scp

Secure copy eli scp on työkalu, jonka avulla voidaan kopioida tiedostoja kahden tietokoneen välillä, mikäli käyttäjällä on pääsy (käyttäjätunnus) molempiin koneisiin. scp:n käyttö muistuttaa tavallisen kopiointikäskyn, *cp:*n käyttöä:

$ scp paikallinen_tiedosto tunnus@etäkone:etätiedoston_nimi

Etäkopiointikäsky annetaan samassa muodossa kuin paikallinen kopiointikäsky: *scp mistä mihin*.

[1] Ainoastaan sisäverkossa näkyvät Linux-luokan koneet ja niiden konenimet on (oppaan kirjoitushetkellä) listattu Tuotantotalouden tiedekunnan wiki-sivustolla http://www.it.lut.fi/

Esimerkissä siis kopiodaan tiedosto nimeltä *paikallinen_tiedosto* toiselle työasemalle käyttäjän *tunnus* kotihakemistoon nimellä *etätiedoston_nimi.* Mikäli siirrettävän tiedoston nimi halutaan pitää samana (kuten myös tavallisessa *cp*-käskyssä, voidaan kohdetiedoston nimi jättää tyhjäksi tai merkitä samannimisyyttä pisteellä).

Komennolla voi luonnollisesti myös kopioida tiedostoja etäkoneelta paikalliselle koneelle, eli päinvastoin kuin esimerkissä. Itse asiassa komennolla voi myös siirtää tiedostoja kahden etäkoneen välillä ilman, että siirrettävää tiedostoa tarvitsee ensin kopioida paikalliselle koneelle.

Luku 9: Ohjelmien asennus sekä lähdekoodista kääntäminen

9.1 Ohjelmien asentaminen UNIX- ja Linux-järjestelmissä yleisesti

Järjestelmäämme on asennettu monia vapaita sekä kaupallisia ohjelmistopaketteja, jotka ovat avoinna kaikille käyttäjille. Vaikka ohjelmistovalikoima onkin kattava, ei se kuitenkaan voi koskaan olla täydellinen. Tässä kappaleessa käsitellään ohjelmien asentamista yleisesti, pakettienhallintaohjelman käyttöä sekä ohjelmien kääntämistä (ja asennusta) lähdekoodista

Yliopiston Linux-työasemilla ohjelmien asennus ei onnistu, sillä tavallisella käyttäjällä ei ole pääkäyttäjän oikeuksia. Yksittäinen käyttäjä voi kuitenkin ladata ohjelmia verkosta käyttöä varten, joko suorittamalla valmiiksi käännetyn binääritiedoston tai kääntämällä ohjelmistopaketin itse suoraan lähdekoodista. Huomaa, että yliopiston Linux-luokassa (tai missä tahansa muulla, kuin omalla tietokoneella) työskennellessä ohjelmien asentaminen tai muu **pääkäyttäjän oikeuksia vaativa toiminta on yksiselitteisen kiellettyä!**

Ohjelmien asentaminen pakettimanagerin avulla

Omalla tietokoneella, minkä pääkäyttäjä olet, ei luonnollisesti mikään estä ohjelmien asentamista, poistamista tai päivittämistä. UNIX/Linux -ympäristössä ohjelmien ja ohjelmistojen asentaminen tapahtuu pääsääntöisesti käyttämällä pakettienhallintaohjelmaa. Pakettienhallintaohjelma pitää automaattisesti huolta ohjelmistojen päivityksistä sekä ohjelmien keskinäisten riippuvuuksien ylläpitämisestä.

Pakettimanagereita on luonnollisesti sekä graafisia että komentorivipohjaisia, ja tässä oppaassa keskitymme luonnollisesti komentorivityökaluihin. Eri Linux-jakeluissa on käytössä erilaisia pakettimanagereita: Debian- ja Ubuntu- pohjaisissa distribuutioissa on käytössä **APT** (The Advanced Packaging Tool) -niminen pakettienhallintajärjestelmä, jota käytetään komennolla **apt-get**.

Taulukko 9: apt-get:n käyttö.

Komento	**Tarkoitus**
apt-get install ohjelma	asenna ohjelma *ohjelma*
apt-get remove ohjelma	poista ohjelman *ohjelma* asennus
apt-get update	päivitä pakettienhallinnan tietokanta
apt-get upgrade	päivitä järjestelmään asennetut ohjelmistot

Ohjelmien asennus on kuitenkin UNIX-järjestelmissä tiukasti säänneltyä. Kokeillaanpa käyttää edellä mainittua asennuskomentoa ohjelmalle **cowsay**:

$ apt-get install cowsay

E: Could not open lock file /var/lib/dpkg/lock - open (13: Permission denied)
E: Unable to lock the administration directory (/var/lib/dpkg/), are you root?

Permission denied -viesti viittaa siihen, että vaikka oletkin järjestelmän pääkäyttäjä, et saa silti asentaa ohjelmia tai ohjelmistoja noin vain. Tämä johtuu siitä, että järjestelmä haluaa pystyä autentikoimaan käyttäjän järjestelmänvalvojan oikeuksia vaativissa tehtävissä. Kysymys "**are you root?**" kertoo itse vian: Vaikka olisitkin järjestelmän ainoa käyttäjä, et tietoturvasyistä ole automaattisesti kirjautuneena pääkäyttäjäksi. Tästä johtuen kaikkien ohjelmien, hyvän- ja pahantahtoisten (virukset, malware), asennus vaatii aina asennusoikeuksellisen käyttäjän salasanan syöttämisen.

sudo

Huomio: **Älä kokeile** seuraavia komentoja **yliopiston Linux-koneilla**.

Jotta tietoturvasyistä UNIX/Linux-käyttäjä ei olisi jatkuvasti kirjautuneena pääkäyttäjän tunnuksilla järjestelmään modernit Linux-jakelut toteuttavat hyvin yksinkertaista tietoturvaprotokollaa: Pääkäyttäjän (root) tunnuksella **ei ole mahdollista kirjautua sisään**. Pääkäyttäjän oikeuksia vaativia komentoja voi tällaisissa järjestelmissä ajaa komennolla **sudo** (substitute user, do). Lisäämällä komennon *sudo* minkä tahansa komennon eteen tarkoittaa, että käyttäjä haluaa suorittaa ko. komennon pääkäyttäjän oikeuksilla.

Ohjelman **cowsay** voi siis asentaa omalle tietokoneelleen komennolla:

$ sudo apt-get install cowsay

Asennuksen onnistumista voi kokeilla koittamalla ajaa ohjelmaa:

$ cowsay mooh mooh mothertruckers

sudo-komentoa tarvitaan siis kaikissa järjestelmän hallintaan liittyvissä tehtävissä. Peruskäyttäjän ei siis tarvitse käyttää tätä komentoa, mikäli UNIX/Linux -järjestelmällä on joku muu ylläpitäjä, kuin käyttäjä itse. **sudo** antaa siis käyttäjälle hetkellisesti laajimmat mahdolliset käyttöoikeudet. Luvattomasta **sudo**-komennon käytöstä lähetetään järjestelmän ylläpitäjälle varoitusviesti.

9.2 UNIX-ohjelmistopakettien kääntäminen

Tämä kappale käsittelee ohjelmien kääntämistä (ja asentamista) suoraan lähdekoodista. Huomaa jälleen, että ohjelmien asentaminen ei ole mahdollista ilman pääkäyttäjän oikeuksia järjestelmään. Käyttäjä voi joka tapauksissa ladata ohjelmia kotihakemistoonsa, jolloin niitä voi käyttää ainoastaan henkilökohtaisesti.

Edellisessä kappaleessa käsitelty pakettienhallintaohjelma on yleisesti ottaen paras ja suositeltavin tapa kartuttaa UNIX-järjestelmän sovellusvalikoimaa. Joskus pakettienhallintaa ei kuitenkaan ole mahdollista käyttää. Tämä voi johtua esimerkiksi siitä, ettei käyttäjällä ei ole ohjelmien asentamiseen vaadittavia oikeuksia (pääkäyttäjä). Voi myös olla, ettei ohjelma tai ohjelmisto ole pakettienhallinnan kautta saatavilla. Näissä tapauksissa käyttäjän on

ladattava verkosta ohjelman lähdekoodi sekä käännettävä ja mahdollisesti asennettava ohjelma käsin.

Ohjelmistojen asentamiseen lähdekoodista vaaditaan siis muutama, mukava askel:

- Paikallista ja lataa lähdekoodi (joka on usein pakattu)
- Pura lähdekoodi
- Käännä koodi
- Asenna ajettava tiedosto (mikäli olet pääkäyttäjä)
- Aseta polut asennushakemistoon (mikäli olet pääkäyttäjä)

Yllä olevista askeleista vaativin on koodin kääntämisvaihe.

Lähdekoodin kääntäminen

Kaikki korkeatasoisten kielten koodit täytyy muuntaa muotoon, jota tietokone ymmärtää. Esimerkiksi C-kielen lähdekoodi käännetään alempitasoiseksi välikieleksi, jota kutsutaan symboliseksi konekieleksi. Tämän vaiheen tekemä symbolinen koodi muutetaan konekieleksi, joka sisältää koneelle suoraan ymmärrettäviä koodipätkiä. Ohjelman kääntämisen loppuvaihe pitää sisällään konekielisen koodin linkittämisen koodikirjastoihin, jotka sisältävät tiettyjä sisäänrakennettuja toimintoja. Loppuvaihe tuottaa viimeinkin ajettavan ohjelman.

Kaikkien näiden vaiheiden tekeminen käsin on monimutkaista ja tavallisen käyttäjän taitojen yläpuolella. Tähän tarkoitukseen on kehitetty lukuisia apuvälineitä, jotka yksinkertaistavat yllä mainittuja askeleita niin ohjelmoijille kuin loppukäyttäjillekin.

make ja Makefile

make -komento mahdollistaa ohjelmoijan hallita kerralla ohjelmistojen ryhmiä tai suuria ohjelmia. Se avustaa suurien ohjelmien kehittämistä pitämällä lukua siitä, mitä osuuksia koko ohjelmasta on muutettu, ja kääntäen ainoastaan ne osat, jotka ovat muuttuneet edellisen kääntämisen jälkeen. **make**-ohjelma saa kokoelman kääntämissääntöjä tekstitiedostolta nimeltä **Makefile**, joka sijaitsee lähdetiedostojen kanssa samassa hakemistossa. Se sisältää tietoa siitä, kuinka ohjelmisto tulee kääntää, ja liitetäänkö ajettavaan ohjelmaan testausmahdollisuus (debugging). Siinä on myös ohjeet, mihin asentaa valmistuneet, käännetyt binääritiedostot (ajotiedostot), manuaalisivut, datatiedostot, kokoamistiedostot jne. Jotkut ohjelmistopaketit vaativat **Makefile**-ohjelman käsin editointia asennushakemiston ja muiden muuttujien asettamiseksi.

configure

Kun UNIXin variaatioiden määrä lisääntyi, kaikille versioille sopivien ohjelmien kirjoittaminen muuttui vaikeammaksi. Kehittäjillä ei usein ollut pääsyä jokaiseen järjestelmään, ja joidenkin järjestelmien erityispiirteet muuttuivat versiosta toiseen. GNU konfigurointi ja ohjelman versiojärjestelmä yksinkertaistaa ohjelmien rakentamisen lähdekoodien mukaan luokiteltuna. Ohjelmat pyritään rakentamaan käyttäen yksinkertaista ja standardisoitua, kahden askeleen prosessia. Ohjelman kehittäjän ei tarvitse asentaa mitään erikoistyökaluja muodostaakseen ohjelman.

configure- shell-skripti yrittää arvata oikeita arvoja sekalaisille järjestelmästä riippuville muuttujille joita kääntämisen aikana on käytetty. Se käyttää kyseisiä arvoja luodakseen **Makefile**n paketin jokaiselle hakemistolle.

Yksinkertaisin tapa kääntää paketti:

1. Mene **cd**:llä paketin lähdekoodin sisältävään hakemistoon
2. Kirjoita **./configure** konfiguroidaksesi paketin omalle järjestelmällesi sopivaksi
3. Kirjoita **make** paketin kääntämiseksi
4. Halutessasi kirjoita **make check**, jolloin kone ajaa mahdolliset paketin mukana tulevat testaukset.
5. Käytä **make install** -komentoa ja asenna ohjelmat sekä kaikki datatiedostot ja dokumentaatio.
6. Halutessasi kirjoita **make clean** ja poista ohjelman binäärikoodit ja konekieliset tiedostot lähdekoodihakemistosta.

Konfigurointityökalu tukee laajaa valikoimaa optioita. Useimmiten **--help** -optiolla saa listan mielenkiintoisista optioista tietylle konfigurointiskriptille.

Ainoat yleiset optiot, joita todennäköisesti koskaan tullaan käyttämään, ovat **--prefix** ja **--exec-prefix.** Näitä käytetään tarkentamaan asennushakemistoja.

--prefix- optiolla nimetty hakemisto säilyttää koneesta riippumattomia tiedostoja kuten dokumentti-, data- ja konfigurointitiedostoja.

--exec-prefix -optiolla (joka on useimmiten alihakemisto **--prefix**-hakemistossa) nimetty hakemisto pitää sisällään koneesta riippuvia tiedostoja kuten ajotiedostoja.

9.3 Lähdekoodin lataaminen

Tätä esimerkkiä varten ladataan osa ilmaista ohjelmistoa joka muuttaa arvoja eri mittausyksiköiden välillä. Aloita luomalla lataushakemisto

$ mkdir lataukset

Lataa ohjelmisto osoitteesta
http://www.ee.surrey.ac.uk/Teaching/Unix/unix7.html
kohdasta 7.2: Downloading software code, ja sen jälkeen tallenna se uuteen lataushakemistoosi.

9.4 Lähdekoodin purkaminen

Mene *lataukset* -hakemistoon ja listaa sen sisältö

$ cd download
$ ls -l

Kuten voit nähdä, tiedostonimi loppuu päätteellä *tar.gz*. **tar** -komento muuttaa useita tiedostoja ja hakemistoja yhdeksi tar -tiedostoksi. Tämä on pakattu käyttäen **gzip** -komentoa, jolloin muodostuu *.tar.gz* -tiedosto

Avaa tiedosto **gunzip** -komentoa käyttäen. Tämä luo *.tar* -tiedoston.

$ gunzip units-1.74.tar.gz

Sen jälkeen pura *tar* -tiedoston sisältö.

$ tar -xfv units-1.74.tar

Listaa jälleen *lataukset*-hakemiston sisältö ja siirry sen jälkeen *units-1.74* -alihakemistoon.

$ cd units-1.74

Saman purkamisen voi tehdä myös yksittäisellä komennolla, jolloin homma hoituu siistimmin:

$ tar -xzvf

Tämä komento purkaa tar-tiedoston, mutta option **z** avulla purku ajetaan myös **gunzip**in (tai toiselta nimeltään **gzip**) läpi.

9.5 Makefile -tiedoston konfigurointi ja luominen

Ensimmäiseksi tulee lukea huolellisesti *README - ja INSTALL* -tekstitiedostot (käytä less -komentoa). Nämä sisältävät tärkeää tietoa tiedoston kääntämisestä ja ohjelman ajamisesta. Aikaisemmin ladattu Units -paketti käyttää GNU -konfigurointijärjestelmää lähdekoodin kääntämiseen. Koska oletuksena tiedostot asennetaan johtoverkon alueelle johon opiskelijoilla ei ole kirjoituslupaa, asennushakemisto täytyy määrittää erikseen. Tätä varten meidän pitää luoda ja asentaa hakemisto kotihakemistoosi.

$ mkdir ~/units174

Aja sen jälkeen konfigurointityökalu asettaen asennuspolun seuraavanlaiseksi:

$ ./configure --prefix=$HOME/units174

Huom: $HOME-muuttuja on esimerkki ympäristömuuttujasta. $HOMEn arvo on polku kotihakemistoosi. Kirjoita vain

$ echo $HOME

näyttääksesi muuttujan sisällön. Ympäristömuuttujista puhutaan myöhemmin lisää.

Jos **configure** on suoritettu oiken, se on luonut *Makefile:*n kaikilla tarvittavilla vaihtoehdoilla. Halutessasi voit tarkastella tiedostoa (käytä **less**-komentoa), mutta älä editoi sen sisältöä.

9.6 Paketointi

Kokoa paketti ajamalla **make** -komento

$ make

Minuutin tai parin jälkeen (tietokoneen nopeudesta riippuen), ajotiedostot luodaan. Voit tarkistaa onnistuiko kääntäminen kirjoittamalla

$ make check

Jos kaikki on kunnossa, voit asentaa paketin (joissain tilanteessa voidaan vaatia sudo-oikeudet).

$ make install

Tämä asentaa tiedostot aikaisemmin luotuun *~/units174* -hakemistoon.

9.7 Ohjelmiston ajaminen

Olettaen aikaisempien harjoitusten onnistuneen, olet nyt valmis ajamaan ohjelmiston.

$ cd ~/units174

Jos listaat *units* -hakemiston sisällön, näet muutamia alikansioita, kuten taulukossa 10.

Taulukko 10: units-hakemiston sisältö.

bin	Binäärimuotoiset ajotiedostot
info	GNU -info
man	manuaalisivut
share	Jaettu data

Ohjelman ajamiseksi siirry *bin* -hakemistoon ja kirjoita

$ ./units

Esimerkkinä muunna 6 jalkaa metreiksi:

You have: 6 feet
You want: metres
** 1.8288*

Jos saat vastaukseksi 1.8288, onnittelut, ohjelma toimi. Tarkastellaksesi eri yksiköitä joiden välillä ohjelman on mahdollista muuntaa, tutki *data* -tiedostoa hakemistossa *share (*lista on suht kattava).

Koko dokumentaatiota tarkastellaksesi, siirry *info* -hakemistoon ja kirjoita

$ info --file=units.info

9.8 Tarpeettoman koodin poistaminen

Kun osaa koodia kehitetään, ohjelmoijalle on hyödyllistä sisällyttää debuggaukseen liittyvät tiedot ajotiedostoon. Tällä tavoin jos ohjelman ajamisessa on ongelmia, koodaaja voi ladata ajotiedoston debuggausohjelmaan ja metsästää kaikki virheet.

Tämä on ohjelmoijalle hyödyllistä, mutta tarpeetonta käyttäjälle. Voimme olettaa, että valmis ja ladattavissa oleva paketti on valmiiksi testattu ja virheet on etsitty. Kuitenkin ylläolevaa ohjelmaa kääntäessä debugaustiedot oli käännetty myös valmiiseen ajotiedostoon. Koska tietojen tarvitseminen on epätodennäköistä, ne voidaan riisua valmiista ajotiedostosta. Yksi näin saatavista eduista on huomattavasti pienempi tiedosto, jonka pitäisi suorittaa itsensä hieman nopeammin.

Seuraavaksi tutkimme binääritiedoston kokoa ennen ja jälkeen. Siirry ensin *units*-ohjelman asennushakemiston *bin* -hakemistoon

$ cd ~/units174/bin
$ ls -l

Kuten voit nähdä, tiedosto on kooltaan yli 100 kilobittiä. Voit saada lisää tietoa tiedoston tyypistä käyttämällä **file** -komentoa.

$ file units
units: ELF 32-bit LSB executable, Intel 80386, version 1, dynamically linked (uses shared libs), not stripped

Käytä **strip** -komentoa kaiken turhan, kuten debug -tietojen ja rivinumerointien poistamiseen

$ strip units
$ ls -l

Näet, että tiedoston koko on nyt 36 kb - kolmanneksen alkuperäisestä koodista. Kaksi kolmannesta binääritiedostosta oli debug-koodia!

Tarkista tiedoston tiedot uudestaan

$ file units
units: ELF 32-bit LSB executable, Intel 80386, version 1, dynamically linked (uses shared libs), stripped

Joskus **make**-komentoa voi käyttää asentamaan valmiiksi riisuttuja kopioita kaikista binääritiedostoista, kun asennat paketin. **Make install** -komennon sijasta kirjoita yksinkertaisesti **make install-strip.**

Luku 10: Shell scripting

Tähän asti tässä oppaassa olemme keskittyneet käyttämään UNIX-komentoja interaktiivisessa komentotulkissa. Kuten olemme huomanneet, nämä komennot ovat parhaimmillaan erittäin tehokkaita ja aikaa säästäviä. Useat tehtävät voisi kuitenkin automatisoida siten, ettei yksittäisiä komentoja tarvitse kirjoittaa käsin joka ikinen kerta. **Shell scriptit** auttavat tässä.

Shell script on kokoelma UNIX-komentoja tallennettuna tavalliseen tekstitiedostoon. Skriptitiedoston nimessä on usein pääte **.sh**, mutta kuten Unix-tiedostojärjestelmässä yleensäkään, eivät tiedostopäätteet ole pakollisia. Skriptitiedosto käynnistetään kuten mikä tahansa komentoriviohjelma, tosin sillä erotuksella, että käyttäjän omassa kotihakemistossa sijaitsevaan tiedostoon pitää suorittaessa viitata eksplisiittisesti käyttäen sen täydellistä polkua. Esimerkiksi ajettaessa samassa hakemistossa sijaitsevaa skriptiä, komennetaan

$ ./script.sh

sillä komentotulkki alkaa etsiä annettua komentoa vastaavaa ohjelmaa automaattisesti suoritettavien ohjelmien oletushakemistosta (*/usr/bin*, /bin, /usr/sbin jne). Siksi skriptin (ohjelman) sijainti pitää erikseen kertoa komennon yhteydessä.

10.1 Hello bash

Kirjoitetaan lyhyt shell script -tiedosto, jonka suoritus tulostaa “Hello bash” ruudulle.

```
#!/bin/bash
echo “Hello bash”
```

Tallennetaan tiedosto nimellä *hello.sh* ja muokataan tiedoston oikeuksia siten, että se voidaan suorittaa. Lopuksi suoritetaan skriptitiedosto:

```
$ chmod u+x hello.sh
$ ./hello.sh
Hello bash
$
```

Onnistuneesta skriptin kirjoittamisesta seuraa edellisen kaltainen tulostus. Kuten huomaamme, komentosarja ei ole kovinkaan monimutkainen sisältäen vain yhden *echo*-komennon. Mutta mitä tarkoittaa tiedoston ensimmäinen, #-kommenttimerkillä alkava rivi?

10.2 Tiedoston-ajo-oikeudet ja hashbang

Edellisen esimerkin ensimmäistä riviä kutsutaan #- ja !-merkkien englanninkielisten nimitysten mukaan termillä **hashbang**. **#** itsessään on kommenttimerkki - notaatio on sama monessa tulkattavassa ohjelmointikielessä (esim. Ruby, Python, Perl). Siten ensimmäistä riviä ei itse asiassa suoriteta ollenkaan.

Hashbang-rivi sijaitsee aina skriptitiedoston alussa. Sen tehtävä on kertoa komentotulkille sen ohjelman polku, jolla skripti halutaan suorittaa. Tässä tapauksessa haluamme suorittaa skriptin Bash-komentotulkissa, mutta polun voisi asettaa osoittamaan myös toiseen komentotulkkiin (Z Shell, C Shell jne).

Huomionarvosta on myös, että komentokehotteessa suoritettavia skriptejä voi kirjoittaa myös muilla tulkattavilla ohjelmointikielillä. Esimerkiksi Pythonin, Perlin tai Rubyn käyttö on yhtä lailla mahdollista shell skriptien kirjoittamiseen. Eri ohjelmointikieliä käsitellään vielä lyhyesti tämän kappaleen lopuksi.

10.3 Tulostus ja syöttö

Edellisestä “Hello bash” -esimerkistä huomasimme, että tulostus shell scriptissä tapahtuu samalla *echo*-komennolla kuin interaktiivisen komentotulkin puolella. Yksinkertaisimmillaan skripti voikin koostua sarjasta erilaisia Unix-komentoja.

```
#!/bin/bash
# caldate.sh
# ohjelma tulostaa kuukauden kalenterin sekä
# näyttää päiväyksen ja tarkan kellonajan

cal
date +%A-%d-%B-%Y
date +%T
echo
```

Yllä oleva esimerkkiskripti toimii juurikin vain suorittamalla peräkkäin erinäisiä komentoja. Se ei vaadi käyttäjältä syötteitä, eikä ohjelmien suoritukset vaikuta toisiinsa.

Jos käyttäjältä halutaan kysyä syötettä, se tapahtuu käyttämällä **read**-komentoa:

```
#!/bin/bash
# friend.sh
#Ohjelma kysyy käyttäjältä syötettä, ja tulostaa sen

echo “Hello friend, what is your name:“
read name
echo “Hello $name, pleasure to meet you!”
```

10.4 Muuttujat ja komentoriviparametrit

Edellisessä esimerkissä luettiin käyttäjän antama syöte, tallennettiin se muuttujaan ja lopuksi tulostettiin muuttujan arvo ruudulle. Muuttujat komentotulkkiskriptissä toimivat saman kaltaisesti kuin useissa muissa tulkattavissa ohjelmointikielissä. Muuttujia ei tarvitse erikseen esitellä ennen niihin sijoittamista. Muuttujan nimi tulee alkaa kirjaimella tai **_**-merkillä, eikä se saa sisältää erikoismerkkejä. Lisäksi nimet ovat merkkikoosta riippuvaisia (case-sensitive).

Muuttujaan sijoitetaan arvo **=**-operaattorin avulla. Muuttujaan sijoittamisessa on rajoitteena, että **=**-merkin ympärillä ei saa olla välilyöntejä

```
a=1 #ok
a =1 # väärin
a=   1 #väärin
a = 1 #väärin
```

Muuttujan voi myös alustaa tyhjäksi:

```
muuttuja=
muuttuja=""
```

Tämän lisäksi muuttujan nimi ei saa olla mikään varattu sana tai muu Unix-muuttuja. Muuttujista puhutaan lisää liitteessä 1.

Kun muuttuja on otettu käyttöön, sen sisältämään arvoon pääsee käsiksi lisäämällä muuttujan eteen **$**-merkin:

```
muuttuja=42
echo $muuttuja
echo "Muuttujan arvon, $muuttuja voi tulostaa vaikkapa merkkijonon sisään"
```

Muuttujan nimi yksinään on siis vain osoitin muuttujan itsensä sijaintiin - ei sen sisältöön. Muuttujaan sijoitettuun arvoon viitataan taas **$**-operaattorilla.

Käyttäjä voi välittää tietoa skriptille myös komentoriviparametrien avulla. Komentoriviparametrit tallennetaan automaattisesti shell-muuttujiin ***$(numero)***, jolloin ensimmäiseen komentoriviltä annettuun parametriin pääsee käsiksi viittauksella ***$1***, toiseen viittauksella ***$2*** ja niin edelleen. Näitä shell-muuttujia on enintään yhdeksän (9) kappaletta. Komentoriviparametrien lukumäärä puolestaan on shell-muuttujassa **$#**.

Esimerkkinä tästä rakennamme nyt yksinkertaisen summain-ohjelman, joka ottaa parametrinä kaksi lukua ja laskee ne yhteen. Komentotulkki itse ei tue aritmeettisia operaatioita (toisin kuin monen muun ohjelmointikielen tulkit), joten yhteenlaskua varten käytämme **expr**-nimistä ohjelmaa. Expr ottaa paremetrinä aritmeettisen lausekkeen ja tulostaa ko. laskutoimituksen tuloksen.

```
#!/bin/bash
# summain.sh
# Yksinkertainen summain

expr $1 + $2
```

Skripti ajetaan siis antamalla komentoriviltä kaksi parametria

```
$ ./summain.sh 2 2
4
```

10.5 Vertailuoperaattorit ja valintarakenne

IF-ELIF-ELSE

Valintarakenteen syntaksin avainsanat ovat: **if**, **elif**, **else**, **then** ja **fi**. Lisäksi ehtolause kiedotaan hakasulkujen ([ja]) sisään. Huomaa, että hakasulut on erotettava muista merkeistä välilyönneillä. IF-ELIF-ELSE -vertailurakenne on siis seuraavanlainen:

```
if [ 1 ]
then
        echo "Tosi ehto!"
elif [ 0 ]
then
    echo "tätä ei pitäisi tulostua"
else
    echo "eikä tätä"
fi
```

Esimerkin pitäisi tulostaa “Tosi ehto!”, sillä ehtona ykkönen on tosi toisin kuin nolla. Else-haaraan koodi ei voi edetä, sillä toinen edeltävistä ehdoista on joka tapauksessa tosi. Kuten muissakin ohjelmointikielissä, vertailurakenteessa haaroja voi olla yksi (pelkkä if), kaksi (if-else), kolme (if-elif-else) tai enemmän.

case

case-rakenne on toinen tapa toteuttaa vertailu. Bash-skriptissä case alkaa ilmauksella *case $muuttuja in*, jonka jälkeen muuttujan mahdolliset arvot kirjoitetaan muotoon *arvo)*. Tämän jälkeen kirjoitetaan lauseet, jotka suoritetaan mikäli vertailu arvon ja muuttujan välillä on tosi. Yhden case-tapauksen lopetusmerkkinä toimivat kaksi puolipistettä (;;). Mikäli mikään vertailu case-rakenteen sisällä ei ole tosi, voidaan ***-merkillä ilmaista rakenteen “else”-haara. case-rakenne lopetetaan avainsanaan *esac*.

```
age=5

case $age in
    1) echo "a year old" ;;
    2) echo "two years old" ;;
    5) echo "a toddler" ;;
    25) echo "a jaded teaching assistant"  ;;
    *) echo "something else, young or old who knows" ;;
esac
```

IF-valinnan yhteydessä hakasulkeiden sisään laitetaan jokin todeksi tai epätodeksi osoittautuva vertailu. Vertailuoperaatioita on **aritmeettisia**, **loogisia** sekä **merkkijoinoille** tehtävä yhtäsuuruusvertailu.

Aritmeettiset vertailut

Aritmeettisilla vertailuilla tarkoitetaan matemaattisia suuruus- ja yhtäsuuruusvertailuja. Vertailuoperaattorit poikkeavat useista muista ohjelmointikielistä, sillä =, < ja > -symbolit eivät ole käytössä.

- *-lt* Kokonaisluku on pienempi kuin verrokki
- *-gt* Kokonaisluku on suurempi (a > b)
- *-le* Kokonaisluku on pienempi tai yhtäsuuri (a <= b)
- *-ge* Kokonaisluku on suurempi tai yhtäsuuri (a >= b)
- *-eq* Kokonaisluvut ovat yhtäsuuret (a = b)
- *-ne* Kokonaisluvut ovat eri suuret (a != b)

Tosia vertailuja ovat esimerkiksi

```
[ 1 -lt 2 ]  # 1<2
[ 3 -eq 3 ]  # 3 == 3
```

Loogiset vertailut

Loogisia vertailuoperaattoreita käytetään muiden vertailuoperaattoreiden kanssa.

- ! Negaatio-operaattori
- -a *Ja*-operaattori (leikkaus)
- -o *Tai*-operaattori (unioni)

Esimerkiksi tosia vertailuja ovat

```
[  ! 1 -gt 2  ]  # Negaatio 1 suurempi kuin 2 (== 1 pienempi kuin 2)
[  1  -lt  2  -a  3  -gt  2  ]  # 1<2 JA 3>2
```

Merkkijonovertailu

Kahden merkkijonon vertailu toisiinsa tapahtuu **=**-operaattorilla. Vertailu on tosi mikäli verrattavat merkkijonot on identtiset. Esimerkki merkkijonovertailusta alla. Huomaa, että käytettäessä **=**-merkkiä vertailuoperaattorina operandit voi erottaa operaattorista välilyönnillä kumminkin puolin.

```
nakki="nakki"
keppi="nakki"
[ $nakki = $keppi ]  # vertailun pitäisi olla tosi
```

Yhtäsuuruuden lisäksi merkkijonolle voi tehdä myös vertailuja tarkistamaan merkkijonomuuttujan olemassaoloa tai tyhjäksi alustusta. Näitä emme kuitenkaan käsittele tämän oppaan puitteissa tarkemmin.

Tietotyypeistä Bash-skripteissä

Kuten aikaisemmista kappaleista voimme havaita, Bash-komentotulkin skripteissä toimivia tietotyyppejä ovat ainakin kokonaisluvut (integer) ja merkkijonot (string). Komentotulkki ei kuitenkaan esimerkiksi osaa suoraan käsitellä kokonaislukuja, summain-esimerkissä käytimme **expr**-ohjelmaa hoitamaan aritmetiikan. Aloittelevan Bash-ohjelmoijan onkin syytä varoa näitä sudenkuoppia ja keskittyä käsittelemään lähinnä merkkijonomuotoisia muuttujia.

Bash-skriptissä on mahdollista käyttää kokonaislukuja, liukulukuja, merkkijonoja sekä boolean-tyyppisiä muuttujia, mutta emme käsittele erilaisia tietotyyppejä syvällisemmin tässä oppaassa.

10.6 Toistorakenteet

Toistorakenteet ovat olennainen ohjelmointikielen perusominaisuus. Bash-skripteissä on mahdollisuus tehdä sekä **for**- että **while**-silmukoita. For-rakenteella askelletaan listaa elementti kerrallaan. While-silmukka taas toistuu niin kauan, kuin sille annettu ehto on tosi.

for

For silmukkaa voi käyttää kahdella vaihtoehtoisella syntaksilla

*for **muuttuja** in **lista***
*do **komento tai monta komentoa***
done

tai

*for ((**lauseke1** ; **lauseke2**; **lauseke3**))*
*do **komento tai monta komentoa***
done

Yksinkertaisimmillaan askellettavan listan alkiot voidaan kirjoittaa for silmukkaan sisälle välilyönnillä eroteltuna käyttäen ensin esiteltyä syntaksia:

```
for i in 1 2 3 4
do echo $i
done
```

Vaihtoehtoinen esitystapa samalle silmukalle olisi:

```
for (( i=1 ; i < 5; i++ ))
do echo $i
done
```

while

while-silmukan syntaksi on seuraava:

```
while [  ehto  ]
do komento
done
```

Edellinen esimerkki while-silmukan avulla olisi seuraavanlainen:

```
i=1
while  [ $i  -lt  5 ]
do
        echo $i
        i=`expr $i + 1`
done
```

Huomaa jälleen, että shell-skriptissä aritmeettiset operaatiot ovat hankalia suoritettavia, ja joudumme jälleen käyttämään **expr**-komentoa apuna muuttujan arvon korottamiseen. Expr-lauseen ympärillä olevat gravismerkit (`) tarkoittavat komennon ajamista muuttujaan sijoituslauseen sisällä - komentojen ajamisesta shell-skriptin sisällä puhutaan seuraavassa kappaleessa.

until

until-rakenne on syntaksiltaan samanlainen kuin while, mutta toimintaperiaatteeltaan päinvastainen. While-silmukka pyörii kunnes sille annettu ajoehto lakkaa olemasta tosi. until-silmukkaa taas suoritetaan, kunnes sille annettu lopetusehto on tosi.

Luvut 1-4 tulostava esimerkkimme until-rakenteen avulla kirjoitettuna olisi:

```
i=1
until [  $i -ge 5  ]
do
  echo $i
  i=`expr $i + 1`
done
```

10.7 Komentojen ajaminen skriptissä

Kuten olemme huomanneet, komentorivikäyttöliittymän toimintaperiaate on nelivaiheinen. Toimintaa kutsutaan nimellä Read-Evaluate-Print-Loop (REPL)[2]:

1. Read - Luetaan käyttäjältä syöte
2. Evaluate - Suoritetaan käyttäjän antama komento
3. Print - Näytetään komennon / ohjelman tuloste
4. Loop - Palataan tilaan 1 odottamaan käyttäjän syötettä

[2] http://en.wikipedia.org/wiki/Read-eval-print_loop

Komentojen ajamiseen, tapahtui se interaktiivisesti komentotulkissa tai shell-skriptin avulla, kuuluu siis komennon tuottaman tulosteen tulostaminen käyttäjälle. Shell-skripteissä tulee kuitenkin usein tarve tallentaa komentojen tuloste talteen, eikä kaikkien komentojen tulosteita ole aina tarpeellista näyttää käyttäjälle.

Komentojen tulosteen tallentaminen muuttujiin myöhempää käyttöä varten kutsutaan englanninkielisellä termillä *command substitution*. Törmäsimme tällaiseen jo edellisen kappaleen esimerkissä, jossa tarvitsimme *expr*-komennon laskemaa lukuarvoa silmukan lopetusehdon tarkastamiseksi. Komentojen tulosteiden tallentaminen muuttujaan tapahtuu joko sijoittamalla komento gravismerkkien (` `) sisään tai sulkeiden sisään dollarimerkin jälkeen:

```
# command.sh
a=$(echo "hello world")
b=`echo "these are command substitutions"`
echo "$a, $b"
```

Skriptin tuloste on seuraava:

```
$ ./command.sh
hello world, these are command substitutions
```

Huomaa, kuinka *command substitution* syö sen sisälle asetetun *echo*-komennon tulostaman rivinvaihdon - normaalistihan kaksi peräkkäistä *echo*-komentoa tulostuisi kumpikin omalle rivilleen.

10.8 Komentoriviohjelmointi muilla kielillä

Unix-järjestelmissä on usein valmiiksi asennettuna myös muiden ohjelmointikielien tulkkeja. Esimerkiksi GNU/Linux -käyttöjärjestelmissä Python-kielen versio 2 on laajalti käytössä ja systeemiohjelmointi Python-kielellä on suosittua. Python on kuitenkin suhteellisen nuori ohjelmointikieli ja sen rinnalla Linux-distribuutioissa käytetään myös Perliä. Ylläpitäjät voivat tietysti asentaa oman mieltymyksensä mukaisia ohjelmointikielten tulkkeja ja kääntäjiä järjestelmäänsä.

Modernien ohjelmointikielten käyttäminen shell-skriptien tekoon voi olla usein kannattavampaa kuin Bash-skriptaus. Esimerkiksi aloittelevalle ohjelmoijalle vaikkapa Python tai Ruby on usein syntaksiltaan suoraviivaisempaa ja ne mahdollistavat vaikeampien tietojenkäsittelytehtävien suorittamisen pitäen silti ohjelmoijan kosketuksissa komentorivityökaluihin järjestelmäkutsujen avulla.

10.9 Esimerkki: Tiedoston rivien lukumäärän laskeminen Pythonilla

Esimerkkinä komentoriviohjelmoinnista Python-kielellä teemme skriptin, joka toimii samalla tavalla kuin komento *wc -l tiedosto.txt* eli laskee rivien määrän tiedostossa. Esimerkkiajo:

```
$ chmod u+x wcl.py
$ ./wcl.py shakes.txt
63 shakes.txt
$
```

Yksinkertainen, tiedoston rivit laskeva Python ohjelma voisi olla esimerkiksi seuraavanlainen:

```
#!/usr/bin/python
# wcl.py
# a python script that mimics  wc -l
#
import sys
lines = 0
with open(sys.argv[1], "r") as f:
        while(f.readline() != ""):
                lines = ( lambda x: -~x ) (lines)
print lines, sys.argv[1]
```

10.10 Esimerkki: Tiedostojen listaus tiedostopäätteen mukaan Perlillä

```
#!/usr/bin/perl
# filesbytype.pl
# Prints out files that have ending passed as argument
#
opendir(DIR, ".");
@files = grep(/\.$ARGV[0]/, readdir(DIR));
closedir(DIR);
foreach $file (@files) {
   print "$file\n";
}
```

Skripti tulostaa kaikkien tiedostojen nimen, joilla on käyttäjän komentoriviargumenttina antama pääte.

```
$ ./filesbytype.pl py
why-no-run.py
no-run.py
read-input.py
```

10.11 Esimerkki: Palindromi-tiedostonimien listaus Rubyllä

```
#!/usr/bin/ruby
# palindromefiles.rb
# Ruby shell script that goes over a folder
# and prints out any file name that is a palindrome!
#
Dir['*'].each { |name| puts name if name==name.reverse }
```

Skripti tulostaa kaikki tiedostot, joiden nimi itsessään on palindromi. Esimerkiksi:

```
$ echo “palindromi” >> saippuakauppias
$ ./palindromefiles.rb
saippuakauppias
$
```

Luku 11: Säännölliset lausekkeet

Usein tulee tarve etsiä tai etsiä ja korvata tiettyjä merkkijonoja eri tarkoituksissa: Oli kyseessä sitten tekstitiedoston käsittely tai tiedostopolun hakeminen, etsiminen ja korvaaminen on aina sama tehtävä. Tätä tarkoitusta varten UNIX-järjestelmät tukevat *säännöllisiä lausekkeita.* Säännölliset lausekkeet (regular expressions) muodostuvat tarkasti määritellystä kielestä, jonka avulla pystymme ilmaisemaan erilaisia kielioppisääntöjä. Näitä kielioppisääntöjä puolestaan voi käyttää useissa ohjelmistoissa haku- ja korvausehtojen muodostamiseen.

11.1 Mitä ovat säännölliset lausekkeet?

Säännölliset lausekkeet on spesifioitu POSIX-standardissa ISO/IEC 9945-2:1993. Säännölliset lausekkeet noudattavat formaalia, tarkasti määriteltyä ja yksikäsitteistä kielioppia, joka on suunniteltu mahdollisimman kompaktiksi.

Yksinkertaisimmillaan säännöllinen lauseke voi olla merkkijono, joka vastaa täydellisesti, 1:1 haettavaa sanaa. Esimerkiksi voimme tarkentaa *ls*-komennon tulostetta vastaamaan hakuehtona annettavaa tiedostonimeä tenttimateriaalikansiossa:

$ ls | grep giraffe
giraffe.txt

Erikoismerkityksellisten merkkien käyttö puolestaan mahdollistaa laajempien hakuehtojen muodostamisen. Olemme törmänneet tällaiseen hakuun jo aiemmin jokerimerkkien yhteydessä: Jokerimerkki tarkoitti, että kysymysmerkki (?) vastaa hakuehdossa mitä tahansa muuta merkkiä. Säännöllisten lausekkeiden syntaksi poikkeaa tosin jokerimerkeistä hieman, sillä sekä kysymysmerkillä (?) että tähdellä (*) on oma, erilainen erikoismerkityksensä. Esimerkiksi mitä tahansa muuta merkkiä tarkoittava erikoismerkki säännöllisessä lausekkeessa on **piste** (.).

Erikoismerkkien käyttötarkoitukset on listattu tämän luvun lopussa.

Esimerkki: Demonstroidaan säännöllisten lausekkeiden näppäryyttä hakuehtojen luomisessa. Luodaan tätä varten muutama tiedosto, joiden nimet ovat samankaltaiset.

$ echo “kissa” >> cat
$ echo “hattu” >> hat
$ echo “tsätti” >> chat
$ echo “bileet” >> hipat

Oletetaan, että haluamme saada tulostettua ruudulle tiedostolistauksen, joka sisältää kolme ensimmäistä luomaamme tiedostoa (cat, hat ja chat), mutta ei viimeistä (hipat, bileet on siis peruttu). Pelkkä sääntö “at” ei siis toimi:

$ ls | grep at

cat
chat
hat
hipat

Sen sijaan käytämme säännöllisten lausekkeiden kielioppia, jonka avulla voimme tarkasti ilmaista haluavamme vain at-päätteisiä sanoja, joissa at-ilmausta edeltää kirjaimet c tai h:

ls | grep "[hc]at"

Huomaat ehkä tulosteessa, kuinka grep maalaa hakuehtoa vastaavan tekstin, minkä perusteella kyseinen rivi on tulostettu. Kuten huomaamme, hakuehtomme ei ole aivan täydellinen, sillä lauseke vastaa merkkijonoja, jotka sisältävät täsmälleen merkkijonot “cat” tai “hat”. Täsmällinen hakuehto esimerkkitapauksessa olisikin

.[hc]at*

jossa piste tarkoittaa mitä tahansa merkkiä ja tähti edeltävän jokerimerkin (pisteen) toistumista haettavassa merkkijonossa (eli toisin sanoen mikä tahansa merkki saa toistua 0-n kertaa merkkijonossa).

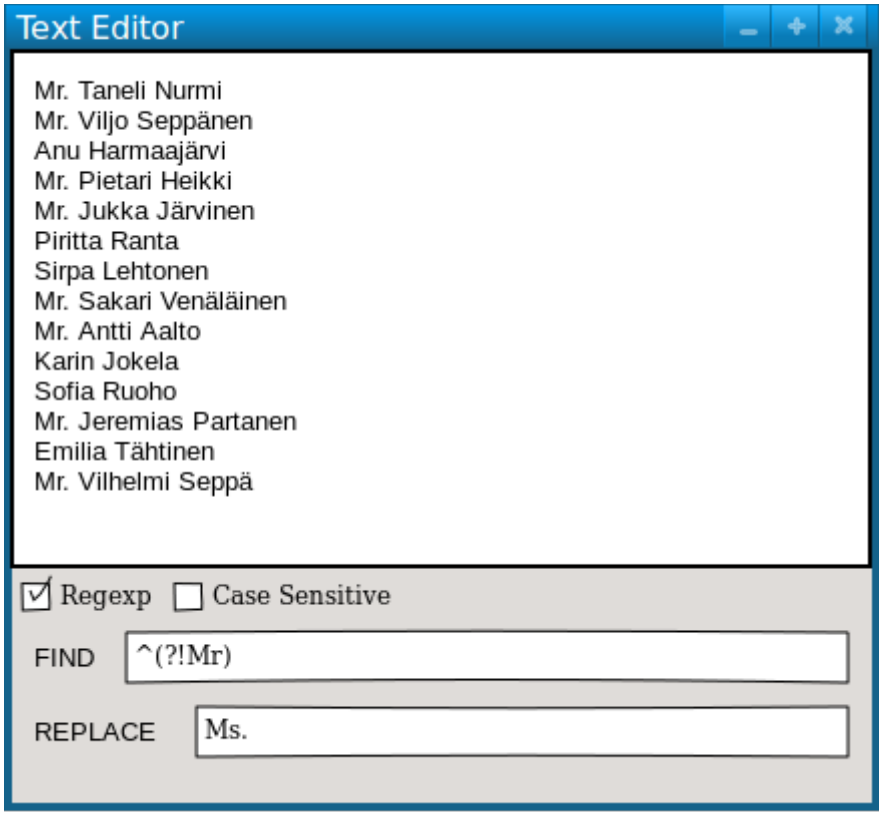
Säännöllistä lauseketta käsittelevä ohjelma puolestaan rakentaa lausekkeesta epädeterministisen äärellisen automaatin, joka muutetaan deterministiseksi äärelliseksi automaatiksi. Tämän jälkeen ohjelma vertaa sille parametrina annettua tekstiä automaatin muodostamaan kielioppiin ja palauttaa tuloksena kielioppia vastaavat osatekstit.

Lisäinfoa automaateista voi katsoa vaikkapa wikipediasta:
http://en.wikipedia.org/wiki/Nondeterministic_finite_automaton
http://en.wikipedia.org/wiki/Powerset_construction

11.2 Esimerkki: Etsi ja korvaa käyttäen säännöllisiä lausekkeita

Useat tekstieditorit tukevat myös säännöllisiä lausekkeita: Esimerkkinä mainittakoon Notepad++ (Windows), TextWrangler (OS X) tai Kate (Linux).

Säännölliset lausekkeet ovat pohjimmiltaan hakuehtoja, jotka muodostuvat erikoismerkityksellisistä merkeistä. Näitä hakuehtoja voidaan käyttää samankaltaisten ilmausten etsimiseen tekstistä. Seuraavassa esimerkissä tutkimme säännöllisten lausekkeiden käyttöä käytännössä.



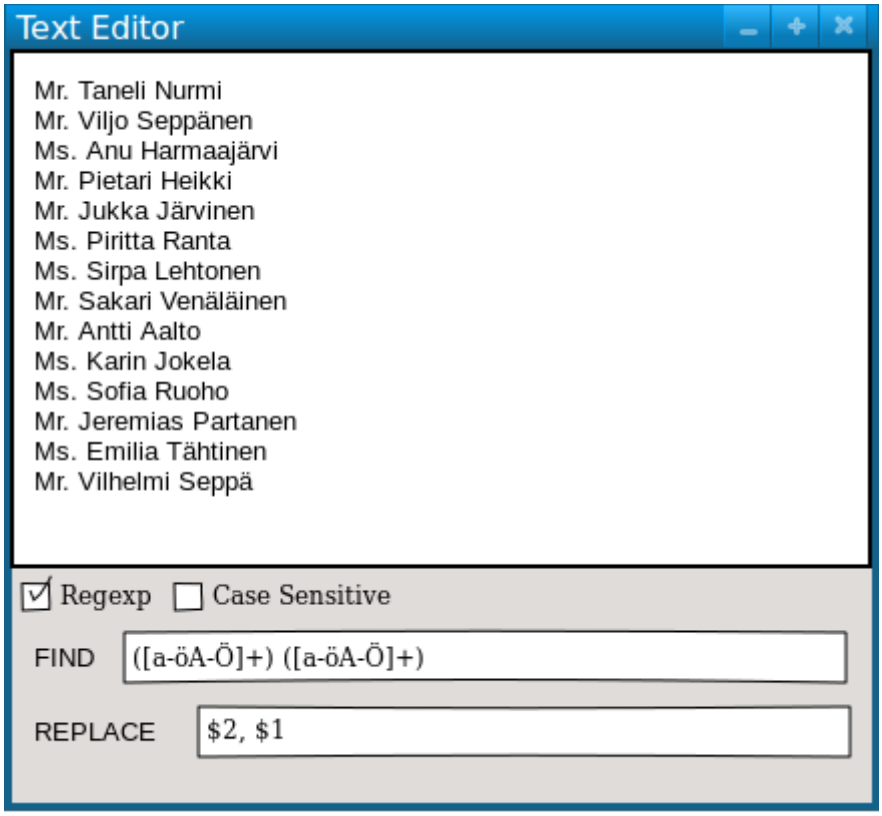
Kuva 6: Säännölliset lausekkeet tekstieditoreissa.

Kuvassa meillä on tekstieditorissa avattu tiedosto, joka sisältää useita ihmisten nimiä. Nimet on voitu saada esimerkiksi suoraan verkkolomakkeesta, käyttäjätietokannasta tai muusta vastaavasta. Nimet eivät kuitenkaan ole samanlaisessa muodossa: Kaikkien miesten nimien edessä on titteli “Mr.”, mutta naisten nimien edestä vastaava “Ms.” puuttuu.

Vaikka nimiä onkin vain rajallinen määrä, voi tällaisten virheiden korjaus yksitellen viedä aikaa. Tällöin voi olla hyödykästä automatisoida virheellisten rivien haku ja korjaus. Käytetään tähän tarkoitukseen tekstieditorin Etsi ja korvaa (Find and Replace) -toimintoa yhdessä säännöllisen lausekkeen kanssa.

Jokainen nimi on omalla rivillään, joten haluamme hakea tekstistä kaikki ne rivit, jotka eivät ala merkkijonolla “Mr”. Rivin aloitus ilmaistaan säännöllisessä lausekkeessa merkillä **^**. Merkkijono “**Mr**” vastaavat suoraan kaikkia tiedostossa olevia “M”:n ja “r”:n peräkkäisiä esiintymisiä. Koska halusimme kuitenkin ne rivit, joita em. ilmaus ei aloita - tähän voimme käyttää negaatio-operaattoria **!**. Operaattorien **?** ja **!** avulla voimme etsiä kaikki rivien aloitukset, jotka eivät ala merkkijonolla “Mr”.

Lauseke, jolla etsimme kaikkien naispuolisten henkilöiden nimet on tässä tapauksessa siis **^(?!Mr)**. Kun tekstieditorin korvauskenttään lisätään oikea titteli “Ms”, pitäisi lopputuloksen näyttää seuraavan kuvan kaltaiselta:



Kuva 7: Säännölliset lausekkeet tekstieditoreissa, osa 2.

Jatketaan esimerkkiä seuraavasti: Mitä jos haluaisimme muuttaa nimilistauksen sellaiseen muotoon, että henkilöiden sukunimi tulee ennen etunimeä? Miten henkilön nimeä tässä listauksessa pitäisi ajatella, kun haluamme kirjoittaa yleispätevän säännön sille tässä tiedostossa?

Henkilön nimi (etu- ja sukunimi) erottuu välilyönnillä. Nimessä on vain kirjaimia A-Ö ja a-ö, eikä yhtään erikoismerkkiä, esimerkiksi titteliä täydentävää pistettä. Kirjoitamme siis seuraavaksi säännöllisen lausekkeen, joka vastaa suomenkielisen aakkoston kirjainten yhdistelmiä ja erottelee etu- ja sukunimen toisistaan välilyönnin perusteella.

Nimen (eli kirjainten yhdistelmän) saamme lauseella **([a-öA-Ö]+)**. **[ ]** -merkit sisältävät säännön aakkostolle, jota sääntömme tulee vastata, tässä tapauksessa pienet aakkoset a:sta ö:hön sekä vastaavasti suuret aakkoset A:sta Ö:hön. **+** -merkillä täsmennämme, että em. aakkoston mukaisia kirjaimia voi olla sanassa yksi tai useampi peräkkäin.

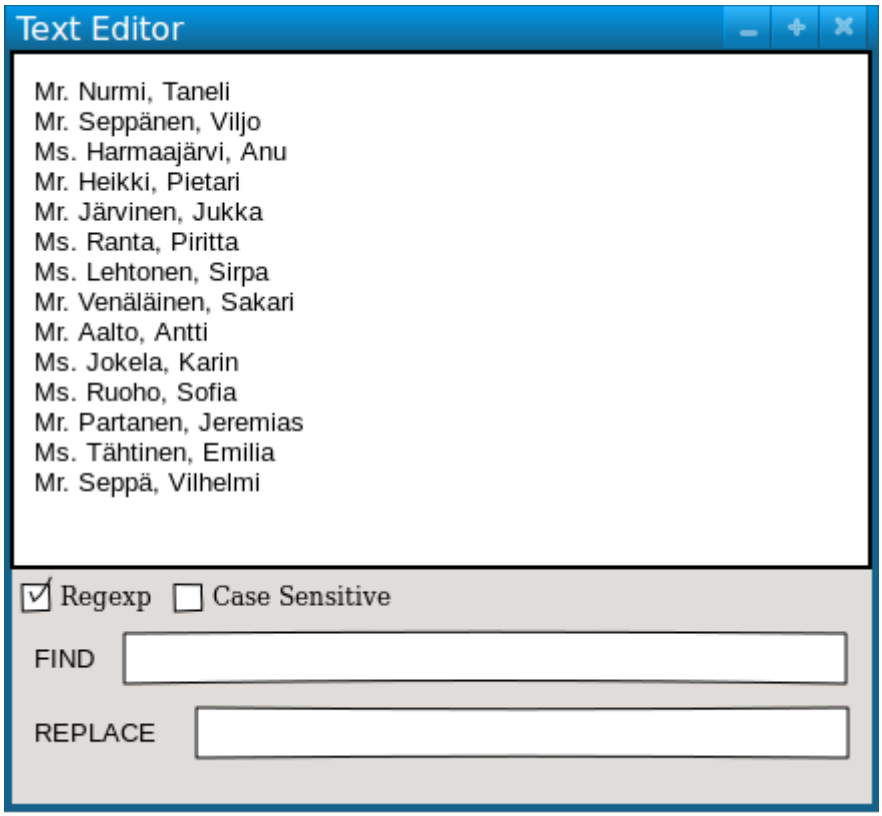
Tämä siis vastaisi kaikkia etunimiä ja kaikkia sukunimiä. Koska tiedämme etu- ja sukunimen erottuvan toisistaan välilyönnillä, voimme käyttää tätä yksittäistä nimeä vastaavaa lauseketta kahdesti peräkkäin välilyönnillä erotettuna:

([a-öA-Ö]+) ([a-öA-Ö]+)

( ) -sulkeet puolestaan antavat mahdollisuuden päästä käsiksi löydettyyn merkkijonoon. Jokaista lauseketta vastaavaa merkkijonoa kohden lausekkeen ensimmäisiä ()-sulkeita vastaava merkkojono tallennetaan muuttujaan $1, toisia sulkeita vastaava merkkijono muuttujaan $2 ja niin edelleen. Kun lisäämme korvauskenttään

$2, $1

pitäisi lopputuloksen näyttää seuraavalta:



Kuva 8: Säännölliset lausekkeet tekstieditoreissa, osa 3.

11.3 Merkkilistaus

Tässä kappaleessa listataan yleisiä säännöllisten lausekkeiden erikoismerkkejä ja sääntöjä. Listaus ei ole täydellinen, mutta kattaa suurimman osan säännöllisten lausekkeiden kieliopista.

. - vastaa jokaista yksittäistä merkkiä paitsi rivinvaihtoa.

[ ] - Vastaa hakasulkujen sisälle kirjoitettua merkkiä, esimerkiksi *[abc]* vastaa kirjaimia “a”, “b” ja “c”.

( ) - Määrittää lausekkeen sisälle lausekkeen.

+ - Vastaa merkkiä edeltävää sääntöä ainakin kerran.

? - Vastaa merkkiä edeltävää sääntöä 0-1 kertaa. (Vähintään 0, enintään 1).

***** - Vastaa merkkiä edeltävää sääntöä 0-n kertaa (eli vähintään nolla ja enintään mielivaltaisen määrän).

{M,N} - Määrittää säännölle minimi- ja maksimivastaavuuksien määrän (vähintään M, enintään N).

[...] - Määrittää säännön hyväksyttävistä merkeistä (esim. mikä tahansa vokaali: **[aeiouyäö]**).

| - Tai-operaattori, erottaa vaihtoehtoiset säännöt toisistaan.

\w - Mikä tahansa alfanumeerinen merkki (Iso tai pieni kirjain tai numero).

\W - Mikä tahansa ei-alfanumeerinen merkki.

\s - Välilyönti- tai muu tyhjä merkki, kuten sisennys, rivin- tai sivunvaihto tai muu muotoilumerkki.

\S - Mikä tahansa ei-tyhjä merkki.

\d - Numero (eli sama kuin **[0-9]**).

\D - Ei-numero.

^ - Rivin alku.

$ - Rivin loppu.

Luku 12: Loppusanat

Tämä opas on nyt lopussansa. Opas on antanut kuvan miten Linux/Unix-komentorivi toimii ja mitä hyötyä siitä on. Vaikka monet tässä oppaassa esitetyt asiat voidaan hoitaa graafisten työkalujen avulla, on komentorivin käyttö useimmiten nopeampaa ja joskus se on myös ainut vaihtoehto, jos esimerkiksi ylläpidetään palvelinta etäyhteyden yli.

Komentorivin käyttö on monipuolista ja, kuten tietotekniikassa niin usein, vain taivas on rajana. Tämä opas ei siis mitenkään kyennyt käsittelemään kaikkia aiheita, vipuja ja komentoja. Internet on onneksi ohjeistusta täynnä, joten kysyvä ei tieltä eksy. Tästä on hyvä jatkaa kohti ylläpitäjyyttä ja asiantuntijuutta.

Happy hacking!

Liite 1: UNIX -muuttujat

Muuttujat ovat tapa välittää tietoa komentoriviltä ohjelmille, kun niitä ajetaan. Ohjelmat etsivät “ympäristöstä” tiettyjä muuttujia, ja löydettäessä käyttävät niihin varastoituja arvoja. Toiset muuttujista ovat järjestelmän asettamia, osa käyttäjän ja osa komentorivin tai minkä tahansa muun ohjelman, joka lataa toista ohjelmaa. UNIXin standardimuuttujat on jaettu kahteen kategoriaan, ympäristö- ja komentorivimuuttujiin. Laajasti ilmaistuna komentorivimuuttujat vaikuttavat vain nykyisessä komentorivin ilmentymässä ja niitä käytetään asettamaan lyhytaikaisia työskentelyolosuhteita; ympäristömuuttujilla on laajemmalle ulottuva merkitys, ja kirjautumisen aikana asetetut muuttujat ovat käyttökelpoisia istunnon koko keston ajan. Sopimuksen mukaan ympäristömuuttujien nimet kirjoitetaan ISOILLA ja komentorivimuuttujien taas pienillä kirjaimilla.

L1.1 Ympäristömuuttujat

Esimerkkinä ympäristömuuttujasta toimii OSTYPE -muuttuja. Muuttujan arvo määräytyy sillä hetkellä käytössä olevan käyttöjärjestelmän mukaan. Kirjoita
$ echo $OSTYPE
Muita esimerkkejä ympäristömuuttujista on esimerkiksi

- USER (käyttäjänimesi)
- HOME (kotihakemistosi polunnimi)
- HOST (käytössä olevan koneen nimi)
- ARCH (koneen prosessorin tiedot)
- DISPLAY (näytä X ikkunaa tietokoneen ruudulla)
- PRINTER (oletustulostin, johon tulostukset lähetetään)
- PATH (hakemistot joista komentorivi etsii komennon löytämiseksi)

Muuttujien nykyisten arvojen selvittäminen

YMPÄRISTÖmuuttujat asetetaan käyttäen **setenv** -komentoa, näytetään käyttäen joko **printenv**- tai **env**- komentoja, ja niiden asetus perutaan käyttäe **unsetev** -komentoa. Näytä muuttujien arvot kirjoittamalla
$ printenv | *less*

L1.2 Komentorivimuuttujat

Esimerkkinä komentorivimuuttujasta toimii history -muuttuja. Sen arvo määrittää, kuinka monta viimeisimmistä komennoista tulee säilyttää eli kuinka monen edellisen komennon läpi käyttäjä voi selata. Kirjoita
$ echo $history
Muita esimerkkejä komentorivimuuttujista:

- cwd (current working directory, nykyinen työskentelyhakemisto)
- home (kotihakemistosi polunnimi)
- path (hakemistot joista komentorivi etsii komennon löytämiseksi)
- prompt (kehote)

Muuttujien nykyisten arvojen selvittäminen

Komentorivimuuttujat sekä asetetaan että näytetään **set**-komentoa käyttäen. Asettamisen voi peruuttaa **unset** -komennolla. Näiden muuttujien kaikki arvot näyttääksesi kirjoita
$ set | less

Mitä eroa sitten on PATH:illa ja path:illa?

Yleisesti ottaen samannimiset (kun kirjainkokoa ei huomioida) ympäristö- ja komentorivimuuttujat ovat erottuvia ja itsenäisiä, mahdollisesti samoja kirjainarvoja lukuunottamatta. On olemassa kuitenkin poikkeuksia.
Joka kerta kun komentorivimuuttujia home, user ja term muutetaan, vastaavat ympäristömuuttujat HOME, USER ja TERM vastaanottavat samat arvot. Kuitenkaan ympäristömuuttujien muuttaminen ei vaikuta samoin vastaaviin komentorivimuuttujiin.

PATH ja path määrittävät hakemistot joista etsiä komentoja ja ohjelmia. Molemmat muuttujat esittävät aina samaa hakemistolistaa, ja toisen muuttaminen aiheuttaa automaattisesti myös toisen muuttumisen.

L1.3 Muuttujien käyttäminen ja asettaminen

Joka kerta kun kirjaudut UNIX -hostille, järjestelmä etsii kotihakemistostasi alustustiedostoja. Näiden tiedostojen sisältämää tietoa käytetään työympäristön asettamiseen. C- ja TC-shellit (komentorivit) käyttävät kahta eri tiedostoa nimeltä *.login* ja *.cshrc. (*Huomaa, että molemmat nimet alkavat pisteellä)
*.login*ia käytetään asettamaan olosuhteet, jotka vaikuttavat koko istuntoon ja suorittamaan toimintoja jotka ovat oleellisia vain kirjautuessa.
*.cshrc*tä käytetään asettamaan olosuhteet ja suorittamaan toiminnot, joita käytetään erityisesti komentorivin hyödyntämiseen.
Suosituksena on asettaa YMPÄRISTÖmuuttujat *.login* -tiedostoon ja komentorivimuuttujat *.cshrc* -tiedostoon.

HUOM: Älä laita GUI-ohjelmia (esim webbiselain) .cshrc tai .login -tiedostoihin!

L1.4 Komentorivimuuttujien asettaminen .cshrc -tiedostoon

Esimerkiksi vaihtaaksesi history -muuttujaan tallentuvien komentojen määrän, sinun täytyy asettaa komentorivimuuttujaan uusi arvo. Oletusarvo on 100, mutta halutessa sitä on mahdollista kasvattaa.
$ set history = 200
Kokeile, toimiko komento kirjoittamalla
$ echo $history
Kuitenkin tämä on asettanut muuttujan vain nykyisen komentorivin elinajaksi. Jos avaat uuden terminaali-ikkunan, sillä muuttujaan on asetettu ainoastaan oletusarvo. Haluttaessa asettaa muuttujaan pysyvä, uusi arvo, **set** -komento tulee lisätä *.cshrc* -tiedostoon.
Aloita avaamalla *.cshrc* -tiedosto tekstieditorissa. Esimerkiksi nedit on helppo, käyttäjäystävällinen editori.
$ nedit ~/.cshrc

Lisää seuraava rivi muiden komentojen perään.

set history = 200

Tallenna tiedosto ja pakota komentorivi lukemaan sen *.cshrc* -tiedosto uudestaan:
$ source .cshrc
Tarkista, toimiko komento kirjoittamalla
$ echo $history

L1.5 Polun asettaminen

Kun kirjoitat komentoa, path (tai PATH) -muuttuja määrittelee hakemistot, joista komentorivi etsii kirjoitettua komentoa. Jos järjestelmä palauttaa viestin "command: Command not found", ei komentoa joko ole olemassa ollenkaan tai se ei ole kyseisessä polussa. Esimerkiksi ajaaksesi ohjelman *units*, sinun täytyy joko määritellä suoraan polku ohjelmalle (~/units174/bin/units), tai hakemiston *~/units174/bin/units* täytyy olla osana polkua. Voit lisätä sen olemassaolevan polun perään ($path kuvaa tätä) antamalla komennon
$ set path = ($path ~/units174/bin)
Kokeile ajaa *units* missä tahansa muussa kansiossa kuin mihin se alunperin oli sijoitettu, ja katso toimiko komento:
$ cd
$ units
Lisätäksesi polun pysyvästi, lisää seuraava rivi *.cshrc* -tiedostoon muiden komentojen perään
set path = ($path ~/units174/bin).

Lähteet

Alkuperäinen opas Michael Stonebank: UNIX Tutorial for Beginners
http://www.ee.surrey.ac.uk/Teaching/Unix/

Linux.fi-wiki
http://www.linux.fi

http://www.freeos.com/guides/lsst/index.html

http://www.linfo.org

http://en.wikipedia.org/wiki/Regular_expression