KOMENTORIVIN PERUSTEET

Published : 2012-06-07
License : GPLv2

JOHDANTO

1. OTA KOMENTO

1. OTA KOMENTO

Komentojen avulla voi tehdä monimutkaisia asioita tietokoneella. Näytämme tämän esittelemällä kaikille tuttuja jokapäiväisiä toimenpiteitä. Jos käytät digikameraa, koneellasi on luultavasti kuvia täynnä oleva kansio. Ajattele, että tahdot muuttaa kuvan "profiili.jpg" koko niin, että se on 300 pikseliä leveä, ja tallentaa sen uuteen tiedostoon, jonka nimi on "profiili_pieni.jpg".

Kuvankäsittelyohjelmalla kävisit luultavasti läpi seuraavat askeleet:

1. Avaa kuvankäsittelyohjelma "Sovellukset"-valikosta.
2. Napsauta "Tiedosto>Avaa" valikosta.
3. Selaa hakemistoon, jossa kuvasi ovat.
4. Napsauta kuvatiedostoa "profiili.jpg" ja sitten "Avaa".
5. Napsauta valikosta "Kuva>Muuta kuvan kokoa" kuvien koon muuttamista varten.
6. Vaihda kuvan leveys 300 pikseliin ja napsauta "Muuta kuvan kokoa".
7. Napsauta valikosta "Tiedosto>Tallenna nimellä" tallentaaksesi tiedoston.
8. Kirjoita uudeksi tiedostonimeksi "profiili_pieni.jpg" ja napsauta "Tallenna".

Komentoriviä käyttämällä voit tehdä saman toimenpiteen kirjoittamalla komennon:

```
convert -resize 300 profiili.jpg profiili_pieni.jpg
```

Komentorivillä tarvitaan siis yksi askel ja graafisella käyttöliittymällä kahdeksan askelta. Ehkäpä ajattelet, että aika, jonka kulutat tämän kirjan lukemiseen ja komentojen opettelemiseen on arvokkaampi kuin seitsemän askeleen välttämisellä säästynyt aika. Mutta entäpä jos 30 kuvaa pitäisi muuttaa uuteen kokoon? Tahtoisitko edelleenkin avata jokaisen kuvan erikseen ja toistaa prosessin 30 kertaa käyttäen kuvankäsittelyohjelmaa? Joutuisit käymään läpi 240 askelta. Etkö kirjoittaisi mielummin yhden komennon ja saisi homman tehtyä? Kaikkien hakemistossa olevien jpg-kuvien nimet voit muuttaa seuraavalla komennolla:

```
convert -resize 300 *.jpg
```

Tässä komennossa "*.jpg" viittaa kaikkiin hakemiston jpg-muotoisiin kuviin. Yksi komento voi tehdä saman asian 30, 300 tai 3000 kuvalle. Tämä on yksi suurimmista syistä aloittaa komentorivin käyttö. Voit oppia aluksi hitaasti, mutta pitkällä tähtäimellä säästät paljon aikaa. Vieläkin tärkeämpää on, että komentorivin oppiminen avaa monia mielenkiintoisia mahdollisuuksia ja hauskoja tapoja työskennellä. Katsotaanpa joitain syitä, joiden vuoksi komentorivin käyttäminen on hyvä idea.

KOMENTOJEN KÄYTÖN YHTEENLASKETUT EDUT

Monet komentoriviä kokeilleet ihmiset ovat hämmästyneet sen mahdollisuuksista, eivätkä tahdo enää palata takaisin graafisen käyttöliittymän pariin. Miksi? Komentorivin käytöllä on seuraavat hyvät puolet yleisiin graafisen käyttöliittymän omaaviin ohjelmiin verrattuna:

- **Joustavuus.** Graafisilla ohjelmilla törmäät joskus rajoituksiin, et voi tehdä mitä tahtoisit, tai joudut keksimään hankalia tapoja kiertää ohjelmien rajoituksia. Komentorivillä voit kuitenkin yhdistellä komentoja ja saavuttaa lähes loputtoman määrän uusia ja mielenkiintoisia toimintoja. Yhdistelemällä komentoja luovasti voit saada komentorivin tekemään täsmälleen mitä tahdot - se asettaa sinut tietokoneen komentoon.
- **Luotettavuus.** Graafiset ohjelmat ovat usein epäkypsiä tai jopa epävakaita. Sen sijaan komentoriviltä käytettävät työkalut ovat hyvin luotettavia. Yksi syy tähän on niiden kypsyys: vanhimmat komentoriviohjelmat ovat olleet olemassa 1970-luvulta asti, joten niitä on testailtu yli kolme vuosikymmentä. Ne toimivat samalla tavalla eri käyttöjärjestelmissä, toisin kuin graafiset työkalut. Jost tahdot monikäyttöisen työkalun johon voit luottaa, komentorivi on sinua varten.
- **Nopeus**. Hienot grafiikat syövät paljon laitteistosi resursseja, mikä johtaa usein hidasteluun tai epävakauteen. Komentorivi käyttää tietokoneen resursseja paljon säästäväisemmin, jättäen muistia ja prosessoritehoa tehtäville, jotka tahdot toteuttaa. Komentorivi itsessään on nopeampi: monien graafisten valikkojen klikkailun sijasta voit kirjoittaa komentoja noin tusinalla näppäimen painalluksella, ja usein käyttää komentoja kerralla useampiin tiedostoihin tai muihin kohteisiin. Jos olet nopea näppäilijä, voit lisätä tuottavuuttasi huomattavasti.
- **Kokemus.** Komentorivin käyttäminen on hieno oppimiskokemus. Kun käytät komentoriviä, kommunikoit tietokoneesi kanssa suoremmin kuin graafisilla ohjelmilla, joten opit paljon enemmän sen toiminnasta: komentorivin säännöllinen käyttö on tapa, jolla GNU/Linux guruksi tullaan.
- **Hauskuus.** Oletko koskaan tahtonut olla kuin ne coolit tietokonehakkerit, jotka saavat GNU/Linux -koneen tekemään asioita, joista et ole edes unelmoinut? Useimmat heistä tekevät sen käyttäen komentoriviä. Kun olet oppinut käyttämään tätä voimakasta työkalua, osaat tehdä hauskoja ja mielenkiintoisia asioita, joita et aikaisemmin edes tiennyt mahdollisiksi.

SKRIPTAUKSEN ARVO

On enemmänkin mahdollisuuksia! Voit myös tallentaa komentoja tekstitiedostoihin. Näitä tekstitiedostoja kutsutaan skripteiksi ja niitä voidaan käyttää pitkien komentoketjujen kirjoittamisen sijasta. Jos esimerkiksi tallennat komennot tiedostoon nimeltä "komentoni.sh", et joudu kirjoittamaan komentoja uudestaan, vaan kirjoitat yksinkertaisesti:

komentoni.sh

Lisäksi voit yhdistää komentoja yksinkertaisilla tai hienostuneilla tavoilla. Voit myös ajastaa skriptit ajettaviksi tiettyinä aikoina tai päivämäärinä tai tiettyjen tapahtumien tapahtuessa tietokoneellasi.

Voit myös kirjoittaa skriptit niin, että ne ottavat sinulta lisätietoa. Esimerkiksi kuvien kokoa muuttama skripti voi kysyä kuvien uutta kokoa ennen kuin alkaa.

Oletko koskaan tehnyt mitään tämän kaltaista käyttäen graafista käyttöliittymää? Ehkä nyt voit nähdä, kuinka komentorivillä työskentely avaa kokonaisen uuden maailman tietokoneesi käytössä.

ONKO TIETOKONEENI SAIRAANA?

Toinen käyttö komentoriville on tietokoneesi terveydentilan tarkastaminen. Voit käyttää monia komentoja tarkastaaksesi tietokoneesi terveydentilan, kovalevyllä jäljellä olevasta tilasta aina keskusprosessorin lämpötilaan asti. Jos tietokoneesi toimii huonosti ja et tiedä syytä, muutaman komennon kirjoittaminen voi auttaa sinua päättelemään nopeasti, onko vika laitteissa vai ohjelmissa, ja korjaamaan vian nopeasti.

KÄYTTÖ VERKON YLI

Toinen mielenkiintoinen komentorivin etu, johon graafiset käyttöliittymät eivät kykene, on käyttö verkon yli. Ajattele, että tietokoneesi on toisessa huoneessa, ja tahdot kääntää sen pois päältä. Kuinka se tehdään? Helppoa? Kävele tietokoneelle ja napsauta "sulje"-nappia.

Komentorivillä voi ottaa yhteyttä naapurihuoneen tietokoneeseen ja kirjoittaa komennon halt.

Tämä voi näyttää tyhjänpäiväiseltä. Ehkä on parempi nousta tuolista ja kuluttaa viisi kaloria kävelemällä naapurihuoneeseen. Entäpä jos suljettava tietokone onkin toisessa kaupunginosassa? Toisessa kaupungissa? Toisessa maassa? Tietokoneen kauko-ohjaus voi olla hyvin hyödyllistä.

Koneen sulkeminen kauko-ohjauksella on vasta alkua. Kaiken komentorivillä tehtävän voi tehdä kauko-ohjauksella. Tämä merkitsee, että voit ajaa skriptejä, suorittaa komentoja, muokata tekstitiedostoja, tarkastaa koneen tilan jne. Komentorivin maailma on paljon suurempi.

JOPA GRAAFISET OHJELMAT OVAT KOMENTOJA

Kun napsautat kuvaketta tai valikkoa aloittaaksesi ohjelman, ajat komennon. Voit huomata, että on hyödyllistä ymmärtää mitä komentoja ajat. Esimerkiksi jos oletat ohjelman toimivan näkymättömissä taustalla ja hidastavan konettasi, voit etsiä sen komennon ja pysäyttää ohjelman. Graafiset ohjelmat lähettävät usein enemmän virheviestejä komentoriville kuin näyttävät virheikkunoissa, ja tästä on hyötyä ongelmia tutkittaessa.

PERUSTEET

2. KÄYTÖN ALOITTAMINEN
3. KOMENNON OSAT
4. YMPÄRIINSÄ SIIRTYMINEN
5. TIEDOSTOT JA HAKEMISTOT

2. KÄYTÖN ALOITTAMINEN

Nykyaikainen tietojenkäsittely on hyvin interaktiivista ja komentorivin käyttö on vain yksi interaktiivisuuden muoto. Useimmat ihmiset käyttävät tietokonetta sen graafisen käyttöliittymän avulla, interaktiivisuus tapahtuu nopeaan tahtiin. Käyttäjä napsauttaa kohdetta, vetää ja pudottaa sen, kaksoisnapsauttaa toista avatakseen sen, muuttaakseen sitä jne.

Vaikka vuorovaikutteisuus tapahtuu niin nopeasti, ettet ajattele sitä, jokainen napsautus tai näppäimen painallus on komento tietokoneelle, johon se reagoi. Komentorivin käyttö on sama asia, mutta tarkoituksellisempi. Kirjoitat komennon ja painat rivinvaihtonäppäintä. Esimerkiksi kirjoitan terminaaliin:

```
date
```

Ja tietokoneen vastaus on:

```
ke 2.9.2009 14.42.00 +0300
```

Tämä on aika tietokonemaista. Myöhemmissä luvuissa selitämme, miten voit kysyä päivämäärää ja aikaa mukavammassa muodossa. Kerromme myös, miten eri maissa ja eri kielillä työskentely muuttaa tietokoneen ulostuloa. Kuvaamamme tapahtuma tapahtui kuitenkin vuorovaikutuksessa tietokoneen kanssa.

KOMENTORIVI VOI TOIMIA PALJON PAREMMIN

date-komento, kuten yllä on nähty, toimii huonosti kalenterin tai kellon katsomiseen verrattuna. Tärkein ongelma ei ole ulostulon epämiellyttävä ulkoasu, joka on jo mainittu, vaan hankaluus käyttää ulostulon arvoa. Esimerkiksi katsoessani verkkokalenteristani päivämäärää lisätäkseni sen dokumenttiin, jota kirjoitan tai päivitän, joudun kirjoittamaan sen osittain uudelleen. Komentorivi voi toimia paljon paremmin.

Kun olet oppinut joitain peruskomentoja ja siistejä tapoja säästää aikaasi, tässä kirjassa kuvaillaan komentojen ulostulon syöttämistä muille komennoille, toimintojen automatisointia, ja komentojen tallentamista myöhempää käyttöä varten.

MITÄ TARKOITAMME KOMENNOLLA?

Tämän luvun alussa käytimme sanaa "komento" hyvin yleisesti viittaamaan mihin tahansa tapaan, jolla kerrotaan tietokoneelle mitä sen pitäisi tehdä. Tässä kirjassa komennolla on kuitenkin hyvin tarkka merkitys. Se on tietokoneellasi oleva tiedosto, joka voidaan suorittaa. Muutamia suoraan suoritettavia komentoja (sisäänrakennetut komennot) lukuunottamatta jokainen komento ajetaan etsimällä tiedosto, jolla on komennon nimi, ja ajamalla kyseinen tiedosto. Kerromme lisää yksityiskohtia kun ne ovat tarpeen.

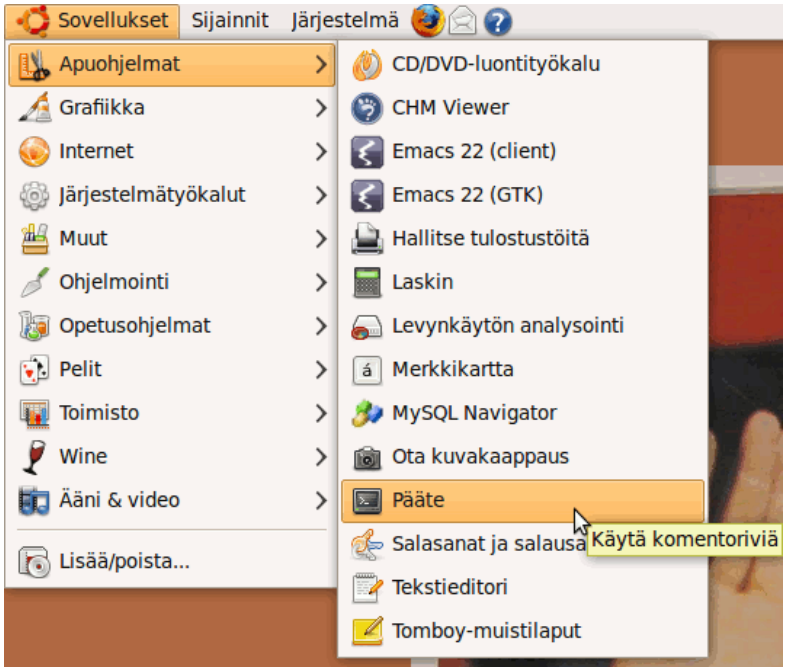
TAPOJA SYÖTTÄÄ KOMENTOJA

Seurataksesi tätä kirjaa joudut avaamaan tietokoneellasi komentorivin (jota kutsutaan GNU/Linuxissa myös komentoliittymäksi, päätteeksi, shelliksi tai terminaaliksi). Grafiikkaa edeltävät tietokonenäytöt näyttivät tämän komentotulkin ensimmäisenä sisäänkirjautumisen jälkeen. Nyt lähes kaikki paitsi ammattimaiset tietokoneiden ylläpitäjät käyttävät graafista käyttöliittymää. Niinpä näytämme sinulle, kuinka voit avata komentorivin.

PÄÄTTEEN LÖYTÄMINEN

Käytännössä jokainen tietokoneen käyttöliittymä tarjoaa ohjelman, joka matkii vanhanaikaisia vain tekstiin perustuvia päätteitä, joita tietokoneet tarjosivat ennen käyttöliittymiksi. Katso työpöytäsi valikkojen läpi, jos löydät päätteeksi (englanniksi usein "terminal" tai "shell") kutsutun ohjelman. Se on usein valikossa, jota kutsutaan "Apuohjelmiksi", mikä ei ole reilua, koska luettuasi tämän kirjan tulet käyttämään päätettä enemmän ja enemmän joka päivä.

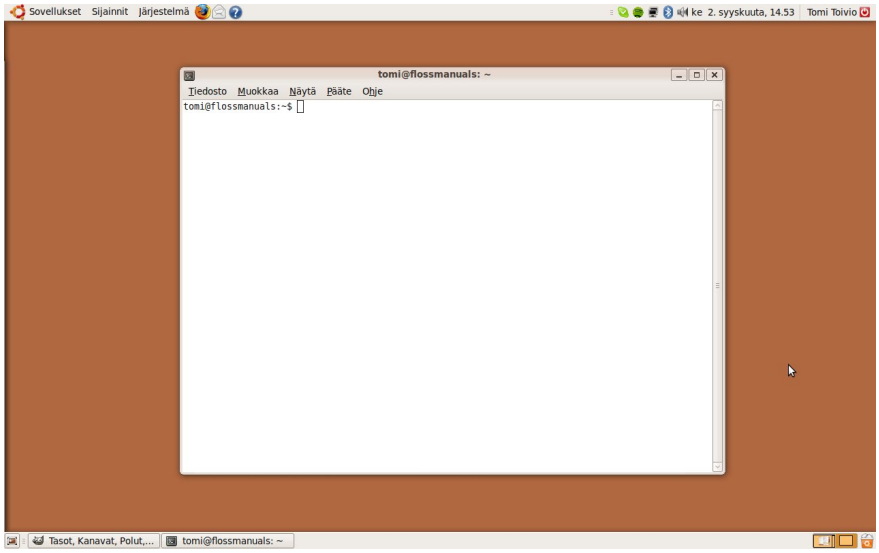
Esimerkiksi Ubuntussa voit valita Sovellukset -> **Apuohjelmat** -> **Pääte**



Missä tahansa se onkin, voit melkein varmasti löytää Linuxistasi pääteohjelman.

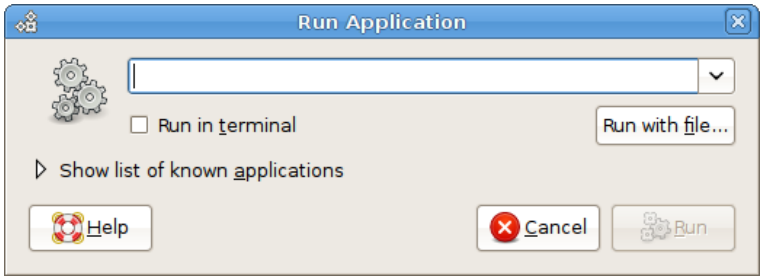
Kun suoritat pääteohjelman, se näyttää vain tyhjän ikkunan, ohjeita ei juurikaan ole näkyvissä. Oletetaan, että tiedät mitä pitää tehdä - sen näytämme sinulle.

Seuraava kuva näyttää pääteikkunan avattuna Linuxin GNOME-työpöydällä.



YKSITTÄISEN KOMENNON SUORITTAMINEN

Monissa graafisissa käyttöliittymissä on myös pieni ikkuna, jota kutsutaan "komennon suorittamiseksi". Siinä on tekstialue, johon voi kirjoittaa komennon, joka suoritetaan syöttöpainiketta ("enter") painamalla.



Tätä laatikkoa voidaan käyttää oikotienä pääteohjelman nopeaan käynnistämiseen, kunhan tiedät tietokoneellesi asennetun pääteohjelman nimen. Jos työskentelet tuntemattomalla käyttöjärjestelmällä etkä tiedä edes oletuspääteohjelman nimeä, yritä kirjoittaa "xterm" avataksesi tyypillisen pääteohjelman (ei hienoja valikkoja, jotka mahdollistavat värien tai kirjasintyyppien valinnan).

KUINKA NÄYTÄMME KOMENNOT JA ULOSTULON TÄSSÄ KIRJASSA

Komentoriviä käsittelevissä kirjoissa on yleinen käytäntö. Kun terminaali avataan, näet pienen merkkijonon, joka osoittaa, että pääte on valmis ottamaan sinulta komentoja. Tätä merkkijonoa kutsutaan kehotteeksi (englanniksi "prompt"), ja se voi olla näin yksinkertainen:

```
$
```

Kun kirjoitat komentosi ja painat rivinvaihtoa, pääte näyttää komennon ulostulon (jos komennolla on ulostulo), sekä uuden kehotteen. Niinpä edellinen vuorovaikutukseni näytettäisiin kirjassa näin:

```
$ date
ke 2.9.2009 14.57.35 +0300
$
```

Sinun täytyy ymmärtää, kuinka edeltävän kaltaisia esimerkkejä tulkitaan. Tässä esimerkissä kirjoitat vain komennon "date". Ja sen jälkeen painat syöttönäppäintä. Sana "date" on lihavoitu, millä osoitetaan, että se on jotain, minkä sinä kirjoitat. Loput on päätteen ulostuloa.

3. KOMENNON OSAT

Ensimmäinen sana, jonka kirjoitat riville, on komento, jonka tahdot suorittaa. Aloitusluvussa näimme komennon "date", joka palautti päivämäärän ja ajan.

ARGUMENTIT

Toinen komento jota voisimme käyttää, on "echo", joka näyttää määritellyn tiedon käyttäjälle. Se ei ole kovinkaan hyödyllinen, jos emme määrittele näytettävää tietoa. Onneksi voimme lisätä komentoon tietoa muuttaaksemme sen käytöstä. Tämä tieto koostuu argumenteista. Onneksi "echo"-komento ei argumentoi vastaan, se vain toistaa mitä käskemme sen toistaa:

```
$ echo hei hei
```

Tässä tapauksessa argumentti oli "hei", mutta ei ole mitään syytä rajoittaa argumenttien määrää yhteen. Jokainen syötetty sana tekstiä, sisältäen ensimmäisen sanan, on uusi argumentti, joka syötetään komennolle. Jos tahdomme echon vastaavan useammilla sanoilla, kuten "hei maailma", voimme antaa sille monta argumenttia:

```
$ echo hei maailma

hei maailma
```

Argumentit erotetaan normaalisti "tyhjällä" (välilyönneillä ja tabulaattoreilla - jotka näkyvät valkoisena paperilla). Ei ole väliä kuinka monta välilyöntiä kirjoitat, kunhan niitä on ainakin yksi. Jos esimerkiksi kirjoitat:

```
$ echo hei maailma

hei maailma
```

Monilla välilyönneillä kahden argumentin välillä, "ylimääräiset" välilyönnit jätetään huomiotta, ja ulostulo näyttää argumentit yhden välilyönnin erottamana.

VALITSIMET

Palataksemme komentoon "date", oletetaan että tahdotkin nähdä UTC-ajan ja päivämäärän. Tätä varten "date"-komennossa on "--utc"-optio. Huomaa kaksi ensimmäistä viivaa. Nämä osoittavat argumentin, jonka ohjelma tarkastaa, kun se alkaa kontrolloida käytöstään. "Date"-komento tarkastaan erityisesti "--utc"-option ja sanoo: OK, tiedän että kysyt juurikin UTC-aikaa. Tämä on erona argumentteihin jotka keksimme itse, esimerkiksi antaessamme "echo"-komennolle argumentit "foo bar".

Sanaa edeltää kaksi viivaa, muuten "--utc" kirjoitetaan kuin mikä tahansa argumentti:

```
$ date --utc

ke 2.9.2009 12.10.27 +0000
```

Yleensä voit lyhentää nämä valitsimet lyhempään arvoon, kuten "date -u". Lyhemmässä versiossa on vain yksi viiva. Lyhemmät versiot on nopeampi kirjoittaa (käytä päätteen komennoissa), kun taasen pitkät valitsimet on helpompi lukea (käytä skripteissä).

Ajatellaanpa, että tahdomme katsoa eilisen päivämäärää tämän päivämäärän sijasta. Tätä varten kirjoittaisimme argumentin "--date", joka ottaa oman argumenttinsa. Valitsimen argumentti on yksinkertaisesti valitsinta seuraava sana. Tässä tapauksessa komento olisi "--date yesterday".

Koska optiot ovat vain argumentteja, voit yhdistää optioita luodaksesi kehittyneempää käytöstä. Voimme esimerkiksi yhdistää kaksi edellistä valitsinta saadaksemme UTC-tiedot eiliseltä:

```
$ date --date yesterday -u

ti 1.9.2009 12.13.09 +0000
```

KOMENTOJEN TOISTAMINEN

Paina nuolta ylöspäin saadaksesi takaisin komennon, jonka annoit aiemmin. Voit liikkua ylös ja alas käyttäen nuolinäppäimiä, jotta saat uudempia ja vanhempia komentoja. Oikealle ja vasemmalle osoittavia nuolia voit käyttää liikkuaksesi komennon sisällä. Yhdistettynä poistonäppäimeen voit muuttaa komennon osia. Painaessasi syöttönäppäintä lähetät komennon päätteeseen ja se toimii.

4. YMPÄRIINSÄ SIIRTYMINEN

Kuka tahansa, joka on käyttänyt graafista käyttöliittymää, on siirtynyt kansioiden välillä. Tyypillinen kuva kansioista näkyy kuvassa 1, jossa joku on avannut kotihakemiston, kotihakemiston alla olevan hakemiston, jota kutsutaan käyttäjän käyttäjätunnuksella ja sen alla olevan "Asiakirjat"-kansion.

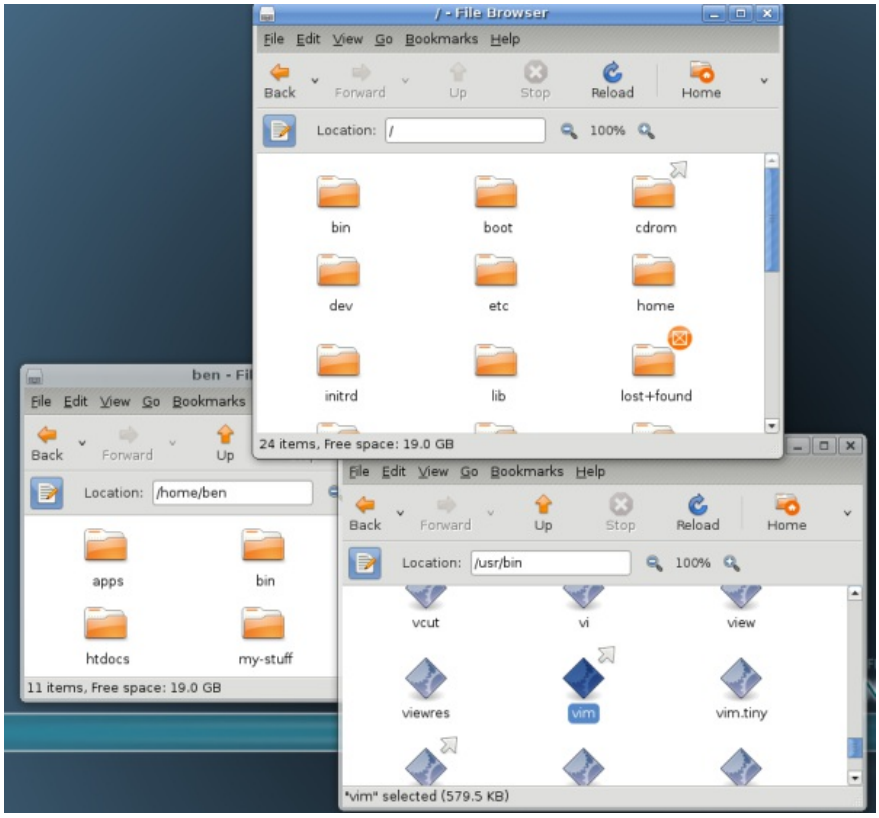
Kuva 1: Kansiot

Kun käytät komentoriviä, kansioita kutsutaan hakemistoiksi. Tämä on vain vanhempi käsite, jota käytetään tietojenkäsittelyssä yleisesti viittaamaan asioiden kokoelmiin. Mitä tahansa teetkin työpöydällä olevassa kansiossa näkyy hakemistossa, kun olet komentorivillä, ja toisin päin. Työpöytä ja komentorivi ovat vain kaksi tapaa katsoa samaa hakemistoa/kansiota, joka on "/home/käyttäjänimesi/Työpöytä/" (englanniksi "/home/username/Desktop/", jossa "username" on käyttäjänimesi, eli jos olet "joe", se on "/home/joe/Desktop/").

Tiedostot sisältävät tietosi - olipa kyse tekstistä, kuvista, musiikista, taulukkolaskentatiedosta, tai jostain muusta - kun hakemistot ovat tiedostojen säilytyspaikkoja. Hakemistot voivat myös sisältää toisia hakemistoja. Tunnet olosi paljon mukavammaksi komentorivillä, kun voit siirtyä hakemistojen välillä, katsoa niitä, luoda ja poistaa niitä, ja niin edelleen.

Hakemistot on järjestetty tiedostojärjestelmiksi. Kovalevylläsi on yhdenlainen tiedostojärjestelmä, CD-ROMilla tai DVD:llä toisenlainen, USB-muistilaitteella omanlaisensa, ja niin edelleen. Tämän vuoksi CD-ROM, DVD tai USB näkyy erillisenä asiana tietokoneellasi, kun laitat sen sisään. Onneksi näistä eroista ei tarvitse murehtia, koska sekä työpöytä että terminaali kätkevät erot. Jossain vaiheessa tässä kirjassa puhumme kuitenkin tiedosta, jota tiedostojärjestelmässä on tiedostoistasi.

Ensimmäistä hakemistoa kutsutaan juureksi (englanniksi "root") ja sen nimi on vain "/" (vinoviiva). Voit ajatella kaikkia hakemistoja ja tiedostoja järjestelmässä puuna, joka kasvaa ylösalaisin tästä juuresta:



Kuva 2: Juurihakemisto

ABSOLUUTTISET JA SUHTEELLISET POLUT

Jokaisella tiedostolla ja hakemistolla järjestelmässä on "osoite", jota kutsutaan sen absoluuttiseksi poluksi tai joskus pelkäksi poluksi. Se kuvailee reittiä, jota pitkin juuresta pääsee tiettyyn tiedostoon tai hakemistoon.

Oletetaan esimerkiksi, että pidät vim-editorista, jonka esittelemme myöhemmässä luvussa, ja sinulle kerrotaan, että voit avata sen komennolla "/usr/bin/vim". Tämä alleviivaa asiaa, jonka mainitsimme aiemmassa luvussa: komennot ovat vain ajettavia tiedostoja. Joten vim-editori on tiedosto, jonka polku on "/usr/bin/vim", ja komennon "/usr/bin/vim" ajaminen suorittaa editorin. Kuten näet näistä esimerkeistä, vinoviiva "/" on myös käytössä eri hakemistojen erotusmerkkinä.

Voitko löytää tiedoston "/usr/bin/vim" kuvassa 2? Polku voidaan ymmärtää seuraavasti:

1. Aloita juurihakemistosta ("/").
2. Siirry alas juurihakemistosta ("/") hakemistoon, jota kutsutaan nimellä "usr".
3. Siirry hakemistosta "usr" alas hakemistoon, jota kutsutaan nimellä "bin".
4. "vim" on siinä hakemistossa.

Olet vasta tottumassa komentorivin käyttöön ja voi tuntua oudolle kirjoittaa komentoja, kun luet tätä kirjaa. Jos tämä luku hämmentää sinua, yritä piirtää kuvan 2 hakemistopuu paperille. Piirrä nuolia paperille, kun ajat komentoja tässä luvussa, jotta pystyt suunnistamaan hakemistoissa.

Huomaa: et voi nähdä onko joku kohde tiedosto vai hakemisto pelkästään katsomalla sen polkua.

Kun työskentelet komentorivillä, työskentelet aina hakemiston "sisällä". Voit löytää tämän hakemiston polun komennolla "pwd" ("print working directory", tulosta työhakemisto):

```
$ pwd

/home/ben
```

Voit nähdä, että "pwd" tulostaa absoluuttisen polun. Jos tahdot vaihtaa työhakemistoasi, voit käyttää komentoa "cd" ("change directory", vaihda hakemistoa), jota seuraa kohdehakemistoon osoittava argumentti:

```
$ cd /
```

Vaihdoit juuri työhakemistosi tiedostojärjestelmän juureen! Jos tahdot palata edelliseen hakemistoon, kirjoita komento:

```
$ cd /home/ben
```

Vaihtoehtoisesti voit "kulkea" takaisin hakemistoon "/home/ben" käyttäen suhteellisia polkuja. Niitä kutsutaan suhteellisiksi, koska ne on määritelty "suhteessa" nykyiseen työhakemistoosi. Jos menet takaisin juurihakemistoon, voit kirjoittaa seuraavat komennot:

```
$ cd /
$ cd home
$ cd ben
$ pwd
/home/ben
```

Ensimmäinen komento muuttaa nykyisen työhakemistosi juureksi. Seuraava komento siirtää sinut hakemistoon "home", jolloin nykyinen työhakemistosi on "/home". Toinen komento muuttaa hakemistoksi "ben", joka on suhteessa hakemistoon "/home", jolloin päädyt hakemistoon "/home/ben".

On hyvä olla taas kotona

Jokainen järjestelmän käyttäjä omistaa hänelle annetun hakemiston, jota kutsutaan kotihakemistoksi. Ei ole väliä mikä nykyinen työhakemistosi on, voit palata nopeasti kotihakemistoosi näin:

```
$ cd
```

Tämä merkitsee, että kirjoita komento "cd" ilman argumentteja.

Kaikki tiedostosi ja asetuksesi on tallennettu kotihakemistoosi (tai sen alihakemistoihin). Jokainen järjestelmäsi käyttäjä, jolla on käyttäjätunnus, saa oman kotihakemiston. Kotihakemistot nimetään käyttäjien käyttäjätunnusten mukaan, ja ne löytyvät yleensä hakemistosta "/home", vaikka joissain järjestelmissä ne sijaitsevat hakemistossa "/usr/home". Kun avaat päätteesi, se sijoittaa sinut kotihakemistoosi.

On olemassa erityinen oikotie, joka viittaa kotihakemistoosi, nimittäin symboli "~" (jota kutsutaan yleensä tildeksi, ja joka löytyy useimpien näppäimistöjen oikealta puolelta, rivinvaihtonäppäimen vierestä). Esimerkiksi "~/Työpöytä" viittaa hakemistoon, jota kutsutaan bunekkö "Työpöytä", ja joka on yleensä olemassa kotihakemistosi sisällä.

"." ja ".." -hakemistot

Merkit "." ja ".." ovat erikoisia ja ne ovat olemassa joka hakemistossa. Yksittäinen piste on lyhenne "tästä hakemistosta." Voit käyttää sitä suhteellisena polkuna ja voit kokeilla ja nähdä mitä tapahtuu, kun teet näin:

```
$ pwd
/usr/bin
$ cd .
$ pwd
/usr/bin
```

Jos "vim" on hakemistossa "/usr/bin", voit ajaa sen kirjoittamalla suhteellisen polun:

```
$ ./vim
```

Toinen erikoishakemisto, "..", on lyhenne "tämän hakemiston ylähakemistosta." Edellisestä esimerkistä jatkaen voit tehdä näin:

```
$ cd ..
$ pwd
/usr
```

Koska ne ovat tiedostojärjestelmän todellisia osia, voit käyttää niitä monimutkaisempien polkujen osana, kuten:

```
$ cd /usr/bin
$ pwd
/usr/bin
$ cd ../lib
$ pwd
/usr/lib
$ cd ../..
$ pwd
/
$ cd home
$ pwd
/home
$ cd ../usr/bin
$ pwd
/usr/bin
```

Yritä liikkua ympäriinsä tietokoneesi sisällä käyttäen komentoriviä ja totut siihen pian!

5. TIEDOSTOT JA HAKEMISTOT

Vaikka olet kiinnostunein tiedostoista omassa kansiossasi tai hakemistossasi, on hyvä tietää mitä muuta järjestelmääsi sisältyy. Tässä luvussa katsomme GNU/Linuxin sisään.

Tässä on lista eräistä yleisistä hakemistoista suoraan juurihakemiston alla (juurihakemiston nimi on vain "/"):

/bin	Perusohjelmat. Ohjelmat, jotka ovat täysin välttämättömiä. Näitä ovat vain komentotulkki ja komennot.
/boot	Käynnistystiedostot. Näitä tarvitaan koneesi käynnistämiseen.
/dev	Laitetiedostot. Kuvailevat fyysisiä asioita, kuten kovalevyjä ja osioita.
/etc	Konfiguraatiotiedostot. Erilaiset asetukset.
/home	Käyttäjien kotihakemistot.
/lib	Peruskirjastot. Perusohjelmat tarvitsevat näitä.
/media	Käyttöönottopisteet siirrettäville medioille.
/mnt	Liitospisteet. Järjestelmänvalvojille, joiden täytyy tilapäisesti ottaa käyttöön tiedostojärjestelmä.
/opt	Kolmannen osapuolen ohjelmat.
/proc	Prosessien tiedostojärjestelmä. Kuvailee prosesseja ja tilatietoa, ei tallenneta levylle.
/root	Järjestelmänvalvojan tiedostot.
/sbin	Järjestelmänvalvojan perusohjelmat. Kuten /bin, mutta vain järjestelmänvalvojan käytettävissä.
/srv	Tiettyyn palveluun liittyvät tiedostot.
/sys	Järjestelmän tiedostojärjestelmä. Kuin proc. Tiedostot muistiin perustuvassa tiedostojärjestelmässä ovat usein hakemistossa tempfs.
/tmp	Tilapäiset tiedostot. Tiedostot, joita ei säilytetä käynnistysten välillä, ovat usein hakemistossa tempfs.
/usr	Useimmat ohjelmat. Hakemistot bin, etc, lib ja sbin uudestaan, mutta vähemmän tärkeille tiedostoille.
/var	Muuttuva data. Kuten tmp, mutta säilytetään uudelleenkäynnistysten välillä

Suurimman osan aikaa sinun ei tarvitse tietää hakemistojärjestelmästä oman kotihakemistosi ulkopuolella, mutta joskus tämä tieto on tarpeen. Ehkä yleisimpiä käyttötapoja on, kun tahdot muuttaa järjestelmän konfiguraatiotiedostoa tai katsoa lokien viestejä, jotka tallentavat järjestelmän tapahtumia ja voivat paljastaa järjestelmässäsi olevien ongelmien syyt. (Lokiviestit löytyvät yleensä hakemistosta "/var/log".)

Perinteisesti GNU/Linuxin konfiguraatio tehtiin editoimalla tekstitiedostoja. Tänään useimmat suositut GNU/Linux -järjestelmät kannustavat käyttäjiä tekemään järjestelmän konfiguraation graafisten hallintatyökalujen avulla. Joskus tämä ei kuitenkaan ole mahdollista tai haluttavaa, joten voit päätyä editoimaan konfiguraatiotiedostoja tekstinkäsittelyohjelmalla. Tämä on yleensä vaikeampaa, koska sinun täytyy tietää, missä nämä tiedostot ovat ja miten niitä voi editoida, ja joskus joudut myös lähettämään ajossa olevalle ohjelmalle viestejä tai uudelleenkäynnistämään sen, jotta se lukee muutoksesi. Tällä menetelmällä on kuitenkin hyvät puolensa, kuten kyky konfiguroida tietokoneita, joissa ei ole grafiikkaa, tai konfiguroida ohjelmia, joille ei ole graafista konfiguraatio-ohjelmaa.

KOMENNOT

6. PERUSKOMENNOT
7. PÄÄKÄYTTÄJÄ (ROOT)
8. USEAMMAT TIEDOSTOT
9. KAIKKI TÄMÄ KIRJOITTAMINEN...

6. PERUSKOMENNOT

Nyt sinulla on perustiedot hakemistoista ja tiedostoista ja osaat toimia vuorovaikutuksessa komentorivikäyttöliittymän kanssa. Opimme nyt joitain komennoista, joita tulet käyttämään monta kertaa päivässä.

LS

Sinun täytyy tietää mitä tiedostoja on jo olemassa, ennen kuin teet muutoksia tiedostoihin. Graafisessa käyttöliittymässä teet tämän avaamalla kansion ja tarkastamalla sen sisällön. Komentoriviltä käytät ohjelmaa "ls" listataksesi kansion sisällön.

```
$ ls
Asiakirjat        Julkinen  Mallit    Työpöytä
examples.desktop  Kuvat     Musiikki  Video
```

Oletusarvoisesti ls käyttää hyvin tiivistä ulostulomuotoa. Monet päätteet näyttävät tiedostot ja alihakemistot eri väreissä, jotka esittävät eri tiedostotyyppejä. Tavallisten tiedostojen nimet eivät näy erikoisväreissä. Jotkin tiedostotyypit, kuten JPEG tai PNG -kuvat, tai tar ja ZIP -tiedostot, ovat yleensä eri värisiä, ja sama pätee myös suoritettaviin ohjelmiin ja hakemistoihin. Yritä komentoa ls itse ja vertaile kuvakkeita, joita graafinen käyttöliittymäsi käyttää väreihin, joita komentorivillä näkyy. Jos ulostuloa ei ole väritetty, voit värittää ls:n ulostulon valitsimella "--color":

```
$ ls --color
```

MAN JA APROPOS

Voit oppia valitsimista ja argumenteista käyttämällä toista ohjelmaa, jonka nimi on "man" ("man" on lyhenne manuaalista):

```
$ man ls
```

Tässä komentoa "man" pyydetään avaamaan käyttöoppaan sivu, joka käsittelee komentoa "ls". Voit käyttää nuolinäppäimiä vierittääksesi käyttöopasta ylös ja alas, ja voit sulkea sen "q"-näppäimellä ("quit", lopeta).

Voit myös käyttää komentoa "man" saadaksesi tietoa uusista ohjelmista. Sanotaanpa, että tahdot nimetä tiedostoja uudestaan käyttäen komentoriviä, mutta et tiedä siihen käytetyn ohjelman nimeä. Voit käyttää ohjelmaa "apropos" sen etsimiseen:

```
$ apropos rename
...
mv (1) - move (rename) files
prename (1) - renames multiple files
rename (2) - change the name or location of a file
...
```

Tässä "apropos" etsii käyttöoppaan sivuja, joista "man"-komento tietää, ja tulostaa komennot, joiden arvelee liittyvän hakemaasi asiaan. Tietokoneellasi tämä komento voi näyttää enemmän tietoja, mutta niihin sisältyy luultavasti yllä näytetyt komennot.

Huomaa, kuinka ohjelmien nimien vieressä on numero. Tämä numero merkitsee niiden osiota. Useimmat ohjelmat, joita voit käyttää komentoriviltä, ovat osassa 1. Voit laittaa apropos-komennon näyttämään tuloksia vain osasta 1 käyttämällä valitsinta "-s 1":

```
$ apropos -s 1 rename
...
mv (1) - move (rename) files
prename (1) - renames multiple files
...
```

Tässä vaiheessa osan numero ei ole kovinkaan tärkeä. Täytyy vain tietää, että osan 1 käyttöohjeen sivut ovat ne, jotka soveltuvat ohjelmiin, joita voit käyttää komentoriviltä. Nähdäksesi listan muista osista, katso "man"-komennon käyttöohjetta käyttäen komentoa "man man".

mv

Kun katsomme komennon "apropos" tuloksia, "mv"-ohjelma näyttää mielenkiintoiselle. Voit käyttää sitä näin:

```
$ mv vanhanimi uusinimi
```

Aivan kuin komennon "apropos" tarjoama kuvaus sanoo, tämä ohjelma siirtää tiedostoja. Jos viimeinen argumenttisi on olemassaoleva tiedosto, "mv" siirtää tiedoston siihen hakemistoon sen sijaan, että nimeäisi sen uudestaan. Tämän vuoksi voit laittaa "mv":hen enemmän kuin kaksi argumenttia:

```
$ mv yksi_tiedosto toinen_tiedosto kolmas_tiedosto ~/kamaa
```

Jos hakemisto "~/kamaa" on olemassa, "mv" siirtää tiedostot sinne. Jos sitä ei ole olemassa, se tuottaa virheviestin, kuten tässä:

```
$ mv yksi_tiedosto toinen_tiedosto kolmas_tiedosto kamaa mv:
kohde ”kamaa” ei ole hakemisto
```

MKDIR

Kuinka hakemiston voi luoda? Käytä "mkdir" -komentoa:

```
$ mkdir ~/kamaa
```

Ja kuinka sen voi poistaa? Käytä "rmdir" -komentoa:

```
$ rmdir ~/kamaa
```

Jos hakemisto, jonka tahdot poistaa, ei ole tyhjä, "rmdir" tuottaa virheviestin eikä poista sitä. Jos tahdot poistaa hakemiston, jossa on tiedostoja, sinun täytyy ensin tyhjentää se. Meidän täytyy luoda hakemisto ja laittaa siihen tiedostoja, jotta voimme nähdä miten poisto tapahtuu. Nämä tiedostot voimme poistaa myöhemmin turvallisesti. Aloitetaanpa luomalla kotihakemistoosi hakemisto, jota kutsutaan nimellä "harjoittelu", ja siirrytään sinne:

```
$ mkdir ~/harjoittelu
$ cd ~/harjoittelu
```

CP, RM JA RMDIR

Kopioidaanpa nyt joitain tiedostoja sinne käyttäen ohjelmaa "cp". Käytämme joitain tiedostoja, jotka ovat hyvin varmasti olemassa tietokoneellasi, joten seuraavien komentojen pitäisi toimia:

```
$ cp /etc/fstab /etc/hosts /etc/issue /etc/motd .
$ ls
fstab hosts issue motd
```

Älä unohda pistettä rivin lopussa! Muista, että se tarkoittaa "tämä hakemisto". Kuten näet, sinun täytyy antaa "cp"-komennolle lista tiedostoja, jotka tahdot kopioida, ja kohde, johon tiedostot kopioidaan. Jos tahdot nyt mennä takaisin kotihakemistoosi ja poistaa hakemiston, jota kutsutaan nimellä "harjoittelu", "rmdir" tuottaa virheviestin:

```
$ cd ..
$ rmdir harjoittelu
rmdir: ”harjoittelu”:n poisto epäonnistui: Hakemisto ei ole tyhjä
```

Voit käyttää ohjelmaa "rm" poistaaksesi tiedostot:

```
$ rm harjoittelu/fstab harjoittelu/hosts harjoittelu/issue
harjoittelu/motd
```

Nyt voit yrittää poistaa hakemiston uudestaan:

```
$ rmdir harjoittelu
```

Nyt se toimii, näyttämättä mitään ulostuloa.

Mutta mitä tapahtuu, jos hakemistojesi sisällä on hakemistoja, joiden sisällä on tiedostoja. Voisit joutua tyhjentämään hakemistoja viikkojen ajan! "rm"-komento ratkaisee tämän ongelman "-r" -optiolla, mikä merkitsee "rekursiivista". Seuraavassa esimerkissä komento epäonnistuu, koska "foo" ei ole pelkkä tiedosto:

```
$ rm foo/
rm: tiedostoa ”foo/” ei voi poistaa: Tiedostoa tai hakemistoa ei ole
```

Joten ehkä kokeilet "rmdir"-komentoa, mutta sekin epäonnistuu, koska "foo"-hakemiston alla on jotain muuta:

```
$ rmdir foo
rmdir: foo: File exists
```

Kun käytät komentoa "rm -r", se onnistuu eikä tuota viestiä.

```
$ rm -r foo/
```

Kun sinulla on suuri hakemisto, et joudu poistamaan jokaista alihakemistoa.

Ole varovainen, sillä "-r" on hyvin voimakas argumentti, ja voit menettää tietoa, jonka olisit tahtonut säilyttää!

CAT JA LESS

Et tarvitse editoria katsoaksesi tiedoston sisältöä. Sinun täytyy vain näyttää se. "cat"-ohjelma sopii tähän tarkoitukseen:

```
$ cat myspeech.txt
Friends, Romans, Countrymen! Lend me your ears!
```

Tässä "cat" vain avaa tiedoston "myspeech.txt" ja tulostaa koko tiedoston näytöllesi, niin nopeasti kuin kykenee. Kuitenkin, jos tiedosto on todella pitkä, sisältö menee ohi hyvin nopeasti, ja kun "cat" on valmis, näet pelkästään tiedoston viimeiset rivit. Katso pitkän tiedoston sisältöä (tai minkä tahansa tekstitiedoston sisältöä) käyttäen "less"-ohjelmaa:

```
$ less myspeech.txt
```

Aivan kuin käytettäessä komentoa "man", käytä nuolinäppäimiä navigointiin ja paina "q" poistuaksesi ohjelmasta.

7. PÄÄKÄYTTÄJÄ (ROOT)

Joidenkin järjestelmän osien ajatellaan vaativan erityistä suojelua. Jos joku voi muuttaa esimerkiksi "cat" tai "less" -peruskomentoja, hän voi saada sinut pilaamaan omia tiedostojasi. Joten eräitä komentoja voidaan käyttää vain erikoisoikeuksilla, joita kutsutaan pääkäyttäjän tai rootin oikeuksiksi.

Ennen vanhaan tietokonejärjestelmät maksoivat satojatuhansia dollareita ja niitä käyttivät sadat ihmiset, pääkäyttäjän oikeudet oli annettu tietyille henkilöille, jotka muodostivat eräänlaisen papiston tietokoneen temppelissä. Nykyisin jokainen PC:n omistaja voi ajaa pääkäyttäjän komentoja (tämä ei tosin ole aina totta mobiililaitteissa). Jokaisessa GNU/Linux -järjestelmässä on pääkäyttäjä, jonka nimi on root (juuri). Järjestelmä tekee tästä käyttäjästä herkkien järjestelmätiedostojen omistajan.

Juurikäyttäjällä ei ole mitään tekemistä tiedostojärjestelmän juurihakemiston ("/", englanniksi "root directory") kanssa.

Pääkäyttäjän komennot ovat voimakkaita ja niitä täytyy käyttää varovasti, mutta niiden käyttö on aika yleistä. Kun työpöydän käyttäjä asentaa ohjelmia, hänen täytyy ryhtyä pääkäyttäjäksi muutaman minuutin ajaksi.

KOMENTO SUDO

Monissa nykyaikaisissa järjestelmissä kirjoitat komennon "sudo" ennen kuin annat pääkäyttäjän komennon:

```
$ sudo rm -r /roska_hakemisto
```

Tämän jälkeen sinulta kysytään salasanaa, joten kukaan joka kävelee ohimennen koneellesi ei voi ajaa vaarallista komentoa. Järjestelmä pitää salasanasi muistissaan jonkin aikaa, joten voit kirjoittaa lisää pääkäyttäjän komentoja vaivautumatta kirjoittamaan salasanaa uudestaan.

Järjestelmä tarjoaa "su"-komennon, joka kirjaa sinut sisään pääkäyttäjänä ja antaa sinulle uuden päätekehotteen. Kaikki järjestelmät eivät anna käyttäjän käyttää sitä, koska käyttäjä voi innostua liikaa ja tehdä päivittäisiä töitä pääkäyttäjänä - ja huomata yllättäen, että on kaatanut järjestelmänsä kirjoitusvirheen vuoksi. On paljon parempi tehdä tavallinen järjestelmän ylläpito käyttäen komentoa "sudo".

Jos muut ihmiset jakavat järjestelmäsi ja tahdot antaa jollekin pääkäyttäjän oikeudet, tarvitset lisää tietoa järjestelmän ylläpidosta.

8. USEAMMAT TIEDOSTOT

Totuttuasi komentoriviin joudut etsimään tapoja tehdä enemmän lyhyemmässä ajassa. Yksi helpoimmista tavoista saada tämä aikaan on muokata monia tiedostoja samaan aikaan, joten sen sijaan että poistat tiedostot yksitellen:

```
$ rm vanha
$ rm turha
$ rm virhe
$ rm tarpeeton
```

Poistat vain kaikki näistä tiedostoista yhdellä komennolla. Monet komennot, kuten "rm", antavat valita kaikki poistettavat tiedostot kerralla:

```
$ rm vanha turha virhe tarpeeton
```

Edelleenkin, täytyy olla parempi tapa!

RYHMITTELY

Tiedostojen ryhmittely on komentorivin tapa käsitellä monia tiedostoja käyttäen mahdollisimman vähän kirjoitusmerkkejä. Pääte käsittelee joitain merkkejä koodeina, joita voit käyttää määrittelemään asioita ryhmiin, joihin tahdot komentojen vaikuttavan. Näitä merkkejä kutsutaan yleensä "jokerimerkeiksi", koska ne ovat kuin pelikortti, jonka pelaajat ovat määritelleet merkitsemään mitä tahansa.

JOKERIMERKKI "*"

Ajattele hakemistoa, jossa on tiedostoja:

```
$ ls
vanha turha virhe tarpeeton
```

jonka tahdot poistaa. Pitkäveteisen homman voi muuttaa yksinkertaiseksi käyttämällä "*" eli asteriski -jokerimerkkiä.

```
$ rm *
```

Kun sitä käytetään yksinään, asteriski-jokerimerkki viittaa kaikkiin tiedostoihin hakemistossa. Voimme sanoa, että pääte laajentaa jokerimerkkiä. Pääte tietää mitä hakemistossa on, se korvaa tiedostonimet asteriskilla ja toteuttaa käytännössä seuraavan komennon:

```
$ rm vanha turha virhe tarpeeton
```

Voit lisätä merkkiin "*" muita merkkejä tehdäksesi siitä valikoivan.

```
$ rm t*
$ ls
vanha virhe
```

Mitä tässä tapahtui? Pääte katsoi ensin merkkiä "t" ja laajensi sitten asteriskin käsittämään kaikki tiedostot, jotka alkoivat merkillä "t". Jos olisit yrittänyt sen sijaan "v*", olisi pääte poistanut kaikki tiedostot, jotka alkavat merkillä "v". Korvataanpa alkuperäiset tiedostot ja katsotaan mitä tapahtuu:

```
$ rm v* $ ls
turha tarpeeton
```

Asteriski -jokerimerkki voidaan sijoittaa mihin tahansa sanan sisällä. Siirrytäänpä "ls"-komentoon, koska on helpompi nähdä, mitä jokerimerkit tekevät:

```
$ ls tu*rha
turha
```

Siirtymällä komennosta "rm" komentoon "ls" näemme tärkeän jokerimerkin ominaisuuden: voit käyttää niitä minkä tahansa komennon kanssa, koska pääte käsittelee ne ennen kuin edes kutsuu komennon. Itse asiassa et voi antaa komentoa ottamatta huomioon jokerimerkkien käytöstä, koska ne ovat päätteen ominaisuus. (Onneksi tiedostonimissä ei koskaan ole asteriskia.)

Useampia asteriskeja voidaan myös käyttää yhdessä. Tällä tavalla voit esimerkiksi löytää tiedostonimiä, joissa sarjan keskiosa on sama, mutta ne alkavat ja päättyvät eri tavalla. Kokeillaanpa alkuperäisten neljän tiedoston kanssa:

```
$ ls *i*
virhe
```

Asteriskia käytetään usein poistamaan tiedostot, jotka ovat kaikki yhtä tyyppiä. Esimerkiksi, jos olet työskennellyt kuvien kanssa ja tahdot poistaa ".jpg" -loppuisia tiedostoja kun olet lopettanut, voit poistaa ne nykyisessä hakemistossasi seuraavalla tavalla:

```
$ rm *.jpg
```

Oletetaan, että sinulla on joitain ".jpg" -loppuisia tiedostoja ja joitain ".jpeg" -loppuisia tiedostoja. Asteriski tekee silti siivoamisesta helppoa:

```
$ ls *.jp*g
```

Tai oletetaan, että JPEG-tiedostot ovat jakautuneena useampaan alihakemistoon. Sinulla on hakemistot nimeltä "kuvat1", "kuvat2", "kuvat3" ja niin edelleen, jokaisessa on JPEG-tiedostoja, jotka tahdot poistaa. Jokerimerkki auttaa sinua listaamaan noiden alihakemistojen sisällön:

```
$ ls kuvat*
kuvat1:
keskusasema.jpg uusi_kirkko.jpg

kuvat2:
ica.jpeg teatteri.jpeg

kuvat3:
bayeux_katedraali.jpeg rouen_katedraali.jpeg matka.odt
```

Ja voit valita hakemiston poistettavien tiedostonimien kanssa:

```
$ rm kuvat*/*.jp*g
$ ls kuvat*

kuvat1:

kuvat2:

kuvat3:
matka.odt
```

Ainoastaan tiedosto "matka.odt" jää jäljelle (koska siinä ei ole merkkijonoa ".jp*g") kaikkien tekemiesi matkojen listana.

On kuitekin yksi rajoitus asteriski-jokerimerkin käyttöön. Oletusasetuksena se ei ota huomioon kätkettyjä tiedostoja (niitä tiedostonimiä, jotka alkavat pisteellä, sinun täytyy käyttää komentoa "ls -a" nähdäksesi nämä).

```
$ ls -a
.
..
.piilossa
turha
vanha
virhe
tarpeeton
$ rm *
$ ls -a
.
..
.piilossa
```

Jos tahdot tuhota nämä kätketyt tiedostot jokerimerkillä, on tarpeen laittaa piste jokerimerkin eteen. Huomaa, että normaaleja tiedostoja (eli niitä, jotka eivät ole kätkettyjä, eli eivät ala pisteellä) ei poisteta, kun teet näin:

```
$ls -a
.
..
.piilossa
$rm .*
$ls -a
.
..
```

Lopuksi, on tärkeää huomata, että asteriski voi myös sopia tyhjään. Seuraavassa listassa homma listataan niiden tiedostojen lisäksi, joilla on jotain asteriskiin sopivaa:

```
$ ls homma*
homma  hommaA  hommaB  hommaXY
```

"?"-JOKERIMERKKI

"?" tai kysymysmerkki -jokerimerkki on hyvin samantapainen kuin asteriski-jokerimerkki. Varsinainen ero on, että kysymysmerkki-jokerimerkki korvaa vain yhden kirjoitusmerkin.

```
$ ls homma*
homma  hommaA  hommaB  hommaXY
$ ls homma?
hommaA  hommaB
$ ls homma??
hommaXY
```

Kuten olemme jo nähneet, asteriski sopii kaikkiin tiedostoihin, jotka alkavat kirjaimilla "homma". Yksittäinen kysymysmerkki sopii tiedostoihin, joissa on yksi merkki "homma"-kirjainten jälkeen. Kaksi kysymysmerkkiä vaatii täsmälleen kahta kirjoitusmerkkiä tässä paikassa.

"[ ]" -JOKERIMERKKI

Hakasulku-jokerimerkit voivat olla vielä tarkempia, sillä ne osoittavat tietyn merkkivälin. Seuraava "ls" -komento sisältää "-1" -valitsimen, mikä merkitsee "listaa yksi merkki joka rivillä." Näin on helpompaa nähdä, kuinka tiedostot tässä esimerkissä eroavat toisistaan.

```
$ ls -1
```

```
tiedosto_1
tiedosto_2
tiedosto_3
tiedosto_a
tiedosto_b
tiedosto_c
```

Käyttämällä hakasulkuja voit poistaa tiettyjä tiedostoja kirjoittamatta jokaista nimeä kokonaan.

```
$ rm tiedosto_[1,3,a,c]
$ ls -1
tiedosto_2
tiedosto_b
```

Voit myös laittaa hakasulkuihin joukon yksittäisiä merkkejä ilman pilkkuja. Lopputulos olisi silti sama:

```
 $ rm tiedosto_[13ac]
```

Hakasulkujen sisällä merkkien järjestyksellä ei ole merkitystä.

Yhdistämällä hakasulut tavuviivan kanssa voit tehdä komennon tietyllä välillä olevalle joukolle tiedostoja. Aloitetaanpa hakemistosta, jossa on monta numeroihin päättyvää tiedostoa:

```
 $ ls -1
tiedosto_1
tiedosto_2
tiedosto_3
...
tiedosto_78
```

Aluksi voi olla houkuttelevaa käyttää asteriski-jokerimerkkiä tässä. Mitä jos meidän täytyy kuitenkin poistaa vain tiedostot 11-34? Voisimme käyttää pilkulla eroteltua muotoa hakasulku-jokerimerkistä, mutta joutuisimme silti kirjoittamaan 23 numeroa, sekä pilkut. Onneksi on paljon helpompikin tapa.

```
 $rm tiedosto_[11-34]
```

Nyt ainoat jäljelle jääneet tiedostot ovat 1-10 ja 35-78. Käyttämällä väliviivaa hakasuluissa olevien numerojoukkojen välissä, saat päätteen laajentamaan kuviota luomalla nimen, jossa on jokainen numero väliviivan vasemmalla olevasta alkuarvosta aina oikealla olevaan loppuarvoon asti.

Numerovälien sijasta voidaan käyttää myös kirjainvälejä.

```
 $ls -1
tiedosto_a
tiedosto_b
tiedosto_c
tiedosto_d $ rm tiedosto_[a-c] $ ls -1
tiedosto_d
```

Sekä pilkut että kirjain- ja numerovälit voidaan yhdistää samoihin hakasulkuihin.

```
 $ls
file_a
file_b
file_c
file_1
file_2 $rm file_[a-c,1,2] $ls
```

RYHMITTELY KUN MIKÄÄN TIEDOSTO EI SOVI

Oletetaan, että käytät jokerimerkkiä ja pääte ei löydä sopivaa tiedostonimeä.

```
$ ls -1
tiedosto_a
tiedosto_b
tiedosto_c
tiedosto_d $rm tiedosto?
rm: cannot remove `tiedosto?': No such file or directory
```

Kun ei ole kuvioon sopivaa tiedostoa, pääte lähettää jokerimerkin ohjelmalle laajentamattomana. Tämän vuoksi saat virheviestin "rm"-ohjelmalta, et päätteeltä.

JOKERIMERKIN OTTAMINEN POIS PÄÄLTÄ

Okei, tiedämme että päätteelle pitää lähettää jokerimerkki optiona ohjelmalle, kun se ei löydä tiedostoa, mutta mitä teemme, kun tahdomme lähettää ohjelmalle merkin, joka on myös jokerimerkki? Tässä on yleinen esimerkki: etsimme tiedostosta jokaista asteriskin ilmentymää.

```
$ ls
2tiedosto
*tiedosto
*?****[a-b]
```

Nyt tahdomme tiedoston "*tiedosto", mutta saamme:

```
 *file
2file
```

Miksi? Koska asteriski on jokerimerkki, pääte laajensi sen ennen kuin lähetti sen "ls"-komennolle. Joten laajennuksen jälkeen komento näyttää tälle:

```
$ ls *tiedosto 2tiedosto
```

Jos tahdomme "ls"-komennon löytävän asteriskin, täytyy tehdä jotain muuta.

"\" eli kenoviiva kertoo päätteelle, että seuraavaa merkkiä täytyy käsitellä normaalina merkkinä eikä jokerimerkkinä.

```
$ls \*tiedosto
*tiedosto
```

Koska asteriski on seuraava merkki kenoviivan jälkeen, pääte lähettää asteriskin ls-komennolle muuttamattomana. Toisin sanoen: kenoviiva päästää asteriskin pakoon. Kenoviivamerkki toimii hyvin, kun meillä on vain yksi jokerimerkki, jonka tahdomme lähettää ohjelmalle, mutta jos tahdomme laittaa kokonaisen merkkijonon, kuten "*?****[a-b]", jossa on paljon merkkejä, jotka tulkittaisiin normaalisti jokerimerkeiksi? Jos käytämme kenoviivoja niiden päästämiseksi pakoon, joudumme merkitsemään jokaisen merkin erikseen. Lyhyestä merkkijonosta tulisi: "*\?****

$$a - b$$

". Sen sijaan, että kirjoittaisimme kaksinkertaisen määrän merkkejä, käytämme paria heittomerkkiä.

```
$ls '*?****[a-b]'

*?****[a-b]
```

Pääte jättää muuttamatta jokaisen merkkijono, joka on suljettu heittomerkkeihin, jopa sen ollessa täynnä jokerimerkkejä.

JOKERIMERKIN KÄÄNTÄMINEN TOISIN PÄIN

Joskus tahdot kaikki tiedostot hakemistossa, paitsi ne, jotka sopivat kuvioon. Esimerkiksi sinulla on hakemisto, jossa on muutamia satoja tiedostoja ja noin viisikymmentä ei noudata mitään nimeämiskäytäntöä, mutta sataviisikymmentä on nimetty jonkin käytännön mukaan. Voisit kääntää jokerimerkkien merkityksen toisin päin, jolloin saisit juuri sen, mitä tahdot. Tässä kohdistinmerkki "^" ja hakasulut "[]" ovat käytännöllisiä.

```
$ rm [^tiedosto_]*
```

Kohdistinmerkki kääntää kuvion ympäri. Se käskee päätteen käsitellä kaikkia tiedostonimiä, jotka eivät sovi kuvioon. Tässä tempussa on kuitenkin yksi ongelma. Ainoastaan asteriski-jokerimerkki voidaan esittää hakasulkujen ulkopuolella.

```
$ ls -1
tiedosto_1
tiedosto_2
...
tiedosto_100
satunnainen_tiedosto
toinen_satunnainen_tiedosto
tiedosto_xyz
$ rm [^tiedosto_]* $ ls -1
tiedosto_1
tiedosto_2
...
tiedosto_100
tiedosto_xyz
```

Tässä tapauksessa tiedosto "tiedosto_xyz" sopii ryhmitykseen "tiedosto_", sitä ei laajenneta, joten "rm" ei tee sille mitään.

9. KAIKKI TÄMÄ KIRJOITTAMINEN...

Kaikki tämä kirjoittaminen voi muuttua jossain vaiheessa tylsäksi. Onneksi komentorivi tarjoaa monta tapaa tehdä työstäsi tehokkaampaa.

AUTOMAATTINEN TÄYDENTÄMINEN

Jokaisessa näppäimistössä on tabulaattorinäppäin, ja se on erittäin hyödyllinen päätteessä. Olet ehkä käyttänyt aiemmin tätä näppäintä sisentääksesi sanoja tekstinkäsittelyohjelmassa. Voit edelleenkin tehdä tämän GNU/Linux tekstinkäsittelyohjelmissa, mutta kun käytät tabulaattoria GNU/Linux-päätteessä, siitä tulee sellainen ajansäästäjä, että kun hallitset sen käytön, tulet käyttämään sitä koko ajan.

Pohjimmiltaan tabulaattori on automaattisen täydennyksen komento. Jos tahdon esimerkiksi siirtää tiedoston "dsjkdshdsdsjhds_ddsjw22.txt" johonkin komennolla "`mv`", voin joko kirjoittaa jokaisen tyhmän tiedostonimen kirjaimen, tai voin kirjoittaa komennon "`mv`" (siirrä), jota seuraa muutama ensimmäinen kirjain tiedostonimestä, jonka jälkeen painan tabulaattoria. Loput tiedostonimestä täydennetään automaattisesti. Jos tiedostonimeä ei kirjoiteta loppuun asti, se merkitsee, että on monta samoilla kirjaimilla alkavaa tiedostoa tai hakemistoa. Tämän korjaamiseksi voin kirjoittaa muutaman tiedostonimen kirjaimen lisää ja painaa taas tabulaattoria, tai helpottaakseni tehtävää voin painaa tabulaattoria kaksi kertaa, jolloin se antaa minulle listan noilla kirjaimilla alkavista tidostoista.

Voit myös käyttää tabulaattoria täyttämään komentojen nimet automaattisesti.

Tabulaattori on ystäväsi, käytä sitä paljon.

KOPIOI JA LIITÄ

Vaikka työskentelet komentorivillä voit silti käyttää monia graafisen käyttöliittymän mukavuuksia. Leikkaa ja liimaa toimii ehkä komentorivillä hieman eri tavalla kuin muissa käyttöjärjestelmissä, mutta havaitset sen pian hyvin intuitiiviseksi.

Tekstin kopiointi

Tekstin kopiointi toimii korostamalla teksti, jonka tahdot kopioida, pitämällä vasenta hiiren nappia alhaalla ja korostamalla teksti aivan kuin olet varmaankin tottunut tekemään. Tai napsauta vasenta nappia kaksi kertaa valitaksesi sanan tai kolme kertaa valitaksesi rivin.

Tekstin liittäminen

Korostamasi ja kopioimasi teksti on leikepöydällä, kunnes liität sen kursorisi sijaintiin painamalla keskimmäistä hiiren nappia (tai rullaa).

Huomaa: Jos hiiressäsi on vain kaksi nappia, *molempien painaminen yhdessä* tunnistetaan "keskimmäisen napin" painallukseksi.

Kuitenkin se toimii näin päätteessä, jossa ei ole grafiikkaa. Voit huomata, että se ei ole näin työpöydällä. Voi siis olla hyvä idea kirjautua sisään teksti-istuntoa varten. Käytä näppäimiä "" päästäksesi pois työpöydältä.

Kokeile sitä! Valitse alla oleva kappale vasemmalla hiiren napilla, avaa uusi pääte ja liitä teksti keskimmäisellä hiiren napilla.

```
echo "Tämä on liitetty teksti."
```

Kun näet tekstin päätteessä, paina Enter-näppäintä ja `"echo"` -komento toistaa tekstin komentorivin lainausmerkkien välissä.

Huomaa: Jos kopioit tekstiä verkkosivulta välimerkkejä ei aina käsitellä oikein. Voit itse asiassa kopioida näkymätöntä muotoilua tekstin mukana, mikä hajottaa kopioimasi komennon syntaksin.

HISTORIA

On myös mahdollista käyttää näppäimistön nuolia ylös ja alas navigoimaan takaisin ja eteenpäin kirjoittamiesi komentojen historian läpi. Kun navigoit tällä tavalla aikaisempaan komentoon, on mahdollista painaa Return tai Enter -näppäintä ja komento suoritetaan uudelleen. Voit muokata sitä ensin saadaksesi sen tekemään jotain muuta.

KESKIVAIKEITA KOMENTOJA

10. KOMENTOJEN PUTKITUS
11. KOMENTOHISTORIAN OIKOPOLUT
12. LISÄÄ UUDELLEENOHJAUKSESTA

10. KOMENTOJEN PUTKITUS

Putket antavat ohjelmien toimia yhteistyössä yhdistämällä yhden ohjelman ulostulon toisen ohjelman sisääntuloon. Käsitteellä "ulostulo" on tässä tarkka merkitys: ohjelma kirjoittaa sen standardiulostuloon C-ohjelmointikielen komentojen avulla. Näitä komentoja ovat "printf" ja vastaavat. Normaalisti ulostulo näkyy terminaalin ruudulla. "Sisääntulo" tarkoittaa standardisisääntuloa, joka tulee yleensä näppäimistöltä. Putket rakennetaan käyttämällä pystysuoraa palkkia ("|") putken symbolina.

Niinpä voit auttaa englanninkielistä Hortense-tätiäsi järjestelemään yksityistä kirjakokoelmaansa. Sinulla on "kirjat"-niminen tiedosto, joka sisältää hänen kirjansa, yksi kirja jokaisella rivillä, muodossa "kirjoittaja:kirja", jotenkin tähän malliin:

```
$ cat kirjat
Carroll, Lewis:Through the Looking-Glass
Shakespeare, William:Hamlet
Bartlett, John:Familiar Quotations
Mill, John Stuart:On Nature
London, Jack:John Barleycorn
Bunyan, John:Pilgrim's Progress, The
Defoe, Daniel:Robinson Crusoe
Mill, John Stuart:System of Logic, A
Milton, John:Paradise Lost
Johnson, Samuel:Lives of the Poets
Shakespeare, William:Julius Caesar
Mill, John Stuart:On Liberty
Bunyan, John:Saved by Grace
```

Tämä on melko epäjärjestelmällistä, sillä ne eivät ole missään järkevässä järjestyksessä. Käytetäänpä komentoa "sort" järjestämään ne:

```
$ sort kirjat
Bartlett, John:Familiar Quotations
Bunyan, John:Pilgrim's Progress, The
Bunyan, John:Saved by Grace
Carroll, Lewis:Through the Looking-Glass
Defoe, Daniel:Robinson Crusoe
Johnson, Samuel:Lives of the Poets
London, Jack:John Barleycorn
Mill, John Stuart:On Liberty
Mill, John Stuart:On Nature
Mill, John Stuart:System of Logic, A
Milton, John:Paradise Lost
Shakespeare, William:Hamlet
Shakespeare, William:Julius Caesar
```

Nyt tuloksena on siisti kirjoittajan mukaan järjestetty lista. Entäpä pelkästään kirjoittajien nimien listaaminen ilman kirjojen nimiä? Voit tehdä tämän komennolla "cut":

```
$ cut -d: -f1 kirjat
Carroll, Lewis
Shakespeare, William
Bartlett, John
Mill, John Stuart
London, Jack
Bunyan, John
Defoe, Daniel
Mill, John Stuart
Milton, John
Johnson, Samuel
Shakespeare, William
Mill, John Stuart
Bunyan, John
```

Pieni selvitys tarvitaan. Valitsin `"-d"` valitsi kaksoispisteen erottimeksi. Tämä käskee komennon `"cut"` leikata jokainen rivi erottimen ilmestyessä. Jokaista rivin erillistä osaa kutsutaan kentäksi. Meidän formaatissamme kirjoittajan nimi näkyy ensimmäisenä kenttänä, joten meidän täytyy laittaa "1 -f" valitsimeen kertoaksemme komennolle `"cut"` mitä tahdomme nähdä tässä kentässä.

Mutta näet, että lista on taas järjestämättä. Putket apuun!

```
$ sort kirjat | cut -d: -f1
Bartlett, John
Bunyan, John
Bunyan, John
Carroll, Lewis
Defoe, Daniel
Johnson, Samuel
London, Jack
Mill, John Stuart
Mill, John Stuart
Mill, John Stuart
Milton, John
Shakespeare, William
Shakespeare, William
```

Hyvä! Olet ottanut aakkosjärjestyksessä kirjoitetun listan, joka on komennon "sort" ulostulo, ja syöttänyt sen sisääntulona komennolle "cut". Älä anna komennolle "cut" tiedostonimeä käytettäväksi, sillä tahdot sen käsittelevän tekstiä, joka on putkitettu ulos "sort"-komennosta.

Putket ovat niin yksinkertaisia - teksti virtaa putkea pitkin komennosta toiseen.

Entäpä jos tahtoisitkin järjestetyn listan kirjojen nimistä? Koska kirjojen nimet ovat toisessa kentässä, kokeillaan käyttää valitsinta `"-f2"` `"cut"` -komennossa valitsimen `"-f1"` sijasta:

```
$ sort kirjat | cut -d: -f2
Familiar Quotations
Pilgrim's Progress, The
Saved by Grace
Through the Looking-Glass
Robinson Crusoe
Lives of the Poets
John Barleycorn
On Liberty
On Nature
System of Logic, A
Paradise Lost
Hamlet
Julius Caesar
```

Hups. Mitä tapahtui? Kun katsot putkea, sinun pitää katsoa vasemmalta oikealle. Tässä tapauksessa järjestimme ensimmäisen tiedoston ennen kirjojen nimien uuttamista. Niinpä se järjesti rivit velvollisuudentuntoisesti ensimmäisellä rivillä olevien kirjailijoiden nimien mukaan. Saadaksesi *kirjojen nimet* oikeaan järjestykseen, joudut järjestämään ne vasta niiden uuttamisen *jälkeen*:

```
$ cut -d: -f2 kirjat | sort
Familiar Quotations
Hamlet
John Barleycorn
Julius Caesar
Lives of the Poets
On Liberty
On Nature
Paradise Lost
Pilgrim's Progress, The
Robinson Crusoe
Saved by Grace
System of Logic, A
Through the Looking-Glass
```

Paljon parempi. Tämä kaikki on hienoa, mutta ehkä olisit voinut tehdä nämä asiat taulukkolaskentaohjelmalla. Yksinkertaisempien tehtävien suhteen tämä on luultavasti totta. Mutta oleta, että täti Hortensen tapana on kysyä outoja kysymyksiä kokoelmastaan. Hän tahtoo esimerkiksi tietää, kuinka monta kirjaa hänellä on John-nimisiltä kirjailijoilta. Taulukkolaskentaohjelmalla tai muulla graafisella ohjelmalla voi olla vaikeuksia käsitellä kysymyksiä, joita ohjelman tekijät eivät odottaneet. Komentotulkki tarjoaa kuitenkin monia yksinkertaisia komentoja, jotka voidaan yhdistää ennennäkemättömillä tavoilla monimutkaisten tehtävien toteuttamiseksi.

Löytääksesi tietyn merkkijonon tekstiriviltä voit käyttää "grep"-komentoa. Huomaa, että yhdistäessäsi komentoja niiden täytyy olla oikeassa järjestyksessä. Et voi käyttää komentoa "grep" tiedostoon ensin, sillä se sopii otsikkoon "John Barleycorn" John-nimisten kirjoittajien lisäksi. Joten lisää se putken loppuun:

```
$ cut -d: -f1 kirjat | sort | grep "John"
Bartlett, John
Bunyan, John
Bunyan, John
Johnson, Samuel
Mill, John Stuart
Mill, John Stuart
Mill, John Stuart
Milton, John
```

Tämä vie meidät lähemmäs, mutta nyt et tahdo saada nimeä "Samuel Johnson" listalle ja suututtaa Hortense-tätiä. Usein työskennelläsi "grep"-komennolla joudut tarkentamaan sopivaa tekstiä täsmälleen siihen, mitä tarvitset. Komento grep tarjoaa "-w" -valitsimen, joka antaa sen sovittaa "John" ainoastaan kun "John" on kokonainen sana, ei sen ollessa osa sanaa "Johnson". Mutta ratkaisemme tämän ongelman lisäämällä pilkun ja välilyönnin sovitettavan merkkijonon eteen, jolloin se sopii ainoastaan Johnin ollessa etunimi:

```
$ cut -d: -f1 kirjat | sort | grep ", John"
Bartlett, John
Bunyan, John
Bunyan, John
Mill, John Stuart
Mill, John Stuart
Mill, John Stuart
Milton, John
```

Tämä on parempi. Nyt joudut vain laskemaan yhteen kirjojen määrän jokaiselta kirjoittajalta. Pieni komento, jota kutsutaan nimellä "uniq" toimii hyvin. Se poistaa samanlaiset rivit (niiden täytyy olla toisiaan seuraavia rivejä, joten varmista, että tekstisi on järjestetty ensin) ja valitsimen "-c" kanssa se tarjoaa myös laskee samanlaiset rivit:

```
$ cut -d: -f1 kirjat | sort | grep ", John" | uniq -c
      1 Bartlett, John
      2 Bunyan, John
      3 Mill, John Stuart
      1 Milton, John
```

Siinäpä se! Hyvin järjestetty lista Johneja ja heidän kirjojaan. Esimerkkimme on yksinkertainen homma ja voisit tehdä sen kynällä ja paperilla. Mutta tämä sama putki voisi käsitellä paljon suurempia tietomääriä - se ei välittäisi, vaikka täti Hortensella olisi satojatuhansia kirjoja navetassaan.

Järjestelmänvalvojat käyttävät tämän kaltaisia putkia usein käsitelläkseen verkko- ja sähköpostipalvelinten lokitiedostoja. Sellaiset tiedostot voivat venyä kymmeniin tai satoihin megatavuihin ja komentoputki voi olla nopea tapa luoda tilastoja ilman koko lokin lukemista.

Hyvä puoli putkien rakentamisessa on, että voit tehdä sen komento kerrallaan, nähden täsmälleen jokaisen vaikutuksen ulostuloon. Tämä voi auttaa sinua havaitsemaan koska tarvitset valitsinten muuttamista tai komentojen järjestyksen vaihtamista. Esimerkiksi laittaaksesi kirjoittajat järjestykseen kirjojen lukumäärän mukaan voit vain lisätä komennon `"sort -nr"` edelliseen putkeen:

```
$ cut -d: -f1 kirjat | sort | grep ", John" | uniq -c | sort -
nr
      3 Mill, John Stuart
      2 Bunyan, John
      1 Milton, John
      1 Bartlett, John
```

Kokeile!

11. KOMENTOHISTORIAN OIKOPOLUT

Komentotulkki antaa sinun hakea vanhoja komentoja ja syöttää ne uudelleen, voit myös muuttaa niitä tarvittaessa. Tämä on yksi helpoimmista ja tehokkaimmista tavoista vähentää kirjoittamista, sillä toistuvat komentojaksot ovat hyvin yleisiä. Esimerkiksi seuraavassa komentojaksossa käymme läpi eri hakemistoja, listaamme niiden sisällön, poistamme tarpeettomat tiedostot ja tallennamme tiedostoja eri nimillä:

```
cd Kuvat/
ls -l status.log.*
rm status.log.[3-5]
mv status.log.1 status.log.bak

cd ../Asiakirjat/
ls -l
status.log*

rm status.log.[2-4]
mv status.log.1 status.log.bak

cd ../Videot/
ls -l status.log*
rm status.log.[2-5]
mv status.log.1 status.log.bak
```

Jos joudut tekemään tällaista siivoamista usein, tahdot ehkä kirjoittaa skriptin, joka automatisoi sen, ja ehkä käyttää "cron"-tehtävää ajamaan sen säännöllisin väliajoin. Mutta toistaiseksi katsomme vain, kuinka voit suuresti vähentää kirjoittamisen määrää syöttäessäsi komentoja käsin.

Edellisessä luvussa katsoimme, kuinka nuolinäppäimiä käytetään liikuttaessa ympäri komentohistoriaa aivan kuin olisit editoimassa tiedostoa. Tässä luvussa tarkastelemme monimutkaisempaa ja vanhempaa tapaa manipuloida komentohistoriaa. Joskus tämän luvun menetelmät tuntuvat helpommilta, joten on vaivan arvoista harjoitella niitä. Oleta esimerkiksi, että kirjoitit tunti sitten komennon "mv" ja tarvitset sitä nyt. On vaikeampaa painaa paluunuolta monta kertaa kuin käyttää tässä luvussa esiteltyä tekniikkaa.

KOMENNON PALAUTTAMINEN MERKKIJONOLLA

Huutomerkkioperaattori on nimetty huutomerkin "!" mukaan ja se antaa sinun toistaa tiedostohistoriassa olevia komentoja.

"!merkkijono" suorittaa viimeisimmän komennon, joka alkaa *merkkijonolla*. Niinpä täsmälleen saman "mv"-komennon suorittamiseksi voit kirjoittaa:

```
!mv
```

Ehkä et tahdo täsmälleen samaa komentoa? Jos tahdot muuttaa sitä hieman ennen sen suorittamista? Tai ehkä tahdot vain katsoa, mitä huutomerkkioperaattori palauttaa, jotta voit varmistaa, että se on oikea komento? Voit palauttaa sen ilman sen suorittamista lisäämällä valitsimen ":p" (joka tarkoittaa tulostamista):

```
!mv:p
```

Näytämme kohta komentojen editoinnin.

Ehkä olet antanut paljon "mv"-komentoja, mutta tiedät, että komennon keskellä on ainutlaatuinen merkkijono, jonka tahdot? Ympäröi merkkijono kysymysmerkeillä, kuten seuraavassa:

```
!?log?
```

Kaksi huutomerkkiä peräkkäin toistaa viimeksi ajetun komennon. Erittäin hyödyllinen komentohistoriatekniikka on viimeisen komennon ajaminen pääkäyttäjän oikeuksilla:

```
sudo !!
```

sillä kirjoittelemme usein komentoja ilman oikeita käyttöoikeuksia.

Viimeisimmän komennon ajaminen voi vaikuttaa melko hyödyttömältä, mutta tätä menetelmää voidaan muuttaa ajamaan vain osia viimeisestä komennostasi, kuten näemme myöhemmin.

KOMENNON PALAUTTAMINEN NUMERON PERUSTEELLA

Komentorivi numeroi jokaisen suoritetun komennon järjestyksessä. Jos tahdot palauttaa komentoja numeron perusteella, voit muuttaa komentokehotettasi sisältämään numeron (myöhempi luku näyttää tämän). Voit myös katsoa komentojen listaa ajamalla komennon "history":

```
$ history
...
  502  cd Kuvat/
  503  ls -l status.log*
  504  rm status.log.[3-5]
  505  mv status.log.1 status.log.bak
  506  cd ../Asiakirjat/
  507  history
$
```

Tässä olemme näyttäneet vain ulostulon viimeiset muutamat rivit. Jos tahdot suorittaa uudestaan viimeisimmän "rm"-komennon (komento numero 504), voit tehdä sen syöttämällä komennon:

```
!504
```

Mutta numerot ovat luultavasti käyttökelpoisempia, kun ajattelet takaperin. Jos esimerkiksi muistat antaneesi "rm"-komennon, jota seurasi kolme muuta komentoa, voit ajaa "rm"-komennon seuraavasti:

```
!-4
```

Tämä käskee komentotulkin aloittaa nykyisestä sijainnista, laskea taaksepäin neljä komentoa ja suorittaa siinä sijaitsevan komennon.

Parametrien toistaminen

Usein tahdot suorittaa edellisen komennon osia, joko tehtyäsi kirjoitusvirheen, tai koska ajat komentojen sarjaa tiettyä tehtävää varten. Me saamme tämän aikaan käyttämällä huutomerkkioperaattoria valitsimilla.

Kolme hyödyllisintä valitsinta ovat: "*", "!^", ja "!$", jotka ovat oikotiet kaikkiin, ensimmäiseen ja viimeiseen parametriin. Katsotaan näitä järjestyksessä.

"komennonNimi *" suorittaa komennon "komennonNimi" millä tahansa parametreillä, joita käytit viimeisessä komennossasi. Tämä voi olla hyödyllistä, jos teit kirjoitusvirheen. Esimerkiksi kirjoitettuasi "emasc" sen sijaan, että olisit kirjoittanut "emacs":

```
emasc /home/fred/mywork.java /tmp/testme.java
```

Tämä epäonnistuu selvästi. Nyt voit kirjoittaa:

```
emacs !*
```

Tämä ajaa komennon `"emacs"` viimeksi kirjoittamillasi parametreillä. Se on sama, kuin jos kirjoittaisit:

```
emacs /home/fred/mywork.java /tmp/testme.java
```

"komennonNimi !^" toistaa ensimmäisen parametrin.

```
emacs /home/fred/mywork.java /tmp/testme.java
svn commit !^    # vastaa komentoa: svn commit /home/fred/mywork.java
```

"komennonNimi !$" toistaa viimeisen parametrin.

```
mv /home/fred/downloads/sample_screen_config /home/fred/.screenrc
emacs !$     # vastaa komentoa: emacs /home/fred/.screenrc
```

Voit käyttää näitä myös yhdistettynä. Ehkä kirjoitit:

```
mv mywork.java mywork.java.backup
```

kun tahdoit oikeastaan tehdä kopion. Voit korjata tämän suorittamalla:

```
cp mywork.java.backup mywork.java
```

Mutta koska käytät parametrejä päinvastaisessa järjestyksessä, hyödyllinen oikotie olisi:

```
cp !$ !^
```

Hienovaraisempaa parametrien hallintaa varten voit käyttää kaksoishuutomerkkiä valitsimella ":N" valitaksesi parametrin numero N. Tämä on käyttökelpoisinta, kun ajat komennon sudona, sillä alkuperäisestä komennostasi tulee ensimmäinen parametri. Alla on esimerkki, joka esittelee sen tekemisen.

```
sudo cp /etc/apache2/sites-available/siteconfig
/home/fred/siteconfig.bak
echo !^ !!:2  # vastaa komentoa echo cp /etc/apache2/sites-
available/siteconfig
```

Kantama on mahdollinen myös valitsimella `"!!:M-N"`.

Parametrien muokkaaminen

Usein tahdot suorittaa edellisen komennon, mutta muuttaa yhden merkkijonon sen sisällä. Oletetaanpa, että tahdot ajaa komennon tiedostolle "tiedosto1":

```
$ wc tiedosto1
     443    1578    9800 tiedosto1
```

Nyt tahdot poistaa tiedoston "tiedosto2", jolla on melkein sama nimi kuin tiedostolla "tiedosto1". Voit käyttää edellisen komennon viimeistä parametriä merkeillä "!$", mutta muuta sitä seuraavasti:

```
$ rm !$:s/1/2/
rm tiedosto2
```

Tämä näyttää hieman monimutkaiselta, joten puretaan parametri:

```
!$   :   s/1/2/
```

Merkkiä "!$" seuraa kaksoispiste ja sitten "s"-komento, joka merkitsee "korvausta". Tämän jälkeen on merkkijono, jonka tahdot korvata (1) ja merkkijono, jonka tahdot laittaa sen tilalle (2), joita ympäröivät kauttaviivat. Komentotulkki tulostaa komennon samoin kuin se tulkitsee syötteesi, jonka jälkeen se suorittaa sen.

Koska tällainen korvaaminen on niin yleistä, voit olla tyytyväinen kuullessasi, että on yksinkertaisempi tapa suorittaa komento uudestaan pienellä muutoksella. Voit muuttaa vain yhden merkkijonon komennossa tällä syntaksilla:

```
$ wc tiedosto1
     443    1578    9800 tiedosto1
$ ^1^2
wc tiedosto2
```

Käytimme tarkemerkkiä "^", merkkijonoa, jonka tahdomme korvata, toista tarkemerkkiä "^", ja merkkijonoa, jonka tahdomme laittaa sen tilalle.

KOMENTOHISTORIAN LÄPI ETSIMINEN

Käytä näppäinyhdistelmää **Ctrl + R** suorittaaksesi käänteisen haun (englanniksi "reverse-i-search"). Jos esimerkiksi tahdot käyttää komentoa, jota käytit käyttäessäsi viimeksi komentoa "snort", aloita näppäimillä **Ctrl + R**. Terminaali-ikkunassa näet:

```
(reverse-i-search)`':
```

Kun kirjoitat jokaisen kirjaimen (s, n, jne.), komentotulkki näyttää viimeisimmät komennot, joissa tuo merkkijono on jossain. Kun lopetat merkkijonon "snort" kirjoittamisen, voit käyttää näppäimiä **Ctrl + R** toistuvasti etsiäksesi taaksepäin kaikkien merkkijonon "snort" sisältävien komentojen läpi. Kun löydät komennon, jota olet etsimässä, voit painaa oikeaa tai vasenta nuolta sijoittaaksesi komennon varsinaiselle komentoriville, jotta voit muokata sitä, tai vain painaa **Enter** suorittaaksesi komennon.

BASHIN HISTORIAN JAKAMINEN

Bash-komentotulkki tallentaa komentohistorian, jotta voit palauttaa edellisten istuntojen komentoja. Mutta historia tallennetaan vain sulkiessasi päätteen. Jos työskentelet kahdella päätteellä samaan aikaan, tämä merkitsee, että et voi jakaa komentoja.

Korjataksesi tämän - jos tahdot päätteen tallentavan jokaisen komennon heti sen suorittamisen jälkeen - lisää seuraavat rivit *"~/.bashrc"* -tiedostoosi:

```
shopt -s histappend
PROMPT_COMMAND='history -a'
```

Näiden oikoteiden opetteleminen voi säästää sinulta valtavan määrän aikaa, joten ole hyvä ja kokeile!

12. LISÄÄ UUDELLEENOHJAUKSESTA

Kuinka putket toimivat? Ne käyttävät kolmea viestintäkanavaa, jotka ovat tarjolla jokaiselle suoritettavlle tiedostolle.

stdin (standardisisääntulo) on oletusarvoisesti se, mitä kirjoitamme näppäimistöllä. Voimme käyttää merkkiä "<" tiedostonimen kanssa saadaksemme ohjelman ottamaan sisääntulon tiedostosta.

stdout (standardiulostulo) on oletusarvoisesti tulostus tietokoneesi ruudulle. Voimme käyttää merkkiä ">" tiedostonimen kanssa lähettääksemme tuohon tiedostoon, kirjoittaen yli kaiken sen sisällön, tai käyttää merkkejä ">>" lisätäksemme standardiulostulon tiedoston loppuun.

stderr (standardivirhe) on vaihtoehtoinen ulostulon tyyppi. Ohjelmat käyttävät sitä lähettääkseen virheviestejä. Tämä voi olla käyttökelpoista, sillä saatat tahtoa nähdä virheviestejä päätteellä, vaikka uudelleensuuntaisit ulostulon tiedostoon. Tässä on esimerkki:

```
$ ls *.bak > listatiedosto
ls: *.bak: No such file or directory
```

Tässä tahdoimme listata kaikki tiedostot, jotka päättyvät ".bak". Sellaisia tiedostoja ei kuitenkaan ole tässä hakemistossa. Jos "ls" lähetti virheviestinsä standardiulostuloon (joka on tässä tapauksessa uudelleenohjattu tiedostoon), emme tietäisi, että tässä on ongelma katsomatta tiedoston "listatiedosto" sisältöä. Mutta koska "ls" lähetti viestinsä standardivirheeseen, näemme sen. Virheviesti alkaa ohjelman nimellä ("ls"), jota seuraa kaksoispiste ja varsinainen virheviesti.

Putki yksinkertaisesti uudelleensuuntaa ensimmäisen ohjelman standardiulostulon toiseen ohjelmaan:

```
$ ls *.bak | more
```

Joskus tahdomme suunnata komennon ulostulon tiedostoon, mutta tahdomme myös nähdä ulostulon ohjelman toimiessa. Komento "tee" tekee juuri sen:

```
$ ls -lR / | tee kaikkiTiedostoni
```

tarjoaa täydellisen, yksityiskohtaisen listan tiedostojärjestelmästäsi, joka tallennetaan tiedostoon *"kaikkiTiedostoni"*. Tämän suorittaminen kestää jonkin aikaa; "tee" pelastaa sinut tuijottamasta elotonta ruutua, ihmettellen tapahtuuko mitään.

Jokainen ohjelma voi avata paljon tiedostoja, joista jokaisella on numero, jota kutsutaan "tiedostokuvaajaksi", joka on merkityksellinen vain tuon ohjelman sisällä. Ensimmäiset kolme numeroa on aina säästetty tiedostokuvaajille, joita juuri kuvasimme.

stdin	0
stdout	1
stderr	2

UUDELEENSUUNNATAAN STDERR

Kun uudelleensuuntaamme standardisisäänulostulon stdin, kuten teimme aiemmin, virheviestit ilmestyvät edelleenkin ruudulle. Esimerkiksi

```
$ ls /eisellaistapaikkaa > /dev/null
ls: /eisellaistapaikkaa: No such file or directory
$
```

Uudelleensuunnataksemme "stderr"-standardivirheviestit voimme käyttää yleisempää uudelleensuuntauksen muotoa, joka käyttää äsken mainittuja tiedostonumeroita, ja näyttää tältä:

```
$ ls /eisellaistapaikkaa 2>/tmp/errors
$
```

Tämä lähettää tiedostoon numero "2" ("*stderr*") lähetetyn virheviestin tiedostoon *"/tmp/errors"*.

Voimme nyt esitellä monimutkaisemman uudelleensuuntauksen, joka suuntaa standardiulostulon ja standardivirheviestit samaan tiedostoon:

```
$ ls *.bak > listfile 2>&1
```

Merkki "&" tässä komennossa ei liity mitenkään komennon laittamiseen taustalle. Merkin "&" täytyy tässä seurata suoraan merkkiä ">", ja se lähettää tiedoston numero "2" tiedostoon numero "1".

Putken tapauksessa tämä pitää laittaa ennen putkea:

```
$ ls *.bak 2>&1 | more
```

LISÄTÄÄN ENEMMÄN KUVAAJIA

Joskus on kätevää pitää muita tiedostoja avoinna ja lisätä niihin vähitellen. Voit tehdä tämän uudelleensuuntauksella ja komennolla `"exec"`.

```
$ exec 3>/tmp/kolmastiedosto
$ exec 4>/tmp/neljästiedosto
$ echo tip >&3
$ echo tap >&4
$ echo taas tip >&3
$ echo taas tap >&4
$ exec 3>&-
$ exec 4>&-
```

Ensimmäiset kaksi riviä avaavat yhteydet kahteen muuhun tiedostokuvaajaan, jotka ovat "3" ja "4". Voimme tämän jälkeen laittaa niihin tekstiä komennolla "echo", uudelleensuunnata niihin ohjelmia ja niin edelleen, käyttäen ainoastaan komentoja `">&3"` tai `">&4"`. Lopulta voimme sulkea ne syntaksilla `"3>&-"` ja `"4>&-"`.

KEHITTYNEEMPIÄ KOMENTOJA

13. KÄYTTÖOIKEUDET
14. INTERAKTIIVINEN EDITOINTI
15. LOPPUTILA
16. ALIKOMENNOT
17. LIIKUTAAN TAAS
18. HYÖDYLLISTÄ KUSTOMISOINTIA
19. PARAMETRIEN KORVAAMINEN
20. GNU SCREEN
21. SSH
22. OHJELMIEN ASENTAMINEN

13. KÄYTTÖOIKEUDET

Tietokonejärjestelmäsi varastoi paljon tietoa tiedostoista. Tämä tieto pysyy normaalisti piilossa, kun luot ja käytät tiedostoja. Yksi ominaisuuksien joukko, johon törmäät, on käyttöoikeudet. Kuka voi muokata tiedostojasi? Toivottavasti ei jokainen järjestelmän käyttäjä (sillä monia järjestelmiä käyttävät nykyisin useammat käyttäjät). Tämä käyttöoppaan osa keskustelee omistajuudesta ja käyttöoikeuksista.

Ensinnäkin, katsotaanpa, mitä järjestelmä itse voi kertoa tiedostoistaan. Suoritamme tutun "ls"-komennon valitsimella "-l", joka antaa pitkän listan:

```
$ ls -l
total 72
drwxr-xr-x   2 root root  4096 Oct  5 09:31 bin
drwxr-xr-x   3 root root  4096 Oct  9 21:47 boot
drwxr-xr-x   1 root root     0 Jan  1  1970 dev
...
```

Ensimmäinen rivi:

```
total 72
```

näyttää kaikkien tiedostojen yhtenäiskoon *kilotavuissa* (kB). Loppu tarjoaa tietoa tiedostoista ja hakemistoista itsessään. Tämä tieto on ryhmitetty seitsemään palkkiin, joita voidaan luonnehtia seuraavasti:

```
Käyttöoikeudet Linkit Omistaja Ryhmä Koko Muutospäivämäärä
Tiedostonimi
```

MITÄ VOIN TEHDÄ? MITÄ MUUT VOIVAT TEHDÄ?

Jokaisella tiedostolla ja hakemistolla järjestelmässä on *omistaja*, se kuuluu *ryhmään* ja siihen liittyy joukko *käyttöoikeuksia*. Yksinkertaisimmalla tasolla nämä käyttöluvat määrittelevät kolme pääsytasoa, yhden tiedoston omistajalle, yhden ryhmälle johon tiedosto kuuluu ja yhden muulle maailmalle. (Itse asiassa maailma tarkoittaa vain niitä ihmisiä, joilla on lupa kirjautua järjestelmään.)

Katsotaanpa taas aiemmin näytettyä ulostuloa. Kolmannessa ja neljännessä palkissa näkyy omistaja (tässä tapauksessa pääkäyttäjä) ja ryhmä (myös pääkäyttäjä). Ensimmäisessä palkissa näkyy käyttöoikeudet hyvin tiiviissä muodossa, kuten tässä:

```
drwxr-xr-x
```

Ensimmäinen merkki tarkoittaa tiedoston tyyppiä, seuraavat kolme merkkiä näyttävät omistajan oikeudet, sitä seuraavat kolme ovat ryhmän oikeudet ja viimeiset kolme ovat muun maailman oikeudet.

Seuraava taulukko näyttää ensimmäisen merkin merkityksen. Edellisessä esimerkissä oli kirjain "d", joka tarkoittaa hakemistoa. Jotkin näistä merkeistä ovat melko harvinaisia. Yleensä sinun tarvitsee ajatella vain tavallista tiedostoa ja hakemistoa.

Merkki	Kuvaus
-	tavallinen tiedosto
d	hakemisto
l	symbolinen linkki
b	erityinen blokkitiedosto
c	erityinen kirjaintiedosto
p	FIFO (nimetty piippu)
s	suoritinkanta
?	ls ei ole tunnistanut tätä

Käyttöluvat on jaettu kolmeen tyyppiin:

- Lue (r): lupa lukea tiedostoa
- Kirjoita (w): lupa kirjoittaa tiedostoon
- Suorita (x): lupa suorittaa tiedostoa

Yhdysmerkki (-) merkitsee mitä tahansa käyttölupaa, jota ei ole asetettu.

Yksinkertaisempi tapa nähdä käyttölupien jakautuminen näytetään alla:

```
:tyyppi : omistaja : ryhmä : muu maailma:
:d      : rwx      : r-x   : r-x
```

Jos tahdot nähdä tiedoston sisällön, tarvitset *lukuoikeuden*. Jos tahdot muukata sen sisältöä, tarvitset *kirjoitusoikeuden*. Jos tiedosto on ohjelma ja tahdot suorittaa sen, tarvitset *suoritusoikeuden*.

Jos tahdot nähdä hakemiston sisällön, tarvitset *sekä luku- että suoritusoikeudet,* pelkät *lukuoikeudet* eivät ole tarpeeksi. Jos tahdot lisätä tai poistaa tiedostoja tuosta hakemistosta, tarvitset *kirjoitusoikeuksia.*

Palataanpa takaisin esimerkkiin ja pohditaan seuraavaa riviä:

```
drwxr-xr-x   2 root root  4096 Oct  5 09:31 bin
```

Kuten nyt näet, se on hakemisto. Kuinka voit saada selville, mitä *sinä* voit itse asiassa tehdä sillä? Sinun täytyy katsoa tiedostolle asetettua käyttäjää ja ryhmää. Mutta ensin: kuka sinä olet?

```
$ whoami
joe
```

Juuri tämän "whoami" kertoo sinulle: kuka olet ja käyttäjätunnuksesi nimen. Kuten näet, et ole juurikäyttäjä. Juurikäyttäjä voi nähdä hakemiston sisällön ja lisätä siihen tiedostoja, mutta et ole juuri. Mikä ryhmäsi sitten on?

```
$ id -G -n
joe dialout cdrom floppy audio video plugdev
```

Tämä on lista ryhmistä, joihin kuulut. Jos joku näistä ryhmistä olisi ollut "root", voisit nähdä hakemiston "/bin" sisällön, mutta et voisi lisätä siihen tiedostoja. Mutta et ole osa "root"-ryhmää. Ainoa jäljelläoleva vaihtoehto on "muu maailma", ja sinut on sisällytetty siihen, joten voit ainoastaan nähdä hakemiston sisällön.

Katsotaanpa toista tiedostoa:

```
$ ls -l /etc/issue
-rw-r--r-- 1 root root 36 2009-02-26 15:06 /etc/issue
```

Kuten voit nähdä, se on tavallinen tiedosto, jota "root" voi lukea ja kirjoittaa ja jota käyttäjät "root"-ryhmässä, missäpä he ovatkin, voivat vain lukea. Ja sitä sinä, "joe", voit myös vain lukea.

Entäpä omat juttusi? Todennäköisesti sinulla on "Työpöytä" tai "Desktop" -hakemisto kotihakemistossasi. Tarkastamme sen käyttöluvat komennolla "ls -l", ja liitämme lisävalitsimen "-d", jotta näemme vain hakemiston "Desktop" rivin, emmekä sen alla olevia tiedostoja ja hakemistoja.

```
$ ls -l -d ~/Desktop
drwxr-xr-x 8 joe joe 4096 2009-03-12 09:27 /home/joe/Desktop
```

Se hakemisto kuuluu sinulle! Ja käyttölupien mukaan voit lukea sisältöä ja laittaa sinne tiedostoja. Muut ihmiset voivat vain katsoa sen sisältöä.

KÄYTTÖLUPIEN ASETTAMINEN KOMENNOLLA CHMOD

Jos tahdot muuttaa tiedoston käyttölupia sinun täytyy omistaa se - et voi vain mennä muuttelemaan muiden ihmisten juttuja. Jos omistat tiedoston (tai hakemiston), voit muuttaa sen käyttölupia komennolla "chmod". On kaksi tapaa määritellä uuden tiedoston käyttöluvat ja molemmissa on hyvät puolensa. Tarkastellaampa molempia.

Luo harjoitushakemisto ja kopioi sinne pari tiedostoa:

```
$ mkdir ~/harjoitus
$ cd ~/harjoitus
$ cp /etc/issue /etc/motd .
$ ls -l
 total 8
-rw-r--r-- 1 joe joe  36 2009-03-21 14:34 issue
-rw-r--r-- 1 joe joe 354 2009-03-21 14:34 motd
```

Sanotaanpa, että tehdot tehdä tiedostosta "issue" luettavan ja kirjoitettavan ryhmällesi ja itsellesi. Tiedostosta "motd" tahdot tehdä luettavan ja kirjoitettavan vain itsellesi. Tämä merkitsee, että viimeisen ulostulon täytyy näyttää suunnilleen tältä:

```
$ ls -l
 total 8
-rw-rw---- 1 joe joe  36 2009-03-21 14:34 issue
-rw------- 1 joe joe 354 2009-03-21 14:34 motd
```

Voit tehdä näin tiedostolle "issue":

```
$ chmod u=rw,g=rw,o= issue
```

Tämä merkitsee:

- "u=rw": aseta käyttäjän käyttöoikeuksiksi lue ja kirjoita
- "g=rw": aseta ryhmän käyttöoikeuksiksi lue ja kirjoita
- "o=": älä anna muille mitään lupia

Tiedostolle "motd" komento menee näin:

```
$ chmod u=rw,g=,o= motd
```

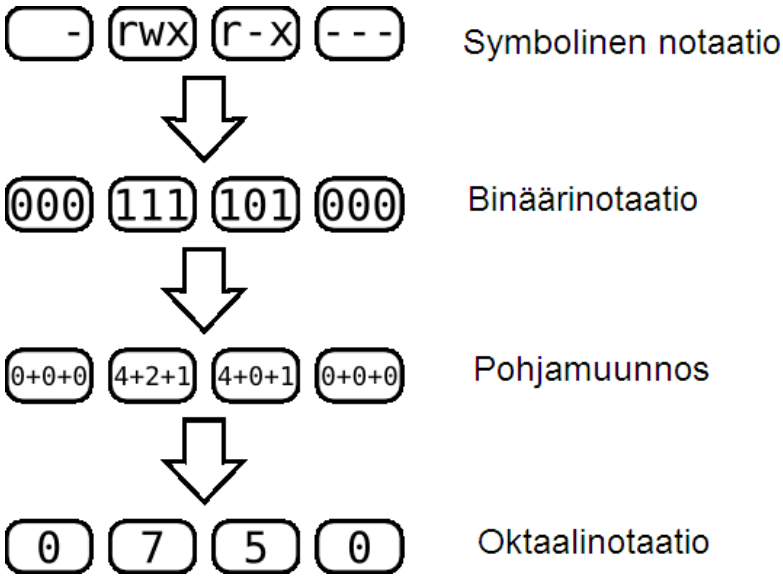
Melko suoraviivaista, vai mitä? Myös paljon kirjoittamista. Lyhempi versio menisi näin:

```
$ chmod ug=rw,o= issue
$ chmod u=rw,go= motd
```

Se on hieman lyhempi, mutta on olemassa vieläkin lyhempi versio:

```
$ chmod 0660 issue
```

Tätä tarvitsee hieman selittää. Numerot merkitsevät samoja käyttölupia kuin yllä. Jos tahdot ymmärtää sen toiminnan, ajattele seuraavaa kaaviota:



Ylin rivi näyttää päämäärämme: tiedoston, jonka omistaja voi lukea, kirjoittaa ja suorittaa, jota ryhmä voi vain lukea ja suorittaa, ja jota muu maailma ei voi käyttää mitenkään. Jokainen kirjain symbolisessa notaatiossa vastaa bittiä binäärisessä esityksessä. Jos kirjain on paikalla, numero on "1", ja jos sitä ei ole, numero on "0". Ensimmäinen "1" luvussa "111" on "4", toinen on "2" ja kolmas on "1". Kun lisäät nämä yhteen, saat numeron "7". Kun teet saman kaikille muille kolmikoille, saat luvun "0750".

Palataksemme edelliseen esimerkkiin, tiedostoon "issue," tahdomme lupien olevan "-rw-rw----", tämä antaa meille numerot "0", "4+2", "4+2" ja "0", josta tulee "0660". Voitko selvittää, mitä tämä on tiedostolle "motd"?

14. INTERAKTIIVINEN EDITOINTI

Monet ihmiset, varsinkin aloittelijat, käyttävät nuolinäppäimiä siirtyäkseen ympäriinsä komentorivillä. Monet ihmiset voivat tehdä niin paljon nopeammin kuin kestäisi muistella monimutkaisempia mutta tehokkaampia vaihtoehtoja. Monet näistä menetelmistä ovat oppimisen arvoisia, joten esittelemme ne tässä.

Komentotulkissa on kaksi erilaista valikoimaa *näppäinkomentoja*, jotka ovat inspiroineet kaksi tehokasta tekstieditoria, Emacs ja vi (luultavasti kaksi tehokkainta olemassaolevaa tekstieditoria). Käyttämällä näppäinkomentoja komentorivivelhot voivat kirjoittaa ja muokata jopa pitkiä komentorivejä sekunnin murto-osassa. Jos opettelet komentotulkin tarjoamia näppäinkomentoja, vaikka ne näyttäisivätkin aluksi epäkäytännöllisiltä, voit pian tehdä niin.

Huomaa: Voit käyttää Emacsin ja vi:n näppäinkomentoja ainoastaan, jos osaat kirjoittaa kymmensormijärjestelmällä. Jo et osaa, kannattaa opetella mahdollisimman nopeasti. Se on ehdottomasti sen arvoista.

Oletusarvoisesti BASH-komentotulkki käyttää Emacsin näppäinkomentoja. Jos tahdot käyttää vi:n näppäinkomentoja, kirjoita seuraava komento:

```
$ set -o vi
```

Voit siirtyä takaisin Emacsin näppäinkomentoihin kirjoittamalla:

```
$ set -o emacs
```

Emacsin ja vi:n näppäinkomennot ovat ihan erilaisia, molempiin tottuminen vie aikaa. Sinun kannattaisi kokeilla molempia, jotta löytäisit itsellesi paremmin sopivan vaihtoehdon. Tämä kappale käsittelee oletusarvoisesti käytössä olevia Emacsin näppäinkomentoja. Jos opettelet vi:tä, voit siirtyä niihin näppäinkomentoihin, ja ne tuntuvat hyvin intuitiivisilta.

Vihje: Älä yritä opetella kaikkia näppäinkomentoja kerralla! Ihmisen aivoja ei ole suunniteltu sellaisiin asioihin, joten unohdat melkein kaikki niistä. Sen sijaan kannattaa opetella 4-5 käyttökelpoisimmalta vaikuttavaa näppäinkomentoa ja käyttää niitä säännöllisesti - oppia tekemällä. Myöhemmin voit palata tähän lukuun ja opetella lisää näppäinkomentoja. Pian pyörit vauhdilla komentorivillä!

EMACSIN NÄPPÄINKOMENNOT

Emacsin näppäinkomennoissa käytetään paljon **Ctrl** ja **Alt** -näppäimiä. Kokeneet Emacsin käyttäjät uudelleenkartoittavat usein **CapsLock** -näppäimensä näppäimeksi **Ctrl**, jotta voivat kirjoittaa Emacs-komentoja mukavammin (ja välttääkseen rasitusvammoja!). Kun aloitat Emacsin näppäinkomentojen säännöllisen käytön, neuvomme sinua tekemään samoin.

Liikkuminen ympäriinsä

Kaksi perusnäppäinkomentoa liikkumiseen ympäriinsä komentorivillä Emacs-tilassa ovat **Ctrl + f** ja **Ctrl + b**. Ne siirtävät kursoria yhden merkin oikealle tai vasemmalle:

Ctrl + f	Liiku eteenpäin yksi merkki
Ctrl + b	Liiku taaksepäin yksi merkki

Tietenkin voit tehdä samat kursorinliikkeet käyttäen nuolinäppäimiä näppäimistölläsi. Kuten yllä mainittiin, voit käyttää Emacsin näppäinkomentoja **Ctrl + f** ja **Ctrl + b** ollaksesi tehokkaampi, sillä käsiesi ei tarvitse lähteä näppäimistön kirjainosasta. Et välttämättä huomaa nopeuseroa heti (varsinkaan jos et ole vielä nopea kirjoittaja), mutta kun saat enemmän kokemusta komentorivin käytöstä, et varmasti tahdo enää koskea nuolinäppäimiin!

Seuraava taulukko listaa joitain näppäinkomentoja, joiden avulla voit navigoida komentorivillä vielä nopeammin:

Alt + f	Liiku yksi sana eteenpäin
Alt + b	Liiku yksi sana taaksepäin
Ctrl + a	Liiku rivin alkuun
Ctrl + e	Liiku rivin loppuun

Käyttämällä ylläolevassa taulukossa esiteltyjä näppäinkomentoja voit nopeuttaa komentorivieditointia dramaattisesti. Jos esimerkiksi kirjoitat väärin hyvin pitkän tiedostonimen, näppäinkomento **Alt + b** vie kursorin takaisin sanan alkuun - hankalien merkki kerrallaan tapahtumien siirtymisten tekeminen nuolinäppäimillä on tarpeetonta. Näppäimet **home** ja **end** ovat vaihtoehto näppäinkomennoille **Ctrl + a** ja **Ctrl + e**.

Tekstin editoiminen

Kaksi yleisimmin käytettyä muokkauskomentoa ovat seuraavat:

Ctrl+t	Vaihda kursoria ennen oleva merkki ja kursorin alla/jälkeen oleva merkki
Alt+t	Vaihda kursorin jälkeen oleva sana ja kursorin alla/jälkeen oleva sana

Näihin kahteen komentoon menee jonkin aikaa tottua, mutta molemmat ovat erittäin käyttökelposia. Komennon **Ctrl + t** päätoiminto on korjata kirjoitusvirheitä, komentoa **Alt + t** käytettiin usein "vetämään" sanaa eteenpäin komentorivillä. Katso seuraavaa komentoriviä (alleviivaus merkitsee kursorin sijaintia):

```
$ echo yksi_kaksi kolme neljä
```

Jos painat tässä tilanteessa näppäimiä **Alt + t**, sana ennen kursoria ("yksi") vaihtuu sanaksi kursorin jälkeen ("kaksi"). Kokeile! Tulos näyttää tältä:

```
$ echo kaksi yksi_kolme neljä
```

Voit huomata kaksi asiaa. Ensinnäkin sanojen "yksi" ja "kaksi" järjestys on käännetty. Toiseksi kursori on siirtynyt eteenpäin sanan "yksi" mukana. Nyt kursorin siirtymisen hyvä puoli on se, että voit painaa **Alt + t** vielä kerran, jotta "yksi" vaihtaa paikkaa seuraavan sanan "kolme" kanssa:

```
$ echo kaksi kolme yksi_neljä
```

Painamalla näppäimiä **Alt + t** toistuvasti, voit "vetää" sanaa eteenpäin, kunnes se on komentorivin lopussa. (Voit tietenkin tehdä saman yksittäiselle kirjaimelle käyttäen näppäimiä **Ctrl + t**.)

Ensin näiden kahden paikanvaihtokomennon toiminta voi vaikuttaa hieman hämmentävältä. Leiki niillä jonkin aikaa, ja totut siihen.

Tekstin poistaminen ja palauttaminen

Tässä on näppäriä komentoja tekstin poistamiseen:

Ctrl + d	Poistaa merkin kursorin alta
Alt + d	Poistaa kaiken tekstin kursorista nykyisen sanan loppuun
Alt + askelpalautin	Poistaa kaiken tekstin kursorista nykyisen sanan alkuun asti

Komennot **Alt + d** ja **Alt + askelpalautin** eivät poista tekstiä, vaan tappavat sen. Tappaminen eroaa poistamisesta siinä, että poistettu teksti on kadonnut pysyvästi, mutta tapetun tekstin voi palauttaa henkiin käyttämällä seuraavaa komentoa:

Ctrl + y	Palauttaa aiemmin tapetun tekstin

Voimme nähdä esimerkin avulla miten tämä toimii:

```
$ echo yksi kaksi
```

Kursorin sijainnin osoittaa taas alaviiva. Jos painat tässä tilanteessa **Alt + askelpalautin**, sana "kaksi" ja sen jälkeen olevat välilyönnit tapetaan, jolloin komentorivi näyttää tältä:

```
$ echo yksi
```

Jos nyt painat näppäimiä **Ctrl + y**, tapettu teksti palautetaan takaisin komentoriville. Voit tehdä tämän monta kertaa. Jos painat **Ctrl + y** kolme kertaa, näet seuraavan rivin:

```
$ echo yksi kaksi kaksi kaksi
```

Kuten voit nähdä, tekstin tappaminen on paljolti samanlainen kuin leikkaustoiminto useimmissa nykyaikaisissa tekstinkäsittelyohjelmissa. Jos tekstiä ei tapeta vaan poistetaan (painamalla **Ctrl + d**), sitä ei voida palauttaa komentoriville. Ainoa tapa palauttaa se on käyttää peruuta-toimintoa, joka esitellään alla.

Seuraavassa on luultavasti tehokkaimmat komennot tekstin tappamiseen:

Ctrl + k	Tapa kaikki teksti kursorista rivin loppuun
Ctrl + u	Tapa kaikki teksti kursorista rivin alkuun

Kuten tavallista, paras tapa oppia nämä komennot on kokeilla niillä. Huomaat, että pitkien tekstinpätkien tappaminen, ja milloin tarpeellista, palauttaminen, voi säästää paljon aikaa.

Muutosten peruuttaminen

Voit peruuttaa viimeisen tekemäsi muutoksen seuraavalla komennolla:

Ctrl + _	Peruuta viimeinen muutos

Vaihtoehtoinen tapa tehdä sama asia on painaa näppäimiä **Ctrl + xu**. (Paina **x** ja **u** vuorotellen, kun pidät näppäintä **Ctrl** pohjassa.)

Navigoi komentotulkin historiassa

Komentotulkki tallentaa viimeksi antamasi komennot. Tämä mahdollistaa aiemmin kirjoitettujen komentojen palauttamisen, jolloin voit välttyä kirjoittamasta niitä uudelleen. Tässä on tärkeimpiä komentoja komentotulkin komentohistoriassa suunnistamiseen:

Ctrl + p	Mene edelliseen komentoon historiassa
Ctrl + n	Mene seuraavaan komentoon historiassa
Ctrl + >	Mene historian loppuun
Ctrl + r	Etsi historiasta aiemmin annettuja komentoja
Ctrl + g	Peruuta hakeminen historiasta

Katsotaanpa näiden komentojen toimintaa yksinkertaisen esimerkin avulla. Avaa komentotulkki ja kirjoita seuraavat komennot:

```
$ echo kaksi
kaksi
$ echo kolme
kolme
$ echo neljä
```

```
neljä
```

Kun olet kirjoittanut nämä komennot, sinulle jää tyhjä komentorivi odottamaan syötettäsi. Paina nyt näppäimiä **Ctrl + p**. Voit nähdä, että viimeksi kirjoitettu komento näkyy komentorivilläsi: `"echo neljä"`. Jos painat uudestaan **Ctrl + p**, siirryt pidemmälle "ylöspäin" komentohistoriassa, jolloin komentorivillä näkyy `"echo kolme"`. Paina nyt **Ctrl + n** ja näet tulleesi takaisin komentoon `"echo neljä"`: **Ctrl + n** toimii aivan kuin **Ctrl + p**, mutta toisin päin. Nuolinäppäimet **ylös** ja **alas** ovat vaihtoehdot näille.

Painettuasi näppäimiä **Ctrl + p** ja kenties **Ctrl + n** muutaman kerran, voit mennä takaisin komentoriville, jota kirjoitit ennen kuin aloitit historiassa navigoimisen. Voit tehdä tämän painamalla **Ctrl + >**.

Kuten voit havaita, komentorivin historia on vain pitkä lista viimeaikaisia komentoja. Voit siirtyä ylös ja alas listalla painamalla **Ctrl + p** ja **Ctrl + n**. Ja voit painaa näppäintä **Enter** enter milloin tahansa suorittaaksesi valitun komentorivin.

Koska komentorivin historia on vain pitkä lista, se on myös etsittävissä. Tämä tehdään useimmiten komennolla **Ctrl + R**. Oletetaanpa, että olet antanut komennot `"echo kaksi"`, `"echo kolme"` ja `"echo neljä"`. Yritä painaa **Ctrl + R** nyt. Voit nähdä, että uusi komentokehote ilmestyy, ja se sanoo "reverse-i-search". Jos kirjoitat kirjaimen "k", hyppäät välittömästi historian viimeiselle komentoriville, joka sisältää kirjaimen "k", joka on tietenkin "echo kolme". Sieltä voit käyttää komentoja **Ctrl + p** ja **Ctrl + n** navigoidaksesi historiassa, kuten aiemmin on mainittu. Tai voit muokata hakua kirjoittamalla toisen kirjaimen, esimerkiksi "a". Silloin hyppäät komentoon "echo kaksi", koska se on viimeisin komento, joka sisältää kirjaimet "a". Tai voit vain peruuttaa haun painamalla **Ctrl + g**.

Jos olet hieman eksyksissä käyttäessäsi komentotulkin historiatoimintoja, älä huolestu! Jos harjoittelet, opit pian siirtymään rutiinimaisesti edestakaisin komentotulkin historiassa, jolloin vältät pitkien komentorivien kirjoittamisen uudestaan.

INTERAKTIIVINEN EDITOINTI: ESIMERKKI

Seuraava esimerkki näyttää sinulle, kuinka interaktiivinen editointi nopeuttaa valtavasti työtäsi. Oletetaanpa, että olet kirjoittanut seuraavan komentorivin:

```
$ echoo ne two three
bash: echoo: command not found
```

Bash heittää virheen, koska komentoa "echoo" ei ole olemassa. Tarkoitit tietenkin komentoa "echo one two three". Ehkä yllätyt, että tarvitaan vain viisi näppäinkomentoa korjaamaan tämä virhe:

1. Paina **Ctrl + p** saadaksesi edellisen komentorivin ruudulle, nimittäin väärin kirjoitetun komentorivin.
2. Paina **Ctrl + a** siirtääksesi kursorin rivin alkuun.
3. Paina **Alt + f** siirtääksesi kursorin yhden sanan eteenpäin. Kursori on nyt väärin kirjoitettujen sanojen "echoo" ja "ne" välissä.
4. Paina **Ctrl + t**. Näet että kursoria edeltävä "o" ja kursorin alla oleva välivyönti on vaihdettu keskenään: "echoo ne" on nyt "echo one".
5. Paina lopuksi **Enter** suorittaaksesi korjatun komentorivin.

15. LOPPUTILA

Kun kirjoitat komentoja, voit yleensä sanoa, toimivatko ne vai eivät. Komennot, jotka eivät voi toteuttaa pyyntöäsi, tulostavat yleensä virheviestin. Tämä riittää, jos kirjoitat jokaisen komennon käsin ja katsot ulostuloa, mutta joskus (jos vaikkapa kirjoitat skriptiä) tahdot komentojesi reagoivan eri tavalla, kun komento epäonnistuu.

Tämän mahdollistamiseksi komento palauttaa *lopputilan*. Lopputilaa ei normaalisti näytetä, sen sijaan se sijoitetaan muuttujaan (nimettyyn muistipaikkaan), jonka nimi on "$?". Lopputila on numero väliltä 0 ja 255; nolla merkitsee onnistumista, mikä tahansa muu arvo merkitsee epäonnistumista.

Yksi tapa nähdä lopputila on käyttää komentoa "echo" sen näyttämiseen:

```
$ echo "tämä toimii hyvin"
tämä toimii hyvin
$ echo $?
0
$ hhhhhh
bash: hhhhhh: command not found
$ echo $?
127
```

Katsotaanpa virheidenhallinnan eri tapoja.

IF/THEN

Virheen hallinta on tapa tehdä jotain ehdollisesti: jos (*if*) jotain tapahtuu, silloin (*then*) tahdot tehdä toimenpiteitä. Komentotulkki tarjoaa yhdistelmäkomennon - muita komentoja ajavan komennon - jota kutsutaan nimellä "if". Perusmuoto on:

```
if
 <komento>
then
 <komennot-jos-onnistuu>
fi
```

Aloitamme perusesimerkillä, jonka jälkeen parantelemme sitä tehdäksemme siitä käyttökelpoisemman. Kun olemme kirjoittaneet "if" ja painaneet näppäintä **Enter**, komentotulkki tietää, että olemme yhdistelmäkomennon keskellä, joten se näyttää erilaisen komentokehotteen (**>**) muistuttamaan meitä siitä.

```
$ if
> man ls
> then
> echo "Nyt tiedät enemmän komennosta ls"
> fi
Komennon ls manuaalisivu vierii ohi...
Nyt tiedät enemmän komennosta ls
```

Tämän komennon ajaminen tuo esille komennon "ls" opassivun. Näppäimistöllä "q" poistuttaessa "man"-komento poistuu onnistuneesti ja komento "echo" suoritetaan.

Komennon epäonnistumisen käsittely

Komentoon voidaan lisätä "else" (muuten) tarkentamaan, mitä komennon epäonnistuessa pitää tehdä:

```
if
 <komento>
then
 <komento-jos-onnistuu>
else
 <komento-joka-epäonnistuu>
fi
```

Ajetaanpa komento "apropos", jos komento "man" epäonnistuu.

```
$ if
> man draw
> then
> echo "Nyt tiedät enemmän komennosta draw"
> else
> apropos draw
> fi
... lista komennon apropos draw tuloksista ...
```

Tällä kertaa "man"-komento epäonnistui, koska komentoa "draw" ei ole olemassa, ja se aktivoi komennon "else".

&& JA ||

Rakenne "if-then" on erittäin käyttökelpoinen, mutta melko monisanainen riippuvien komentojen yhdistämiseen. Operaattorit "&&" *(ja)* ja "||" *(tai)* tarjoavat vähemmän tilaa vievän vaihtoehdon.

```
komento1 && komento2 [&& komento3]...
```

Operaattori "&&" yhdistää kaksi komentoa yhteen. Toinen komento toimii vain, mikäli ensimmäisen poistumistila on nolla, eli komennon onnistuttua. Useampaa kappaletta operaattoria "&&" voidaan käyttää samalla rivillä.

```
$ mkdir mylogs && cd mylogs && touch mail.log && chmod 0660
mail.log
```

Tässä on esimerkki useammista komennoista, joista jokainen olettaa edellisen komennon toimineen onnistuneesti. Jos olisimme käyttäneet tämän tekemiseen rakennetta "if"-then, olisimme ehkä päätyneet näpertelemään suurella joukolla "if" ja "then" -komentoja.

Huomaa, että operaattori "&&" menee *oikosulkuun*, eli yhden komennon epäonnistuessa sen jälkeen tulevia komentoja ei ajeta. Käytämme tätä ominaisuutta hyväksemme estääksemme epätoivotut vaikutukset (kuten tiedoston "*mail.log*" luominen väärään hakemistoon edellisessä esimerkissä).

Jos "&&" vastaa komentoa "then", operaattori "||" vastaa komentoa "else". Se tarjoaa mekanismin sen määrittelyyn, mikä komento suoritetaan, mikäli ensimmäinen komento epäonnistuu.

```
komento1 || komento2 || komento3 || ...
```

Jokainen komento ketjussa toimii vain, mikäli edellinen komento ei onnistunut (sen poistumistila oli jotain muuta kuin nolla).

```
$ cd Työpöytä || mkdir Työpöytä || echo "Työpöytähakemistoa
ei löytynyt ja sitä ei voitu luoda"
```

Tässä esimerkissä yritämme mennä hakemistoon "*Työpöytä*", jonka epäonnistuttua luomme sen, ja jos luominen epäonnistuu, tiedotamme asiasta käyttäjälle virheviestillä.

Tällä tiedolla voimme luoda tehokkaan ja tiiviin "helpme" -funktion. Edellisissä esimerkeissämme olemme näyttäneet kaksi eristyksissä käytettyä operaattoria, mutta ne voidaan myös sekoittaa keskenään.

```
$ function helpme() { man $1 && echo "nyt tiedät enemmän
aiheesta $1" || apropos $1 }
```

Kuten varmaan huomasit, "echo "nyt tiedät enemmän..."" ei ole kauhean hyödyllinen komento. (Se ei välttämättä ole edes kovin paikkansapitävä, sillä ehkä komento "man" esitteli niin paljon erilaisia valitsimia, että käyttäjä meni vain sekaisin). Nyt yksinkertaistamme funktion muotoon:

```
$ function helpme() { man $1 || apropos $1 }
```

Toivottavasti poistut tästä luvusta nollan poistumistilalla!

16. ALIKOMENNOT

Voit suorittaa komentotulkissa yhden komennon toisen komennon sisällä. Tässä on yksinkertainen esimerkki:

```
grep `date +%b` apache_error_log
```

Tarkemerkki (`) on yleensä samassa näppäimessä kuin tilde (~).

Komento tarkemerkkien "``" sisällä suoritetaan ensin. Ulostulo laitetaan sen jälkeen isomman komennon sisään. Niinpä komentotulkki suorittaa ensin komennon:

```
date +%b
```

Tämä on komento "date" parametrillä, joka alkaa plus-merkillä "+", jolla osoitetaan ulostuloon haluttu formaatti. Formaatti "+%b" on melko outo tapa pyytää "date" -komentoa tulostamaan vain kolmen kirjaimen lyhenne nykyisestä kuukaudesta. Jos esimerkiksi ajamme tämän komennon maaliskuussa, se tulostaa:

```
Mar
```

Kolmikirjaiminen lyhenne nykyisestä kuukaudesta siis asetetaan ympäröivän "grep" -komennon sisään. Maaliskuussa tämä komento on siis muodossa:

```
grep Mar apache_error_log
```

Lokitiedosto *"apache_error_log"* tallentaa viestit päivämäärillä ja päivämäärä sisältää kuukauden kolmikirjaimisen lyhenteen:

```
[Mon Mar 09 14:44:23 2009] [notice] Apache/2.0.59 (Ubuntu) PHP/5.2.6
DAV/2 configured
```

Mitä tämä komento siis tekee? Se näyttää kaikki nykyisen kuukauden aikana kirjatut lokiviestit tiedostosta *"apache_error_log"*. Tietenkin lokitiedostossa voi olla useamman vuoden viestit, jolloin saat myös edellisen vuoden saman kuukauden viestit. Laittamalla komennon "date" komennon "grep" sisään olemme luoneet komennon, jonka voimme tallentaa ja suorittaa milloin tahansa määrittelemättä oikeaa kuukautta. Voimme esimerkiksi tallentaa tämän aloitustiedostoon *".bashrc"*:

```
alias monthlog="grep `date +%b` apache_error_log"
```

Nyt meillä on oma komentomme, "monthlog", joka näyttää nykyiset Apachen lokitiedostot.

Bashissa voit tehdä saman asian syntaksilla, joka on monista yksinkertaisempi:

```
grep $(date +%b) apache_error_log
```

Tarkemerkkien sijasta voit laittaa dollarimerkin ja pistää komennon sulkujen sisään.

Komennon korvaus on kuin putki (merkki "|"). Komennon korvaus on kuitenkin joustavampi kuin putki, sillä voit laittaa yhden komennon mihin tahansa toisen komennon sisälle. On olemassa yksi ero: putki antaa molempien komentojen toimia samaan aikaan. Jos komennon sisään laitettu komento kestää kauan, ulompi komento ei toimi ennen kuin sisempi komento on suoritettu.

Jos sisempi komento voi tuottaa ulostuloa, joka on enemmän kuin yksi sana (kuten "Mar 09"), voit lähettää sen yhtenä argumenttina laittamalla komennon lainausmerkkeihin. Komento `"grep"` tässä osassa vaatii merkkijonon lähettämistä yhtenä parametrinä.

17. LIIKUTAAN TAAS

Tähän asti olet varmaankin jo käyttänyt "cd" -komentoa ja "pwd" -komentoa saadaksesi selville nykyisen työhakemistosi. Kun työskentelet komentorivin parissa jonkin aikaa, huomaat vaihtavasi hakemistoa koko ajan. Tämän helpottamiseksi Bash tarjoaa "hakemistopakan", jota voit käyttää liikkuaksesi nopeasti hakemistoissa, joissa teet työtä. (Näytämme pian esimerkkejä, jotka auttavat ymmärtämään "pakan" idean.) Sinulla on käytössäsi seuraavat komennot:

Komento	Toiminto
dirs	Näyttää hakemistopakan, päällimmäinen taso ensin (vasemmalla); muut komennot tekevät tämän päätoimintonsa jälkeen. Kaikkia komentovalitsimia ei näytetä tässä taulukossa.
pushd hakemisto	Työnnä hakemisto pakan päälle ja muuta nykyinen työhakemisto siksi.
pushd	Vaihtaa pakan kaksi ylintä tasoa ja siirtyy uuteen pakan päällimmäiseen tasoon.
pushd **+N**	Kierittää koko pakkaa **vasemmalle N** askelta ja siirtyy pakan uuteen päällimmäiseen tasoon
pushd **-N**	Kierittää koko pakkaa **oikealle N+1** askelta ja siirtyy pakan uuteen päällimmäiseen tasoon
popd	Poistaa pakan täällä olevan hakemiston ja siirtyy uuteen pakan päällimmäiseen tasoon

Jos tarvitset lisää visuaalista apua "pakan" ymmärtämiseen, yksinkertaisin tapa ajatella pakkaa on pitää pakkaa korttipakkana pöydälläsi. Työnnät ("push") uusia kortteja pakan päälle ja poistat ("pop") päällimmäisen kortin pakasta. Molemmat menetelmät toimivat periaatteella viimeinen lisätty kortti on ensimmäinen poistettava kortti.

Voit leikkiä näillä komennoilla ymmärtääksesi niiden toiminnan. Esimerkiksi seuraava taulukko tarjoaa listan komentoja, niiden vaikutuksen nykyiseen työhakemistoon ja niiden vaikutuksen pakkaan.

Komento	Nykyinen työhakemisto komennon jälkeen	Pakka komennon jälkeen
cd	~	~
pushd /	/	/ ~
pushd /usr/bin	/usr/bin	/usr/bin / ~
pushd +1	/	/ ~ /usr/bin

pushd +1	~	~ /usr/bin /
popd	/usr/bin	/usr/bin /
pushd +1	/	/ /usr/bin
popd	/usr/bin	/usr/bin

18. HYÖDYLLISTÄ KUSTOMISOINTIA

Voit todella tehdä komentotulkista omanlaisesti, soveltamalla jokaista ominaisuutta toimintatapaasi (ja jopa erilaisia toimintatapoja joilla toimit viikosta toiseen). Tässä osassa katsomme nopeita muutoksia, joita voit tehdä. Skriptaus on tapa laajentaa ja yhdistää komentotulkin tarjoamia ominaisuuksia ja esitellään myöhemmin.

MUUTTUJAT

Jokaisessa komentotulkissa on *muuttujaksi* kutsuttu konsepti. Muuttujat koostuvat kahdesta osasta: muuttujan *nimestä* ja muuttujan *arvosta*. Jos sanoisin "x=6", "x" on muuttujan nimi ja "6" on sen arvo. Nähdäksesi muuttujan arvon, laitat dollarimerkin muuttujan nimen eteen. Tässä on yksinkertainen esimerkki.

```
$ x=6
$ echo $x
6
$
```

Yllä viimeinen rivi *asettaa* numeron "6" muuttujaan "x" ja toinen rivi pyytää komentotulkkia näyttämään "x":n arvon. Huomaa, että laitamme dollarimerkin muuttujan nimen eteen kun tahdomme nähdä sen arvon, mutta emme koskaan käytä dollarimerkkiä, kun asetamme sille arvon.

Joten mikä tahansa dollarimerkillä ($) alkava tulkitaan komentotulkin taholta muuttujaksi. Yksi muuttuja hiippaili aiempaan lukuun poistumistilasta: huomasit, että "$?" sisältää edellisen komennon poistumistilan.

Mitä muita hyödyllisiä asioita voimme tehdä muuttujilla? Yleinen käyttö on vähentää kirjoittamista. Sanotaanpa, että tiedostot projektiin, jota olet työstänyt koko viikonlopun, on tallennettu hakemistoon, jota kutsutaan nimellä *"/home/jsmith/projects/foo/confoobulator"*. *"/home/jsmith/projects/foo/confoobulator"* on paljon kirjoitettavaa, mutta voit vähentää kirjoittamista tallentamalla arvon muuttujaan.

```
$ p=/home/jsmith/projects/foo/confoobulator
```

Nyt voit siirtyä projektihakemistoosi kirjoittamalla

```
$ cd $p
```

Voit poistaa muuttujan arvon asettamalla sen tyhjäksi merkkijonoksi:

```
$ VAR=""
```

tain käyttämällä komentoa `"unset"`:

```
$ unset VAR
```

TAVALLISET MUUTTUJAT JA YMPÄRISTÖMUUTTUJAT

Useimmat komentotulkit (sisältäen GNU Bash -komentotulkin) tunnistavat kahdenlaisia muuttujia: *tavalliset muuttujat ja ympäristömuuttujat*. Tavallinen muuttuja on tarjolla komentotulkillesi, muttei ohjelmille, joita komentotulkki suorittaa. Sen sijaan ympäristömuuttuja on tarjolla sekä komentotulkille että sen käyttämille komennoille. Tavallisen muuttujan voi muuttaa ympäristömuuttujaksi käyttämällä komentoa `"export"`. Jos kirjoittaisin

```
$ export p
```

Tavallisesta muuttujasta "p" tulee ympäristömuuttuja, jota mikä tahansa komentopäätteen käyttämä komento voi käyttää.

KOMENTOPÄÄTTEEN MUUTTUJAT

Komentopääte tarjoaa listan omista muuttujistaan. Esimerkiksi komennon `"whoami"` (joka näytettiin kirjan alkupäässä) ulostulo on sama kuin "$USER". Kotihakemistosi tallennetaan muuttujaan "$HOME". Voit nähdä minkä tahansa muuttujan laittamalla sen komentoon "echo":

```
$ echo $HOME
```

Esimerkin ensimmäinen dollarimerkki on vain komentokehote, sillä ei ole mitään tekemistä muuttujien kanssa.

Voit nähdä komentotulkin sisäänrakennetut muuttujat (itse asiassa alaryhmä, joka tunnetaan *ympäristömuuttujina*) komennolla:

```
$ env
SHELL=/bin/bash
USER=jsmith
PATH=/usr/local/bin:/usr/bin:/bin:/usr/games
PWD=/home/jsmith
HOME=/home/smith
_=/usr/bin/env
...
```

Ulostulosi näyttää varmasti erilaiselta, mutta monet muuttujien nimet ovat samoja. Jotkin näistä ovat hyödyllisiä myöhemmin.

- "SHELL" on polku komentotulkkiisi.
- "USER" on käyttäjänimesi. Kun kirjauduit GNU/Linux -järjestelmään, tämä on käyttäjänimi, jonka kirjoitit.
- "PATH" on lista hakemistoja, jossa erottimena ovat kaksoispisteet. Kun ajat komennon (kuten `"cat"` tai `"ls"`), komentotulkkisi katsoo näihin hakemistoihin löytääkseen suoritettavan ohjelman. Puhumme enemmän muuttujasta "PATH" hetken päästä.
- "PWD" on nykyinen työhakemistosi (eli kansio, jossa olet).
- "HOME" on kotihakemistosi. Aloitat tästä hakemistosta, kun kirjaudut sisään ensimmäisen kerran.
- "_" on viimeksi suoritettu komento. Tässä tapauksessa "/usr/bin/env".

MUUTTUJIEN LAAJENEMISEN KONTROLLI

Jos laitat muuttujan kiinni muihin kirjaimiin, komentotulkki ei tunnista sitä. Esimerkiksi seuraava ei toimi:

```
$ curr=myfile
$ rm $curr1.jpeg
rm: .jpeg: No such file or
directory
```

Virheviesti voi olla hämmentävä. Tapahtui näin: komentotulkki näki muuttujan nimeltä "$curr1". Kun se ei voinut löytää sellaista muuttujaa, se korvasi sen tyhjällä merkkijonolla. Niinpä yritit loppujen lopuksi suorittaa komennon:

```
$ rm .jpeg
```

Jos tahdot poistaa tiedoston "myfile1.jpeg", käytä se aaltosulkuja muuttujan ympärillä, jotta komentotulkki tietää missä muuttujan nimi loppuu:

```
$ rm ${curr}1.jpeg
```

HAKUPOLKU

Olemme katsoneet monia esimerkkejä komentojen suorittamisesta. Jos kirjoitan "ls -l" komentorivillä, komentotulkkini ajaa komennon "ls", mikä tekee listan tiedostoja. Komento "ls" on itse asiassa ohjelma, joka on tietokoneen kovalevyllä. Voit kysyä komentotulkintasi missä komento asustaa käyttämällä komentoa "which". Jos kirjoitan

```
$ which ls
```

niin päätteeni vastaa hakemistolla "/bin/ls", joka kertoo minulle, että komento "ls" on ohjelma, joka elää kovalevyni hakemistossa "*/bin*". Voimme jopa käyttää komentoa "ls" etsimään itseään

```
$ ls -l /bin/ls
-rwxr-xr-x 1 root root 92672 2007-01-30 15:48 /bin/ls
```

Komentotulkkini löysi komennon "ls" käyttämällä ympäristömuuttujaa "PATH".

```
PATH=/usr/local/bin:/usr/bin:/bin:/usr/games
```

Muuttuja "PATH" on lista hakemistoja, jotka on eroteltu kaksoispisteillä. Kun kirjoitin "ls", komentotulkkini katsoi komentoja hakemistossa *"/usr/local/bin/ls"*, sitten "*/usr/bin/ls*", ja lopulta *"/bin/ls"*. *"/bin/ls"* on komennon sijaintipaikka, joten komentotulkkini pystyi ajamaan sen. Jos ei olisi ollut hakemistoa, komentotulkkini olisi kokeillut hakemistoa "*/usr/games/ls*", ja sitten lopettanut.

KONFIGURAATIOTIEDOSTOT

Olet ehkä nähnyt monta siistiä kustomisaatiota kirjassa - tai ehkä ajatellut muutamia omia kustomisaatioitasi - ja olet ehkä valmis tallentamaan niistä muutamia, jotta voit käyttää niitä uudelleen jokaisessa pääteistunnossa. Kaikki, mitä määrittelet päätteessä, on menetetty, kun suljet pääteikkunan. Joten nyt on hyvä aika katsoa konfiguraatiotiedostoja, jotka tallentavat käyttökelpoiset kustomisaatiot istuntojen välillä.

Kotihakemistosi sisältää useita kätkettyjä tiedostoja, jotka sisältävät asetukset päätteelle ja muille ohjelmille. Lisäksi on kokonaisia kätkettyjä hakemistoja, joihin ohjelmat tallentavat tietoa, kuten värit, jotka päätit laittaa työpöydällesi.

Kuinka nämä hakemistot on kätketty? Yksinkertaisen käytännön kautta: mikä tahansa pisteellä (.) alkava tiedosto on kätketty. Tiedostonhallintaohjelma työpöydälläsi ei näytä sinulle kätkettyjä tiedostoja, ellet valitse erikoisvalitsinta, jolla näet kätketyt tiedostot. Vastaavasti pääte ei näytä niitä oletusarvoisesti "`ls`"-komentoa. Näyttäksesi ne päätteessä, lisää valitsin "`-a`", joka näyttää kaikki tiedostot:

```
$ ls -a
.
..
.bash_history
.bash_logout
.bashrc
.irssi
.profile
foo
examplefile
```

Edellisessä listauksessa (joka näyttää erilaiselta omassa järjestelmässäsi) tiedostot "*.bashrc*" ja "*.profile*" ovat erityisen mielenkiintomme kohteina. Tiedosto "*.bashrc*" liittyy tiettyyn komentotulkin tyyppiin (komentotulkkeja on monia erilaisia), jota kutsutaan Bashiksi, kun taas "*.profile*"-tiedoston lukevat muutkin komentotulkit, mikäli päätä käyttää jotain muutakin kuin Bashia.

Bash-konfiguraatio toimii hyvin yksinkertaisella tavalla: Bash vain suorittaa komennot käynnistyessään, täsmälleen kuin olisit kirjoittanut ne ennen minkään muun tekemistä. Niinpä mikä tahansa, mitä näet tässä osassa, ja mistä pidät - alias, funktio, muutos ympäristömuuttujaan jne. - sopii laitettavaksi konfiguraatiotiedostoon. Kokonaisia skriptejä voidaan lisätä.

Aloitustiedostoissasi on luultavasti komentoja jo valmiiksi. Jotkin on asennettu yhdessä käyttöjärjestelmän kanssa, kun taas toiset ovat työpaikkojen ylläpitäjien lisäämiä. Muuttaaksesi näitä kustomisaatioita tai lisätäksesi omiasi, katso tekstieditoreita käsittelevä osa tästä kirjasta. Valitse yksi tekstieditori ja opettele noin tusina sen peruskomentoja, jotta voit tehdä minimaalisen editoinnin, jota tarvitaan omien kustomisaatioiden tekemiseen.

FUNKTIOT

Voit yhdistää joukon komentoja ja antaa sille nimen; voit käyttää tätä nimeä kuin mitä tahansa muuta komentoa. Harkitse funktion kirjoittamista aina, kun käytät samoja komentoja toistuvasti. Voit myös kirjoittaa taipuisan funktion kirjoittamista, jolloin funktion toiminta perustuu parametreihin, aivan kuin muidenkin komentojen toiminta.

Yksinkertaisena esimerkkinä, oletetaan, että tahdot tallentaa tiedon hakemistoon joka päivä:

```
echo ENTRY -------------- >>~/save/log
date >>~/save/log
du -c >>~/save/log
ls -R >>~/save/log
echo >>~/save/log
```

Tallentaaksesi komentosi funktioon, anna komento nimeltä "function", jota seuraa nimi, jonka tahdot antaa sille, ja komennot aaltosulkujen sisällä. Huomaa, että olemme käyttäneet ristikkomerkkejä (#) lisätäksemme joitain kommentteja, jotta muistat myöhemmin, mitä varten funktio on tehty. Komentotulkki jättää ristikkomerkin ja sitä seuraavan tekstin huomiotta.

```
function savelog {
# Lisää tietoa tästä hakemistosta lokitiedostoon, ~/save/log
  echo ENTRY -------------- >>~/save/log
  date >>~/save/log
  du -c >>~/save/log # Alihakemistojen koko
  ls -R >>~/save/log # Tiedostojen listaus
  echo >>~/save/log
}
```

Nyt voit käyttää komentoa "savelog" ja suorittaa sisällytetyt komennot. Voit laittaa funktion määritelmän aloitustiedostoon, jotta et enää koskaan joudu kirjoittelemaan määritelmää uudestaan.

Edellinen esimerkki oli ehkä hieman liioiteltu, sillä harvoinpa kirjoitat samat komennot peräkkäin. Voit kuitenkin usein käyttää monimutkaista komentoa, jonka ajat eri tiedostoille, tai muille kohteille.

Tässä on esimerkiksi komento, joka näyttää sinulle eron tiedoston nykyisen version ja viimeksi muokatun version välillä, jos muokkaa Emacsilla. Emacs tallentaa tiedostosi vanhan version luomalla uuden tiedoston samalla nimellä, mutta lisää siihen tilden (~). Tässä esimerkissä katsomme eroja tiedoston "*txtfile*" ja varmuuskopion "*txtfile~*" välillä:

```
$ diff txtfile~ txtfile | less
```

Tämä on juuri sopivan monimutkaista (ja yleistä) ollakseen tarpeeksi arvokasta funktioon tallennettavaksi. Tahdot kuitenkin antaa tiedostonimen argumenttina, jotta voit käyttää funktiota mihin tahansa tiedostoon, jota editoit. Määrittele siis parametri "$1", joka on erikoismuuttuja, jota funktio ymmärtää:

```
function d~ {
# Vertaile Emacsin varmuuskopiota nykyiseen versioon.
  diff -u $1~ $1 | less
 }
```

Nyt voit käyttää uutta "d~" -komentoasi mihin tahansa tiedostoon, jolla on varmuuskopio:

```
$ d~ txtfile
```

Kuten voit arvata, funktio voi ottaa korkeintaan yhdeksän parametria, joihin voit viitata nimillä "$1", "$2", aina "$9" asti. Jos tahdot enemmän kuin yhdeksän parametria, voit tallentaa parametrin ja poistaa sen listasta:

```
function manyargs {
  $arg=$1
  shift
  ...
}
```

Ensimmäinen asia, jonka tämä funktio tekee, on tallentaa ensimmäinen parametri omaan "$arg" -muuttujaansa. Komento "shift" poistaa parametrin "$1" ja siirtää kaikki muut tiedostot eteenpäin, jolloin toinen argumentti on nyt "$1". Skriptausta käsittelevässä osassa näet, kuinka parametreja tai muita asioita voi käsitellä yhden kerrallaan silmukoiden avulla.

Jos tahdot siirtää kaikki parametrit komennolle, käytä muotoa "$*". Esimerkiksi seuraava funktio "orth" suorittaa "spell"-hyötyohjelman mille tahansa merkkijonolle, jonka annat sille:

```
function orth () {
  echo $* | spell
}
```

Funktiot voivat sisältää yhdistelmälauseita, kuten if/then-rakenteita. Näyttääksemme kuinka tehokkaita funktioiden ja yhdistelmälauseiden yhdistelmät voivat olla, lisäämme tähän if/then-lauseen, joka näytettiin aiemmin komentojen epäonnistumista käsittelevässä osassa.

```
function helpme() { if man $1 then echo "nyt tiedät enemmän
aiheesta $1" else apropos $1 fi
}
```

Joten seuraava:

```
$ helpme draw
```

vastaa nyt tätä:

```
  if man draw
 then echo "nyt tiedät enemmän aiheesta draw"
 else apropos draw
 fi
```

Kunhan arvaat mitä virheitä tai muita tiloja lopputuloksena on, voit käsitellä ne automaattisesti funktiosa.

TIEDOSTOJEN LÄHTEIDEN LISÄÄMINEN

Jos tämä luku on innostanut sinua kirjoittamaan omia kustomisaatioitasi ja tallentamaan ne tiedostoihin, niin hyvä. Mutta lopulta sinulla on paljon erilaisia funktioita, jotka sijoittuvat eri kategorioihin, ja pidät hämmentävänä pitää niitä kaikkia yhdessä tiedostossa. Tässä vaiheessa voit aloittaa komentojen, muuttujien asetusten ja funktioiden varastoinnin eri tiedostoihin, jotka sopivat eri tarpeisiin, ja lukea ne *".bashrc"* -tiedostoosi tai mihin tahansa muuhun skriptiin. Käytä vain pistettä lukeaksesi tiedoston ja saadaksesi komentotulkin suorittamaan sen sisällön:

```
.  skriptitiedosto
```

On tärkeää laittaa välilyönti pisteen jälkeen, ennen tiedostonimeä.

KEHOTTEIDEN ASETTAMINEN

Kun bash tai joku muu komentotulkki odottaa käyttäjän kirjoittavan komennon, se näyttää komentokehotteen, joka voi olla niin yksinkertainen tai monimutkainen kuin haluat. Minimaalinen komentokehote voisi olla

```
$
```

Oletusarvoinen komentokehote näyttää suunnilleen tältä:

```
user@host:~$
```

Jossa "user" on käyttäjänimi, "host" on tietokoneen nimi, "~" on työhakemisto, lyhenne käyttäjän kotihakemistosta, joka on yleensä *"/home/user"*, ja "$" merkitsee, että nykyinen käyttäjä ei ole pääkäyttäjä (root).

Muutaaksesi komentokehotetta voit antaa uuden arvon muuttujalle "PS1". Tehdäksesi muutoksesta pysyvän voit laittaa tämän muuttujan asetuksen tiedostoon *".profile"*, jonka Bash lukee aina käynnistyessään. Oletusarvo on "\u@\h:\w\$", määritellen käyttäjänimen, tietokoneen, työhakemiston, sekä koristemerkit. Seuraava taulukko kuvailee kentät, jotka voivat näkyä komentokehotteessa, sekä muutamat muut käyttökelpoiset merkit. Komentotulkki voi soittaa päätteen "kelloa", joka on nykyisin yleensä piippaus, se voi sisältää monia rivin alkuun palautusta merkitsevän "\r" -komennon sisältäviä rivejä, se voi sisältää yhdistettyjä päätteenhallinnan jatkoja, jotka alkavat yleensä pakomerkillä. Emme selitä kaikki näitä vaihtoehtoja tässä. Katso lisätietoa oppaasta *Bash Reference Manual*, jonka kirjoittajat ovat Brian Fox ja Chet Ramey. http://www.gnu.org/software/bash/manual/

\a	ASCII-kellomerkki (07)	\d	päivämäärä muodossa "viikonpäivä, kuukausi, päivä" (esim. "Tue May 26")
\]	lopettaa tulostumattomien merkkien jakson	\e	ASCII-pakomerkki (033)
\h	tietokoneen nimi ensimmäiseen pisteeseen '.' asti	\H	tietokoneen nimi
\j	komentotulkin juuri nyt suorittamien töiden määrä	\l	komentotulkin päätelaitteen nimi
\n	uusi rivi	\r	palautus rivin alkuun
\s	päätteen nimi, perusnimi muuttujalle $0 (osa, joka seuraa viimeistä kauttaviivaa)	\t	nykyinen aika 24 tunnin TT:MM:SS-muodossa
\T	nykyinen aika 12 tunnin TT:MM:SS -muodossa	\@	nykyinen aika 12 tunnin aamupäivä/iltapäivä-muodossa
\A	nykyinen aika 24 tunnin TT:MM-muodossa	\u	nykyisen käyttäjän käyttäjänimi
\v	Bashin versio (esim. 2.00)	\V	Bashin julkaisu, versio + korjaustiedoston numero (esim. 2.00.0)
\w	nykyinen työhakemisto	\W	nykyisen työhakemiston perusnimi
\!	tämän komennon numero historiassa	\#	tämän komennon komentonumero
\$	jos voimassa oleva käyttäjän numero on 0 (root), merkki #, muuten merkki $	\nnn	merkki, joka vastaa oktaalinumeroa nnn
\\	kauttaviiva	\[	aloita sarja tulostamattomia merkkejä, joita voitaisiin käyttää asettamaan päätteenhallintajakso komentokehotteeseen

Esimerkki:

```
$ PS1="\a\d, \e[31m\t\r\n\e[0m\u@\h:\w $"
```

aiheuttaisi äänen tietokoneelta, näkyvä kehote olisi

```
Mon Mar 23, 13:47:43
user@host:~ $
```

jossa aika on kirjoitettu punaisella. Tämä käyttää merkkiä "\d" merkitsemään päivää, merkkiä "\[31m" laittamaan päälle punaisen värin, merkkiä "\t" näyttämään ajan, "\e[0m" kääntää punaisen pois päältä, "\r\n" palauttaa kursorin rivin alkuun ja aloittaa uuden rivin, loput on kuin oletuksessa.

Tehdäksesi asioista mielenkiintoisempia voit ajaa komennon kehotteen sisällä laittamalla sen merkkien "[\$( )]" sisään. Tämä esimerkki laskee tiedostojen määrän nykyisessä hakemistossa laskemalla rivit "(`wc -l`)", jotka putkitetaan hakemistolistauksesta "(`ls`)".

```
$ PS1="\u@\h [\$(ls | wc -l)]:\$ "
user@host [3]:$
```

PÄÄKÄYTTÄJÄN OIKEUDET

Jokaisen käyttäjän hakemistossa olevien konfiguraatiotiedostojen lisäksi järjestelmässä on paljon konfiguraatiotiedostoja, jotka hallitsevat järjestelmänlaajuista toimintaa. Joskus on tarpeellista muokata sellaista käsin, käyttäen tekstieditoria. Tässä osassa näytämme, kuin käyttäjälle voi antaa pääkäyttäjän oikeudet, mikä on järjestelmänlaajuinen ilmiö, jota hallitsee tiedosto *"/etc/sudoers"*.

Ei ole välttämättä järkevää muokata tätä tiedostoa tavallisessa tekstieditorissa. Komento `"sudoedit"` tarjoaa turvallisemman tavan muokata konfiguraatiotiedostoja.

```
$ sudoedit /etc/sudoers
```

Tämä tekee tilapäisen kopion tiedostosta ja avaa kopion editorissa. Voit ohittaa oletusarvoisen editorin asettamalla ympäristömuuttujan "VISUAL" tai "EDITOR" arvoksi "vi", "emacs", tai mitä tahansa tahdotkin.

Käyttöluparivit tiedostossa "*/etc/sudoers*" määrittelevät käyttäjän, mitä seuraa tietokoneet, joilla käyttäjä voi käyttää komentoa `"sudo"`, minkä ryhmien jäsenenä käyttäjä voi toimia, ja mitä komentoja käyttäjä voi suorittaa komennolla `"sudo"`.

Yrityksen tai koulun järjestelmänvalvojalla voi olla tämän näköisiä oikeuksia.

```
operator       ALL = DUMPS, KILL, SHUTDOWN, HALT, REBOOT, PRINTING,\
                         sudoedit /etc/printcap, /usr/oper/bin/
```

(Merkki '\' jatkaa käyttölupia seuraavalla rivillä.) Tämä antaa luvan käyttää tiettyä komentojoukkoa, sekä muokata kahta tiettyä konfiguraatiotiedostoa, mutta ei muita. Antaaksesi jollekin oikeuden käyttää mitä tahansa pääkäyttäjän komentoa käyttäen komentoa `"sudo"`, aseta käyttäjänimen käyttölupariviksi:

```
username ALL = (ALL) ALL
```

Tämä antaa sinun muokata mitä tahansa konfiguraatiotiedostoa tietokoneellasi.

LOKALISAATIO

Eri maat käyttävät erilaisia käytäntöjä eri asioissa: kirjoitusmerkit, valuutat, päivämäärien ja aikojen muotoilut, jopa paperin koko ja muoto. Tietokoneita voidaan opastaa käyttämään tiettyä kieltä, sekä valitsemaan kielen tietty versio tietyssä maassa. Tämä kustomoidun tiedon yhdistelmä nähdään komennolla *"locale"*.

Kaikki lokalisaatioasetukset raportoidaan komennolla "`locale`". Esimerkiksi:

```
$ locale
LANG=fi_FI.UTF-8
LC_CTYPE="fi_FI.UTF-8"
LC_NUMERIC="fi_FI.UTF-8"
LC_TIME="fi_FI.UTF-8"
LC_COLLATE="fi_FI.UTF-8"
LC_MONETARY="fi_FI.UTF-8"
LC_MESSAGES="fi_FI.UTF-8"
LC_PAPER="fi_FI.UTF-8"
LC_NAME="fi_FI.UTF-8"
LC_ADDRESS="fi_FI.UTF-8"
LC_TELEPHONE="fi_FI.UTF-8"
LC_MEASUREMENT="fi_FI.UTF-8"
LC_IDENTIFICATION="fi_FI.UTF-8"
LC_ALL=
```

Kieliasetus "LANG fi_**FI.UTF-8**" määrittelee suomen kieleksi, Suomen maaksi, ja merkistön koodaukseksi arvon "Unicode UTF-8". Raha on "euro".

Määrittelet yleensä kielen ja maan, kun asennat käyttöjärjestelmäsi, kaikki ohjelmat mukaanlukien komentopäätteen ottavat nuo arvot käyttöön. Alunperin oli oletettu, että kieli, maa ja kirjainkoodaus olisivat yhdessä, mutta globalisoituvasssa maailmassa voi tapahtua niin, että Yhdysvalloissa oleskeleva unkarilainen voisi valita käyttöön merkistökoodauksen UTF-8, ranskankielen, metrisen (SI) mittajärjestelmän, rahayksikön euro, Sveitsin osoite- ja puhelinnumeromuotoilut ja Yhdysvaltojen letter-paperikoon.

Voit muuttaa kaikkia näitä asetuksia päätteessäsi asettamalla sopivan merkkijonon asiaankuuluvaan ympäristömuuttujaan. Hyväksytyt arvot lokalisointiasetuksille asetetaan komennon "locale" valitsimilla.

```
$ locale -m # saatavilla olevat merkistöt
```

ANSI_X3.110-1983 ANSI_X3.4-1968 ARMSCII-8 ASMO_449 BIG5 BIG5-HKSCS ... # Ubuntu 8.10 tarjoaa 226 vaihtoehtoa

$ **locale -a # listaa koneella saatavilla olevat englannin- ja suomenkieliset lokalisaatiot**

en_AU.utf8 en_BW.utf8 en_CA.utf8 en_DK.utf8 en_GB.utf8 en_HK.utf8 en_IE.utf8 en_IN en_NG en_NZ.utf8 en_PH.utf8 en_SG.utf8 en_US.utf8 en_ZA.utf8 en_ZW.utf8 fi_FI.utf8 POSIX

Saat erilaisia sijaintispesifikaatioita riippuen kielistä ja merkistökoodauksista, joita on valittu järjestelmääsi asennusaikaan tai muutettu myöhemmin.

Muuttaaksesi asetuksiasi voit tarkastaa oikean formaatin käyttäen näitä komentoja ja asettaa sijainnin ympäristömuuttujat vastaavasti tiedostoon "*.profile*".

Toinen olennainen lokalisaation elementti on tahtomasi näppäimistön asettelu, jonka voit asettaa komennolla "loadkeys" komentoriviä varten ja komennolla "setxkbmap" graafista käyttöjärjestelmää X Window System varten.

```
$ loadkeys de-latin1 # Saksa
```

19. PARAMETRIEN KORVAAMINEN

Kuten näimme aiemmin muuttujia käsittelevässä luvussa, voit laittaa aaltosulut muuttujan nimen ympärille erottaaksesi sen ympäristöstään:

```
$ curr=myfile
$ rm ${curr}.jpeg
```

On myös joitain näppäriä temppuja, joita voit tehdä aaltosulkujen sisällä, kuten merkkijonon osien muuttaminen. Oletetaan, että sinulla on tiedosto, jonka nimi on *"mypicture.jpeg*" eikä "*myfile.jpeg"*. Voit muuttaa muuttujaa "$curr", kun laitat sen komentoon:

```
$ rm ${curr/file/picture}.jpeg
```

TURVALLISTA LEIKKIÄ OLEMATTOMIEN MUUTTUJIEN KANSSA

Joskus voit käyttää muuttujia, jotka on poistettu (minkä voit tehdä komennolla "`unset`") tai joita ei ollut koskaan alustettu. Koska oletusarvoisesti komentotulkki käyttää tyhjää merkkijonoa tai olematonta tai määrittelemätöntä muuttujaa, kuten aiemmin näyttämämme komennon "`rm`" lisäksi, on hyödyllistä voida korvata oletusarvoinen arvo muuttujalle.

```
$ cat "${VARIABLE_FILE_NAME:-/home/user/file}"
```

Operaattori "**:-**" kysyy päätteeltä onko muuttuja asetettu, nähdäkseen onko se olemassa ja asetettu joksikin merkkijonoksi. jos sitä ei ole koskaan määritelty, tai sillä ei ole arvoa, pääte korvaa tekstin operaattorin "**:-**" jälkeen.

```
$ cat "${VARIABLE_FILE_NAME:=/home/user/file}"
```

Operaattori "**:=**" tekee paljolti samat asiat, mutta sen sijaan että korvaisi *"/home/user/file*" nykyiseen komentoon "`cat`", jos "VARIABLE_FILE_NAME" ei ole olemassa, pääte asettaa myös muuttujaan vaihtoehtoisen tekstin.

ASIOIDEN HELPOTTAMINEN VAIHTOEHTOISELLA LAAJENNUKSELLA

Vaihtoehtoinen laajennus ei ole mitenkään rajoitettu tiedostonimiin. Se on myös näppärä tapa antaa monimutkaisia, usein käytettyjä valitsimia komennoille.

```
$ export ALT_LS='--color=always -b -h --filetype' $ ls $ALT_LS
```

Koska vaihtoehtoiset valitsimet on tallennettu ja laajennettu muuttujamuodossa, voit käyttää mitä tahansa oletusarvoja tahdot suurimmalle osalle työstäsi, mutta käyttää nopeasti vaihtoehtoista muotoa erikoistapauksissa.

Parametrien laajentaminen on loistava tapa toimia abstraktisti useampien tiedostojen tai pitkien valitsimien listojen kanssa. Kun ymmärrät sen, se laajentaa toimintamahdollisuuksiasi komentorivillä.

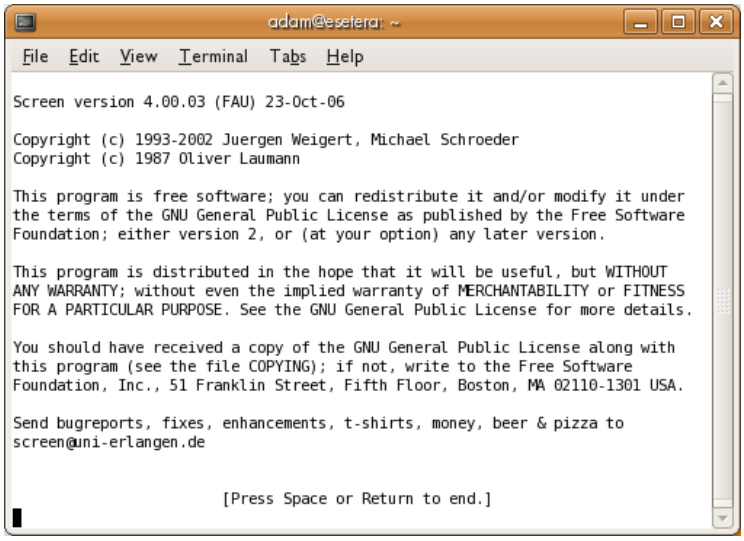
20. GNU SCREEN

GNU Screen auttaa sinua saamaan kaiken irti koneesi tehoista, mikäli sinun täytyy toimia useammassa kuin yhdessä päätteessä samanaikaisesti. Käyttämällä GNU Screeniä sinulla voi olla niin monta prosessia kuin tahdot, esimerkiksi editoreja, verkkoselaimia ja päätteitä, kaikki yhdessä pääteikkunassa. Jokainen työpöytäjärjestelmä antaa sinun avata useampia terminaaleja eri ikkunoissa, ja useimmat terminaaliohjelmat antavat sinun ajaa useampia istuntoja samaan aikaan välilehtiä käyttäen, mutta GNU Screen on usein helpommin hallittavissa ja vähemmän hämmentävä, kun sinulla on useita eri istuntoja. Lisäksi GNU Screen tarjoaa kopioi ja liitä -toiminnon, jolla voi helposti siirtää tekstinpätkiä useiden eri istuntojen välillä. Sitä voi käyttää tavallisessa tekstiterminaalissa ja työpöydällä, ja useimpia käyttötarkoituksia varten se on vaihtoehto useampien pääteistuntojen käytölle ja niiden välillä siirtymiselle **Alt + F#** näppäinyhdistelmällä.

Aloita kirjoittamalla "screen" komentokehotteessasi.

```
$ screen
```

Saat tervetuloviestin ja versiotietoa, tai järjestelmäsi asetukset voivat olla sellaiset, että se menee suoraan päätteen komentokehotteeseen.



Jos painat **Enter** saat komentokehotteen, aivan kuin ennen screenin käynnistämistä. Toimit nyt screenin yhden istunnon sisällä. Luodaksesi toisen istunnon omalla komentotulkillaan, paina **ctrl + a**, jota seuraa **c** tai **ctrl + c**. Kaikki ruudun komennot alkavat näppäimillä **ctrl + a**, mutta jos tahdot käyttää tuota näppäinyhdistelmää muihin tarkoituksiin, voit vaihtaa käytettyjä näppäimiä. Katso sitä käyttöoppaasta (kirjoita "man screen" Bashin komentokehotteessa) nähdäksesi näppäinkomentojen muuttamisen ja monia muita asioita, joita emme käsittele tässä oppaassa.

Jos tahdot ajaa vain yhden komennon uudessa ruudussa ja sitten sulkea ruudun, määrittele komento parametrinä:

```
$ screen irssi
```

ISTUNTOJEN VÄLILLÄ SIIRTYMINEN

Siirtyäksesi seuraavaan istuntoon, paina **ctrl + a**, jota seuraa näppäin **p**. Mennäksesi taas eteenpäin, paina **ctrl + a**, jota seuraa näppäin **n**. Voit nähdä kaikkien luomiesi istuntojen listan käyttäen näppäimiä **ctrl + a**, jota seuraa näppäin **"**. Tämä näyttää vieritysvalikon, joka listaa kaikki avoimet istunnot (tosi hyödyllistä, kun olet luonut monta istuntoa). Jos tahdot säästää edes sen puolisekuntisen, jonka valikon vierittäminen kestää, mutta näet samalla listan saatavilla olevista istunnoista ja hyppäät suoraan toiseen istuntoon, paina **ctrl + a w**. Tämä pitää sinut nykyisessä ikkunassa, mutta lisää ruudun pohjalle listan istunnoista, joista jokaiselle on eri numero. Paina sitten **ctrl + a *istunnon_numero*** hypätäksesi valitsemaasi istuntoon.

Oletetaan, että loit muutamia istuntoja GNU Screenillä. Yhdessä istunnossa on avoinna Vim-editori, paria muuta istuntoa käytät etäpalvelimille kirjautumiseen, yhdessä istunnossa käytät FTP:tä, ja niin edelleen. Oletusarvoisesti **ctrl + a "** näyttää ohjelman, jota käytit jokaisen istunnon käynnistämiseen. Normaalisti aloitit sen komentotulkissa, joten **ctrl + a "** näyttää "bash" jokaiselle istunnolle (tai mikä päätteesi onkaan). Tämä ei ole kovinkaan hyvä, jos tahdot nopeasti löytää istunnon, joka ajaa Vimiä tai FTP:tä.

Paljastuu, että näytön kustomointi on helppoa. Kun olet istunnossa - esimerkiksi Vim-editoinnissa - paina **ctrl + a** A ja ikkunan pohjaan ilmestyy rivi, jossa on istuntosi nimi, jonka voit muuttaa haluamaksesi.

KOPIOI JA LIITÄ

Kopioiminen ja liittäminen toimii graafisissa käyttöliittymissä toimivissa komentotulkeissa. Voit käyttää hiirtä valitsemaan tekstin yhdessä istunnossa ja liittämään sen toiseen istuntoon tai päätteeseen. Ohjelmalla screen on omat menetelmänsä tekstin kopiointiin, kuten tämä:

1. Paina **ctrl + a [** mennäksesi *ruudun kopionti* -tilaan.
2. Navigoi tekstin läpi missä tahansa ikkunassasi, käyttäen nuolinäppäimiä tai editorin komentoja tai mikä tahansa toimiikin tuossa istunnossa. Kopioitavan alueen alussa paina **välilyöntiä**.
3. Navigoi kopioitavan alueen loppuun (teksti korostetaan, kun teet valintaasi).
4. Paina **välilyöntiä** uudestaan kopioidaksesi tekstin leikepöydälle.
5. Siirry istuntoon, johon tahdot kopioida tekstin, navigoi oikeaan paikkaan, ja paina **ctrl + a]**. Tämä liittää valitun tekstin editointipuskuriin tai komentoriville tai minne tahansa, riippuen siitä, mitä tuo istunto tekee.
6. Toista askel 5 kuten tahdot.

Joko hiirimenetelmällä tai ruutumenetelmällä teksti otetaan videomuistista, joka ei säilytä pakomerkkejä, jotka tekevät värejä, lihavointeja ja niin edelleen. Luultavasti tahdot sen toimivan näin, jos kopioit osia käyttöoppaasta komentoriville, mutta ehkä tahtoisit muotoilut, mikäli kopioit otteita yhdestä tiedostosta toiseen.

RUUDUN JAKAMINEN

Useiden monen yhden ikkunan screenien lisäksi Screen tarjoaa myös mahdollisuuden laittaa monta ohjelmaa samaan screeniin.

Käytä komentoa **ctrl + a S** (iso S) jakamaan screenisi kahteen osaan. Alkuperäinen istuntosi on ylhäällä ja uusi tyhjä istunto alhaalla. Ole varovainen, että et vahingossa paina näppäimiä **ctrl + s**. Tämä voi lukita päätteesi. Jos painat vahingossa **ctrl + s**, voit purkaa päätteen lukituksen näppäimillä **ctrl + q**.) Oletusarvoisesti tällä uudella alueella ei ole suoritettava ohjelmaa, mutta voit aloittaa uuden painamalla **ctrl + tab** siirtyäksesi tuolle alueelle ja kirjoittamalla sitten **ctrl + a c**.

Jokainen alue toimii erillisenä istuntona, ja voit siirtyä istuntojen välillä aivan kuin kokoruudun tilassa.

Voit poistaa nykyisen alueen käyttämällä näppäimiä **ctrl + a X**. Tämä ei tuhoa istuntoa tai siinä toimivaa ohjelmaa. Se vain muuttaa toisen istunnon takaisin täysikokoiseksi ikkunaksi.

ISTUNNON IRROTTAMINEN

Yksi Screenin tehokkaimmista ominaisuuksista on istuntojen pysäyttäminen ja palauttaminen. Ehkä teet jotain todella mielenkiintoista tietokoneellasi, mutta joudut jättämään sen ja menemään töihin. Sanotaanpa, että kun olet poissa, tahdot päästä käsiksi siihen, mitä olit tekemässä. Jos molemmille koneille pääsee internetin kautta, voit tehdä tämän Screenin avulla. Voit esimerkiksi aloittaa istunnon etäkoneella käyttäen komentoa `telnet` tai turvallisempaa `ssh` -protokollaa, ja kutsua sitten komentoa `screen`. Kun sinun täytyy lähteä, voit turvallisesti sulkea etänä toimivan Screen -istunnon ja palauttaa sen myöhemmin, kenties ottaen yhteyttä etäkoneelle kolmannelta koneelta, joka sijaitsee toisella mantereella.

Kirjoita **ctrl + a d** screen-istunnossasi. Palaat alkuperäiseen päätteeseesi ja screen poistuu, tulostaen tekstin **[detached]**. Jos käytät nyt komentoa `ps`, huomaat että screen toimii edelleenkin taustalla. Hanki lista kaikista toimivista istunnoista antamalla komennolle `"screen"` valitsin `"list"`.

```
$ screen -list
There is a screen on:
12056.pts-0.hostname
(Detached)
1 Socket in
/var/run/screen/S-user_name.
$
```

Voit kytkeytyä uudestaan tähän istuntoon kirjoittamalla:

```
$ screen -r 12056.pts-0.hostname
```

Tai voit vain käyttää:

```
$ screen -R
```

Tämä kytkeytyy takaisin ensimmäiseen löytämäänsä istuntoon.

Nyt sinun tarvitsee vain kirjautua sisään kotikoneellesi miltä tahansa etäkoneelta ja näet mitä olitkin tekemässä, aivan kuin se oli sen jättäessäsi. Käytä samaa toimintatapaa laittaaksesi `wget` tai `ftp` -latauksen taustalle. Irrotettu istunto on olemassa, vaikka kirjautuisit ulos koneelta.

RUUDUN LOPETTAMINEN

Jos avoinna on vain muutama ohjelma, voit poistua ruudusta yksinkertaisesti lopettamalla ne kaikki. Jos sinulla on kuitenkin monia sovelluksia ja ikkunoita avoinna, voit poistua niistä kaikista kirjoittamalla **ctrl + a **. Sinulta kysytään vahvistusta, ja jos valitset kyllä, Screen lopettaa kaikki ohjelmansa ja poistuu.

21. SSH

Komentorivi on niin käyttökelpoinen työkalu ettei kestä kauan ennen kuin sinun täytyy päästä komentoriville koneella, joka ei ole edessäsi. Muinaisina aikoina turvallisuus ei ollut tärkeää ja ihmiset käyttivät komentoa `telnet`päästäkseen etäkoneen komentoriville. Komennon `telnet` käyttö ei ole enää hyvä idea, sillä tietoa siirretään raa'assa, salaamattomassa muodossa. Normaali turvallinen tapa päästä komentoriville etäkoneella on käyttää komentoa `ssh` (secure shell). Komennon yksinkertaisin muoto on

```
$ ssh toinenkone.domaini.fi
```

Tämä komento olettaa, että käyttäjänimesi etäkoneella on sama kuin käyttäjänimesi koneella, jolla kirjoitat komennon. Etäkone kysyy sinulta salasanaasi. Jos käyttäjänimesi etäkoneella on erilainen kuin käyttäjänimesi paikallisella koneella, käytä valitsinta `-l` (pieni "L") osoittamaan käyttäjänimesi etäkoneella.

```
$ ssh -l käyttäjätunnus toinenkone.domaini.fi
```

Vaihtoehtoisesti voit käyttää sähköpostityylistä merkintätapaa osoittamaan etäkoneen käyttäjänimen.

```
$ ssh käyttäjätunnus@toinenkone.domaini.fi
```

Tähän mennessä kaikki nämä komennot näyttävät komentorivin etäkoneella, josta voit sitten suorittaa mitä tahansa komentoja, joita tämä kone sinulle tarjoaa. Joskus tahdot ehkä suorittaa yhden komennon etäkoneella ja palata sitten oman koneesi komentoriville. Tämä saadaan aikaan laittamalla etäkoneella suoritettava komento heittomerkkien sisään.

```
$ ssh käyttäjätunnus@toinenkone.domaini.fi 'mkdir
/home/nimeni/hakemisto'
```

Joskus sinun täytyy suorittaa aikaavieviä komentoja etäkoneella, mutta et ole varma, onko sinulla tarpeeksi aikaa nykyisen ssh-istunnon puitteissa. Jos suljet etäyhteyden ennen kuin komento on suoritettu loppuun, tuon komennon suorittaminen jää kesken. Estääksesi työsi menettämisen voit aloittaa ssh-yhteydellä etäkoneella olevaan screen-istuntoon ja sen jälkeen irroittaa sen ja kytkeä sen minne tahdot. Irroittaaksesi etäkoneen screen-istunnon voit yksinkertaisesti sulkea ssh-yhteyden: irrotettu screen-istunto jatkaa pyörimistä etäkoneella.

`ssh` tarjoaa monia muita vaihtoehtoja, jotka kuvaillaan manuaalisivulla. Voit myös asettaa suosikkijärjestelmäsi sallimaan kirjauduttuasi komentojen suorittamisen ilman salasanasi kysymystä joka kerralla. Järjestely on monimutkainen mutta voi säästää sinulta paljon näppäilyä: kokeile etsiä webistä käsitteitä "ssh-keygen", "ssh-add" ja "authorized_keys".

SCP: TIEDOSTOJEN KOPIOINTI

SSH-protokolla ulottuu kauas peruskomennon `ssh` tuolle puolen. Erityisen hyödyllinen SSH-protokollaan perustuva komento on `scp`, turvallinen kopiointikomento. Seuraava esimerkki kopioi tiedoston nykyisestä hakemistosta pakallisella koneellasi hakemistoon */home/minä/juttuja* etäkoneella.

```
$ scp myprog.py
käyttäjätunnus@toinenkone.domaini.fi:/home/minä/juttuja
```

Varo, sillä komento ylikirjoittaa minkä tahansa tiedoston, joka on jo olemassa nimellä */home/minä/juttuja/ohjelmani.py*. (Tai saat virheviestin, jos on tämän niminen tiedosto, eikä sinulla ole käyttöoikeutta ylikirjoittaa sitä.) Jos */home/me* on kotihakemistosi, kohdehakemisto voidaan lyhentää.

```
$ scp ohjelmani.py
käyttäjätunnus@toinenkone.domaini.fi:juttuja
```

Voit kopioida toiseen suuntaan yhtä helposti: etäkoneelta paikalliselle koneelle.

```
$ scp
käyttäjätunnus@toinenkone.domaini.fi:Asiakirjat/haastattelu.txt
eilinen-haastattelu.txt
```

Tiedostonimi etäkoneella on *haastattelu.txt* kotihakemistosi alihakemistossa *Asiakirjat*. Tiedosto kopioidaan tiedostoon eilinen-haastattelu.*txt* paikallisessa järjestelmässäsi.

Komentoa `scp` voidaan käyttää kopioimaan tiedosto etäkoneelta toiselle etäkoneelle.

```
$ scp käyttäjä1@kone1:tiedosto1
käyttäjä2@kone2:toinenhakemisto
```

Kopioidaksesi rekursiivisesti kaikki tiedostot ja alihakemistot hakemistossa käytä valitsinta `-r`.

```
$ scp -r käyttäjä1@kone1:hakemisto1 käyttäjä2@kone2:hakemisto2
```

Katso opassivua `scp` nähdäksesi lisää valitsimia.

RSYNC: AUTOMAATTISET JOUKKOSIIRROT JA VARMUUSKOPIOT

`rsync` on erittäin hyödyllinen komento, joka pitää etähakemiston synkronoituna paikallisen hakemiston kanssa. Me mainitsemme sen tässä, koska se on hyödyllinen tapa käyttää verkkoa komentoriviltä, aivan kuin `ssh`, ja koska SSH-protokollaa suositellaan `rsync` -komennon pohjana olevan tiedonsiirron protokollaksi.

Seuraavassa on yksinkertainen ja käyttökelpoinen esimerkki. Se kopioi tiedostoja paikallisesta */home/nimeni/Asiakirjat* hakemistoon nimeltä *backup/* omassa kotihakemistossasi järjestelmässä *quantum.example.edu*. `rsync` itse asiassa minimoi tarpeellisen kopioinnin määrän kehittyneiden tarkastusten avulla.

```
$ rsync -e ssh -a /home/nimeni/Asiakirjat
minä@quantum.example.edu:backup/
```

Valitsin `-e ssh` käyttää SSH-protokollaa siirron pohjana, kuten on suositeltu. Valitsin `-a` (merkitsee "arkistoi") kopioi kaiken määritellystä hakemistosta. Jos tahdot tuhota tiedostot paikallisesta järjestelmästä kun ne koipioidaan, käytä valitsinta a `--delete`. Katso komennon `rsync` opaskirjasivua saadaksesi lisätietoa komennosta `rsync`.

22. OHJELMIEN ASENTAMINEN

Ohjelmien asentaminen GNU/Linuxiin on laaja aihe, sillä jokaisella GNU/Linux-versiolla on oma tapansa tehdä asioita. Monet ovat ohjelman apt-get (Advanced Packaging Tool) versioita. Tätä käyttää Debian, Ubuntu, gNewSense ja näiden jakeluversioiden sukulaiset. Toinen vaihtoehto on yum (Yellowdog Update Manager), jota käyttää Fedora, BLAG ja joukko muita jakeluita. Perussyntaksi on:

```
$ sudo apt-get install paketinnimi
$ sudo yum install paketinnimi
```

Monet ohjelmien apt-get ja yum toiminnot ovat saman nimisiä ja toimivat samalla tavalla, mutta eivät kaikki. Kun tahdot mennä tässä kuvailtuja yksinkertaisia tapauksia pidemmälle, tarkasta dokumentaatio sille ohjelmalle, jota käytät.

Nämä esimerkit käyttävät komentoa `sudo` muistuttamaan sinulle, että ohjelmien asentaminen ja konfiguraatiotiedostojen muokkaaminen vaativat pääkäyttäjän oikeuksia. Voit käyttää joko komentoa `sudo` jokaisen komennon kanssa tai siirtyä pääkäyttäjäksi komennolla `su`. (Muista poistua pääkäyttäjän istunnosta ennen kuin jatkat normaalia työskentelyä.)

Jokaiselle komennolle on monta valitsinta. Kokeile tätä komentoa poistaaksesi paketin.

```
$ sudo apt-get remove paketinnimi
$ sudo yum remove paketinnimi
```

Kone lukee ohjelmavarastojen sisällysluettelot ja päivittää paikallisen pakettitietokannan.

```
$ sudo apt-get update
$ sudo yum update
```

Asentaaksesi kaikki saatavilla olevat uudemmat versiot paketeista.

```
$ sudo apt-get upgrade
```

Korjataksesi hajonneet riippuvuudet, mikäli sellaisia on.

```
$ sudo apt-get --fix-broken
```

Komennolla `yum` ei ole tätä valitsinta. On olemassa muita tapoja käsitellä hajonneita RPM -pakettiriippuvuuksia, mutta ne tarvitsevat enemmän apua, kuin voimme tässä tarjota.

Käyttäjät voivat määritellä useampia ladattavia pakettivarastoja muokkaamalla tiedostoa */etc/apt/sources.list* pääkäyttäjänä. Ole varovainen. Tee varmuuskopio nykyisestä tiedostosta ennen kuin teet muutoksia.

Kaikki GNU/Linuxit antavat käyttäjän asentaa ohjelmia lähdekoodia käyttäen. Asentaaksesi ohjelmia Debian-tyyppisistä paketeista, voit käyttää

```
$ apt-get source paketinnimi
```

Ohjelma `yum` ei käsittele asennuksia lähdekoodista.

Lähdekoodista kääntäminen on erityisen tärkeää sellaisille ohjelmille, joita ei ole saatavilla paketteina, tyypillisesti, koska ne ovat liian uusia. Et luultavasti tahdo taistella tämän prosessin kanssa, ellet tiedä hieman GNU/Linuxin komennoista ja tiedostojärjestelmästä, mutta jos päätät tehdä jotain uutta ja ehkä keskeneräistä, tämä on yleisin menetelmä. Jos et tiedä paljonkaan komennoista ja tiedostojärjestelmistä, voit eksyä helposti tehdessäsi avoimen lähdekoodin asennusta. On paras lukea niistä ensin ja jatkaa sitten tästä kohdasta.

Asennus lähdekoodista toimii missä tahansa GNU/Linux -järjestelmässä, jossa on kääntäjä ja siihen liittyvät työkalut ja kirjastot, joten on hyvä tietää miten tämä prosessi toimii, kunhan saat ensin lähdepaketin käsiisi:

1. Pura arkisto ja siirry komennolla `cd` sen perushakemistoon.
2. Aja konfiguraatioskripti **./configure**
3. Käännä ohjelma komennolla **make**
4. Asenna ohjelma komennolla **make install**

Toisen ja kolmannen askeleen suorittamiseksi tarvitset kääntäjän järjestelmääsi. Jotkin GNU/Linux -järjestelmät sisältävät automaattisesti nämä työkalut, mutta muut eivät sisällä. Mikä tahansa järjestelmä, jota käytät tämän kirjan kanssa, luultavasti antaa sinun ladata tässä tarvitsemasi työkalut: etsi paketteja, jotka sisältävät työkalut `gcc` ja `binutils`.

RIIPPUVUUDET

Ennen kuin aloitamme sanomme sanasen riippuvuuksista. GNU/Linux kehittäjät eivät aina kirjoita ohjelmaa nollapisteestä alkaen; he käyttävät hyväkseen paljon muiden ohjelmoijien aikaisemmin tekemää työtä. Tämä on viisasta, sillä se säästää aikaa, ja tämän prosessin helpottamiseksi monet kiltit ihmiset ovat tehneet koodikirjastoja, joita muut voivat helposti käyttää omissa ohjelmissaan. Nämä kirjastot on tallennettu pysyviin sijainteihin GNU/Linux-järjestelmässä, yleensä niihin hakemistoihin, joiden nimet alkavat */lib*, */usr/lib* ja */usr/share/lib*.

Jos asennat sovelluksen, joka vaatii tiettyjä kirjastoja, se on helppoa niin kauan kuin sinulla on nuo kirjastot valmiiksi asennettuina järjestelmässäsi. Jos sinulla ei kuitenkaan ole vaadittuja kirjastoja, joudut etsimään ja asentamaan ne. Jos ohjelmoijat ovat olleet ajattelevaisia, he ovat lisänneet ohjelmaan tiedot riippuvuussuhteista joko *README* tai *INSTALL* -tiedostoihin, jotka löydät sovelluksen lähdehakemistosta. Jotkut äärimmäisen kiltit ohjelmoijat antavat sinulle nimen ja osoitteen, josta löydät tarpeelliset ohjelmat.

Jos kuitenkin asennat ohjelmaa johonkin muuhun jakeluversioon kuin siihen, jota varten se on ohjelmoitu, kirjastot on usein pakattu eri tavalla kuin kehittäjän järjestelmässä. Tässä tapauksessa joudut ehkä kokeilemaan ja katsomaan virheviestejä: yritä kääntää lähdekoodi, ja kun saat virheviestin, joka kertoo sinulle puuttuvasta riippuvuudesta, yritä asentaa se. Jos et voi asentaa sitä annetulla nimellä, joudut ehkä kysymään joltain kokeneemmalta neuvoa oikean paketin löytämiseen, tai katsoa jakeluversiosi dokumentaatiosta tietoa paketointikäytännöistä.

Yleensä GNU/Linuxin käyttäjät eivät jaksa lukea näitä tiedostoja, he vain suorittavat standardiprosessin ja huomaavat, että konfiguraatiovaiheessa löytyy virhe, joka kertoo heille puuttuvista kirjastoista. nämä laiskat tyypit (tämän kirjoittaja mukaanluettuna) löytävät sitten tarvitut bitit ja palat verkossa ja asentavat ne.

Jos olet kuitenkin uusi GNU/Linuxin käyttäjä, ehdotan että luet tiedostot *README* ja *INSTALL* ennen asennusprosessin aloittamista. Se säästää sinulta paljon aikaa ja sydänsurua.

Muista kuitenkin, että vaikka riippuvuuslista voi olla pitkä, voit yksinkertaisesti hankkia kaikki tarvittavat paketit ja asentaa ne yksi kerrallaan seuraten edellisessä osassa kuvattua prosessia, kunnes lopulta sinulla on kaikki unelmiesi ohjelman asentamiseen ja suorittamiseen tarvittava.

Katsotaanpa seuraavaksi asennusprosessia hieman syvällisemmin.

ARKISTON PURKAMINEN

Useimmat ohjelmistolähteet ovat pakattuja "nauha-arkistotiedostoja", joiden loppupääte on yleensä ".tar" tai ".tgz". GNU:n `tar` -komento voi automaattisesti purkaa tiedostot, jotka loppuvat päätteeseen *.gz* tai *.tgz* (mikä merkitsee, että jakelija on käyttänyt GZIP -pakkausta), mutta jos muita pakkauksen muotoja on käytetty (kuten BZIP2 tai LZMA), voit käyttää sopivaa purkamisohjelmaa saadaksesi esille *.tar* -tiedoston. Käytä komentoa `tar` purkaaksesi arkiston:

```
$ tar zxvf paketinnimi.tar.gz
```

Jossa "paketinnimi" esimerkissä on sen paketin oikea nimi, jonka tahdot asentaa. Komentoa `tar` seuraa parametrit `zxvf` ja se purkaa *tar.gz* -tiedoston ja luo uuden hakemiston kaikista puretuista lähdetiedostoista. Valitsin 'z' määrittää BZIP -pakkauksen; jos tiedostopääte on ".tgz2", määritä BZIP2 -pakkaus käyttämällä valitsinta 'j'. Älä huolestu - jos se ei onnistu purkamaan tiedostoa, saat vain virheviestin. Voit poistaa *tar.gz* -tiedoston sen purettua itsensä onnistuneesti.

Voit nyt muuttaa työhakemistosi tähän uuteen hakemistoon käyttämällä komentoa `cd`. Yleensä hakemiston uusi nimi on pakatun lähdepaketin nimi miinus pakkausloppupääte. Jos esimerkiksi pakettini nimi tosiaan oli *uusiohjelmapaketti-1.0-alpha.tar.gz*, se on komennon `tar zxvf` suorittamisen jälkeen purettu hakemistoon, jonka nimi on *uusiohjelmapaketti-1.0-alpha* ja voisit kirjoittaa komennon `cd uusiohjelmapaketti-1.0-alpha` päästäksesi tähän uuteen hakemistoon. Jos et ole varma tämä uuden hakemiston nimestä, kirjoita `ls`.

AJA KONFIGURAATIOSKRIPTI

Kun olet uuden hakemiston sisällä, tahdomme sinun aloittavan varsinaisen asennusprosessin. Tehdäksesi tämän joudut useimmiten kirjoittamaan seuraavan komennon:

```
$ ./configure
```

Oikein pakatut lähdekoodin jakeluversiot sisältävät yleensä skriptin, joka tarkastaa tarvitut apuohjelmat ja binääritiedostot, sekä valmistelee lähdekoodin puun rakennusta ja asennusta varten. Tässä tapauksessa oletamme että se skripti on `configure`, sillä se on hyvin suosittu valinta sellaiseksi skriptiksi. Joskus joudut käyttämään jotain muuta skriptiä. niissä tapauksissa voit katsoa tietoa tiedostosta *README* tai *INSTALL*.

Näytetyssä komennossa voit laittaa pisteen ja kauttaviivan ennen skriptin nimeä (`./configure`), jolloin kerrot GNU/Linuxille ajaa skriptin, jonka nimi on `configure` ja joka on nykyisessä hakemistossa (tähän viitataan merkeillä `"./"`). Tämän jälkeen skripti ajetaan, se tarkastaa millainen tietokone sinulla on, mitä olet jo asentanut, millainen GNU/Linux sinulla on, ja niin edelleen.

Yksi mahdollisuus komennossa `configure` on erityisen yleinen: valitsin `--prefix`, joka käskee komennolle `configure` asentaa tiedoston oletusarvoisesta poikkeavaan sijaintiin. Useimmissa järjestelmissä oletusarvoinen sijainti on hyvä valinta, ja se voi olla se sijainti, josta muut ohjelmat odottavat löytävänsä ohjelman tai kirjaston, jota olet asentamassa. Joskus et voi asentaa ohjelmaa jaettuun sijaintiin tai tahdot sen olevan jossain oman kotihakemistosi alla, sillä tiedät olevasi ainoa sitä käyttävä ihminen. Vaihtaaksesi hakemistoa, johon ohjelma lopulta asennetaan, määrittele se valitsimella `--prefix`:

```
$ ./configure --prefix ~/bin/ohjelmani
```

Yleisin ongelma on, että tässä vaiheessa konfiguraatioskripti pysähtyy ja kertoo että joku ohjelmistokirjasto, jota ohjelma tarvitsee, puuttuu. Jos kohtaat tämän virheen, tarkasta *README* ja *INSTALL* -tiedostot siltä varalta, että ne kertovat sinulle tavan korjata ohjelman, jos et saa tätä selville, joudut käyttämään hakukonetta selvittääksesi, mitä ohjelman virheviesti sanoo, ja miten voit korjata ongelman. Tämä merkitsee, että asennus voi joskus kestää päiväkausia, kun etsit ja lataat kaikki tarvitsemasi paketit. Tämä on yksi paketinhallinnan suurista eduista, joiden vuoksi kannattaa käyttää ohjelmia, kuten yum ja apt-get: kun kehittäjät luovat paketteja näihin järjestelmiin, he automatisoivat riippuvuuksien asentamisen.

Joissain tapauksissa riippuvuudet ovat valinnaisia. Skripti `configure` tukee itse asiassa monia valitsimia. Voit nähdä mitä valitsimia ohjelmistopakettisi tukee ajamalla komennon:

```
$ ./configure --help
```

KÄÄNNÄ OHJELMISTO

Olettaen, että `configure` -prosessi lopetti onnistuneesti, seuraava asennusprosessissa kirjoitettava komento on:

```
$ make
```

Jos sinulla on monta prosessoria tai prosessoriydintä, voit käyttää useampaa työtä, prosessoinnin nopeuttamiseen lisäämällä valitsimen `-j`:

```
$ make -j3
```

Nämä komennot kääntävät ohjelman sinulle. Sen jälkeen sinulla on paljon käännettyjä tiedostoja, jotka muodostavat yhdessä ohjelmasi. Prosessi `make` voi kestää jonkin aikaa, riippuen koneesi nopeudesta, sekä asentamiesi pakettilähteidesi koosta. Muiden prosessoritehoa vaativien sovellusten ajaminen voi myös hidastaa prosessia.

Toisessa näytetyssä komennossa valitsin `-j3` kertoo komennolle `make` ajaa kolmea käännösprosessia saamaan aikaan, mikä antaa sinun käyttää prosessoritehoja paremmin, mikäli sinulla on dual-core tai tehokkaampi prosessori. Numero valitsimen `-j` jälkeen on valinnainen, mutta hyvä peukalosääntö asiaan on prosessoriytimien määrä plus yksi.

Kuten komennon `configure` tapauksessa, voit kohdata virheitä kääntämisen aikana. Sellaisessa tapauksessa, mikäli et voi korjata ohjelmaa itse, ota yhteyttä ohjelman kehittäjään ja pyydä häneltä kohteliaasti apua, selittäen ongelmasi hyvin selkeästi. Verkkosivu http://www.catb.org/~esr/faqs/smart-questions.html selittää, kuinka voit kirjoittaa kohteliaita ja käyttökelpoisia ongelmaraportteja. Mutta katso ensin onko ohjelmista configure ja make lokitiedostoja. Tämä voi antaa sinulle enemmän tietoa kuin ruudulla näkyi, jopa siinä tapauksessa että näit mitä ruudulla oli, kun teksti vilisi ohi. Voit myös toistaa nämä askeleet, lisäten komentoon " &> logfile" kaapataksesi kaiken ulostulon lokitiedostoon (käytä tiedostonimeä, jota ei ole jo olemassa). Ennen make-komennon toistamista sinun pitäisi luultavasti käyttää komentoa "make clean" poistaaksesi edellisten askelien tekemät askeleet.

ASENNA OHJELMA

Kun `make` on lopettanut toimintansa ilman virheitä, kirjoita seuraava komento:

```
$ sudo make install
```

Tämä asentaa uudet luodut tiedostot ohjelmaasi varten oikeisiin paikkoihin järjestelmässäsi. Tämä on yleensä kohdassa */usr/local/*, koska tämä voidaan ohittaa komennon `configure` valitsimella, kuten olemme nähneet. Koska ohjelmistot on yleensä asennettu jaettuun hakemistoon, johon ainoastaan pääkäyttäjä voi kirjoittaa, sinun täytyy aloittaa komennolla `sudo`, jotta sinulla on luvat lisätä ohjelmasi. Et tarvitse komentoa `sudo`, mikäli käsket komennon `configure` asentamaan hakemistoon oman kotihakemistosi alla.

Joten nyt joudut vain kirjoittamaan sovelluksen nimen pääteikkunaasi ja sen pitäisi toimia. Jos se ei käynnisty, yleinen tapa korjata ongelma on kirjoittaa `ldconfig` ja yrittää sitten uudestaan. `ldconfig` päivittää järjestelmän, jotta käyttöjärjestelmäsi tietää, että uusi kirjasto on lisätty.

TEKSTIEDITORIT

23. TEKSTIEDITORIT
24. NANO
25. VI JA VIM
26. EMACS
27. KEDIT, KWRITE JA KATE
28. GEDIT

23. TEKSTIEDITORIT

Yksinkertaisten komentojen, kuten `ls` ja `grep`, lisäksi voit käyttää komentoriviä aloittaaksesi laajoja, monimutkaisia ohjelmia. Ennen kuin graafiset käyttöliittymät olivat yleisiä, ohjelmat suunniteltiin käyttämään pelkkää tekstiä ja valtaamaan koko ruutu. Nykyisin nämä ohjelmat toimivat samassa ikkunassa, jossa käytät komentoriviä.

Tässä kirjassa keskitymme tekstieditoreihin, koska tarvitset niitä tallentaaksesi komentosi myöhempään käyttöön ja kirjoittaaksesi skriptejä. Ne ovat käyttökelpoisia myös moniin muihin tarkoituksiin: tahdot ehkä editoida HTML-tiedostoja verkkopalvelimella käyttäen tekstieditoria.

TEKSTINKÄSITTELYOHJELMA VAI TEKSTIEDITORI

Miltei jokainen tietokoneen käyttäjä on tutustunut tekstinkäsittelyohjelmaan. Avointen ohjelmien maailma tarjoaa monia tehokkaita tekstinkäsittelyohjelmia, joihin kuuluu OpenOffice.org ja KWrite. Tässä kirjassa esittelemämme tekstieditorit muokkaavat tekstiä, aivan kuin tekstinkäsittelyohjelmat, mutta niissä on olennainen ero.

- Tekstinkäsittelyohjelmat tallentavat paljon enemmän kuin tekstin merkkien virran, jonka näet ruudulla. Ne tarjovat "rikasta tekstiä", jossa on tarjolla kursiivi ja lihavointi, numerointi ja luettelomerkit, värit - mitä tahansa haluatkin. Tämä on selvästi hyödyllistä monissa tapauksissa. Pelkkään tekstiin perustuva ansioluettelo ei innosta monia työnantajia.
- Toisaalta tekstinkäsittelyohjelmat näyttävät rajansa näinä päivinä: jotkin tekstinkäsittelyohjelmat käyttävät yksityisomistuksessa olevia formaatteja, jotka vaikeuttavat dokumenttien käyttöä muissa ohjelmissa. Itse asiassa et usein voi avata dokumenttia saman tekstinkäsittelyohjelman toisessa versiossa, tai dokumentti näkyy huonosti.
- Monista ihmisistä verkon ryhmätyökalut (kuten wikiohjelma, jolla tämä käyttöopas on kirjoitettu) ja sisällönhallintajärjestelmät (kuten monet blogiohjelmistot) ovat helppokäyttöisempiä nykyaikaiseen dokumentintuotantoon kuin tekstinkäsittelyohjelma. Tekstinkäsittelyohjelmat kehittyvät kuitenkin tukemaan yhteistyötä paremmin: luultavasti nämä kaikki työkalut yhdistyvät ajan myötä ja muuttuvat joksikin paremmaksi, kuin mikään aiempi vaihtoehto.

Tekstinkäsittelyohjelmat, wikit, sisällönhallintajärjestelmät ja tekstieditorit ovat kaikki tarpeellisia eri tarkoituksiin. Tämän kirjan toimenpiteet vaativat tekstieditoria. Jos tahdot käyttää tekstinkäsittelyohjelmaa näiden tiedostojen käsittelyyn voit tehdä niin, mutta varmista, että valitset tavallisen tekstimuodon, kun tallennat tiedoston.

MIKSI TARVITSET TEKSTIEDITORIN?

GNU/Linux on hyvin tiedostokeskeinen käyttöjärjestelmä: kaikki on tiedostoa (tai näyttää tiedostolta). Kaikki peruskonfiguraatio tehdään tarkkaan laadituilla tekstitiedostoilla, jotka ovat oikeassa paikassa oikealla sisällöllä. Voit löytää monia graafisia työkaluja GNU/Linux-koneesi konfigurointiin, mutta useimmat niistä vain muokkaavat teksitiedostoja puolestasi.

Noissa tekstitiedostoissa on tarkka syntaksi, jota joudut seuraamaan. Yksinkertainen väärässä paikassa oleva merkki voisi asettaa järjestelmäsi vaaraan, joten tekstinkäsittelyohjelman käyttö tähän tarkoitukseen ei ole ainoastaan huono idea, vaan voisi myös sotkea tiedostosi ylimääräisellä muotoilutiedolla. Konfiguraatiotiedostot eivät tarvitse kursiivia tai lihavointia, ne tarvitsevat vain oikeat tiedot.

Lähdekoodin kanssa on kyse samasta asiasta. Kääntäjät (ohjelmat jotka muuttavat koodia muiksi ohjelmiksi) ovat hyvin tarkkoja syntaksin kanssa. Jotkin niistä välittävät jopa tietyn komennon sijainnista rivillä. Tekstinkäsittelyohjelmat sotkevat tekstin sijainnin riveillä aivan liian pahasti, että kääntäjä voisi pitää niistä. Tarvitset selkeän kuvan siitä, mitä lähdekoodissa tai konfiguraatiotiedostossa on tietääksesi, mitä kirjoitat, sillä järjestelmäsi saa täsmälleen saman tekstin sisäänsä.

Jotkin editorit menevät vieläkin pidemmälle: niistä tulee kehitysympäristöjä (englanniksi Integrated Development Environments, IDE), jotka eivät ainoastaan ymmärrä mitä kirjoitat (olipa kyse Apachen konfiguraatiotiedostosta tai Java-koodista), mutta voivat ennustaa mitä olet kirjoittamassa, ehdottaa muutoksia ja näyttää virheesi. Ne voivat värittää tietyt avainsanat, sijoittaa asiat automaattisesti oikeisiin paikkoihin ja niin edelleen.

Mutta tärkeintä on, että kaikki nuo värit ja korostukset tehdään vain näytöllä. Nämä hienot muutokset eivät siirry tekstitiedostoihin, jotka on tarkoitettu olemaan pelkkää tekstiä. Tämä on erityisen hyödyllinen ominaisuus, jota tekstinkäsittelyohjelmat eivät voi tehdä, ja se on kaikkein olennaisinta tekstieditorin toiminnassa.

MIKSI USEIMMAT TEKSTIEDITORIT OVAT KOMENTORIVIOHJELMIA?

Alussa oli komentorivi. Kaksikymmentä vuotta sitten ei ollut moniakaan graafisia käyttöliittymiä ja Unix oli jo täysikasvuinen käyttöjärjestelmä, joka toimi lukemattomilla hyvin tärkeillä tietokoneilla. Kaikki konfiguraatiot oli jo tallennettu tekstitiedostoihin yksinkertaisuusperiaatteen vuoksi. Unix käytti yksinkertaisuusperiaatetta paljon ja tekstitiedostojen käytöstä tehtiin yksinkertaisempaa antamalla ohjelmien toimia yhteistyössä tekstitiedostojen kanssa. Putket (merkin | käyttö) ovat yksi tehokas tapa käyttää ohjelmia yhteistyössä, kuten olet nähnyt tässä kirjassa.

Nykyisin tietokoneilla on tuhansia kertoja enemmän tehoja kuin noilla vanhoilla koneilla, mutta konfiguraatioiden pitäminen tekstitiedostoissa antaa yhä suuren edun, kun ainoa yhteys palvelimeen on 56-Kbit modeemin kautta ja se on toisessa maassa. Graafisen käyttöliittymän avaaminen ei välttämättä olisi mahdollista, ja jos se olisi ainoa tapa ratkaista ongelma, olisit pulassa.

Konfiguraatioita muokkaavien graafisten ohjelmien tekeminen oli suuri kehitysaskel, koska keskimääräinen käyttäjä voi nyt muuttaa asioita lukematta tuhansia sivuja dokumentaatiota ja ei hajota järjestelmää helposti kirjoittamalla yhden väärän kirjaimen kohtaan, jossa virhettä ei voi korjata, mutta tekstitiedoston ja komentoriviltä toimivan tekstieditorin tarjoaminen on olennaista jokaiselle käyttöjärjestelmälle.

Vaikka useimmat tekstieditorit tulevat komentorivin maailmasta, useimmilla on nykyisin myös graafinen käyttöliittymä. Valikot ja nappulat auttavat paljon Gvimiä tai Emacsia käyttäessä. GEdit ja Kate (joka on täysin graafinen) ovat lyhyitä ja yksinkertaisia, mutta tarjoavat yhä saman perustoiminnallisuuden ja samat tärkeät ominaisuudet tekstieditointia varten.

OLETUSTEKSTIEDITORIN ASETTAMINEN

Koska pääte ja komentorivi ovat niin sidottuja tekstieditoreihin, monet komennot voivat avata sinulle tekstieditorin. Näimme esimerkiksi `sudoedit`-komennon osassa "Hyödyllisiä kustomointeja". Voit asettaa oletuseditorin asettamalla joko ympäristömuuttujan EDITOR tai VISUAL. Esimerkiksi:

```
$ export EDITOR=emacs
```

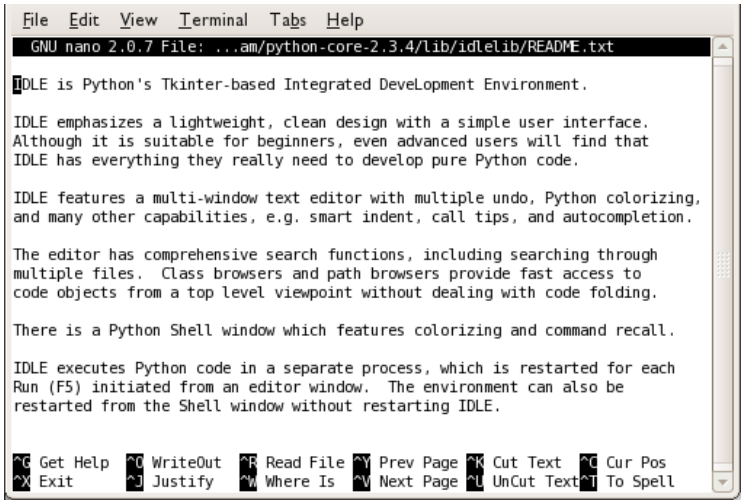
Laita tämä asetustiedostoon kuten *./bashrc*, ja komennot käyttävät valitsemaasi editoria, kun ne näyttävät editoitavan tiedoston.

24. NANO

Nano on yksinkertainen editori. Avataksesi sen ja luodaksesi uuden tekstitiedoston, kirjoita seuraava komentorivillä:

```
$ nano tiedostopolku
```

jossa *tiedostopolku* on polku tiedostoon, jota tahdot muokata (tai ei mitään). Ruudulle ilmestyy ohjelma, kuten näytetään seuraavassa kuvassa.



Ruutu ei ole enää paikka komentojen suorittamissen; siitä on tullut tekstieditori.

NANOSTA POISTUMINEN

Poistuaksesi ohjelmasta `nano`, pidä pohjassa **Ctrl** -näppäintä ja paina näppäintä **x** (tämän näppäinyhdistelmän merkitsemme tässä kirjassa **ctrl + x**). Jos olet luonut tai muuttanut tekstiä, mutta et ole vielä tallentanut sitä, `nano` kysyy:

```
Save modified buffer (ANSWERING "No" WILL DESTROY CHANGES) ?
```

Tallentaaksesi muutokset paina **y** ja nano kysyy kohdehakemistoa. Hylätäksesi muutoksesi paina nd **n**.

Tallentaaksesi muutokset poistumatta ohjelmasta, paina **ctrl + o**. `nano` kysyy sinulta tiedostonimeä, johon teksti kirjoitetaan:

```
File Name to Write:
```

Kirjosta tiedoston nimi ja paina **Enter**-näppäintä (tai jos puskurilla on jo oikea nimi, paina vain **Enter**). Esimerkiksi:

```
File Name to Write: tekstitiedosto.txt
```

TIEDOSTOJEN TUTKIMINEN

Voit siirtyä ympäriinsä tiedostossa ja katsoa eri osia käyttäen nuolinäppäimiä. Tämä on nopea ja ehokas tapa tutkia tiedostoa.

APUA

Lue nanon käyttöopassivu, koska siinä on paljon hyviä vinkkejä. Aputiedosto on nähtävillä nano-istunnon aikana painamalla **ctrl + g** ja päästäksesi takaisin tiedostoosi paina **ctrl + x**.

25. VI JA VIM

Vi on erittäin tehokas komentorivin tekstieditori. Sitä käytetään mihin tahansa nopeista konfiguraatiotiedostojen korjauksista ammattimaiseen ohjelmointiin ja jopa suurten ja monimutkaisten dokumenttien kirjoittamiseen. Se on suosittu, koska se on nopea ja kevyt ohjelma, ja voit tehdä sillä monia asioita muutamalla näppäimen painalluksella. Toisaalta se on myös tehokas: erittäin helposti konfiguroitava ja siinä on paljon sisäänrakennettuja toimintoja.

Vim on kehittyneempi versio ohjelmasta Vi, se tarjoaa ominaisuuksia, jotka helpottavat sekä aloittelijan että asiantuntijan työskentelyä (Vim tarkoittaa *"Vi IMproved"*). Monissa nykyaikaisissa järjestelmissä Vim on asennettu Vi-ohjelman oletusarvoisena versiona. Jos siis käytät komentoa `vi`, ajat itse asiassa ohjelman Vim. Tämä ei yleensä ole hämmentävää, sillä kaikki Vin ominaisuudet voimivat myös Vimissä. Katsomme tässä luvussa Vimiä, mutta jos järjestelmässäsi on Vi, voit käyttää samoja tekniikoita, kunhan korvaat kaikki viittaukset komentoon `vim` viittauksilla komentoon `vi`.

Paras ominaisuus ohjelmassa Vim/Vi on, että se on oletusarvoisesti mukana miltei kaikissa GNU/Linuxin versioissa. Kun opit Vimin, voit käyttää sitä missä tahansa oletkin.

Vimin suurin ongelma on, että samoin kuin Emacsissa (toinen komentorivin editori), sen oppimiskäyrä on vaikea. Näppäimistön pikakomentoja voi olla vaikea oppia.

Onneksi voit kiertää nämä ongelmat käyttämällä Vimin graafista versiota GVim, jossa kaikki nappulat ja valikot ovat tarjolla graafisemmille käyttäjille. Voit kokeilla myös ohjelmaa easy-Vim, joka tarjoaa Notepad-tyylistä editointia.

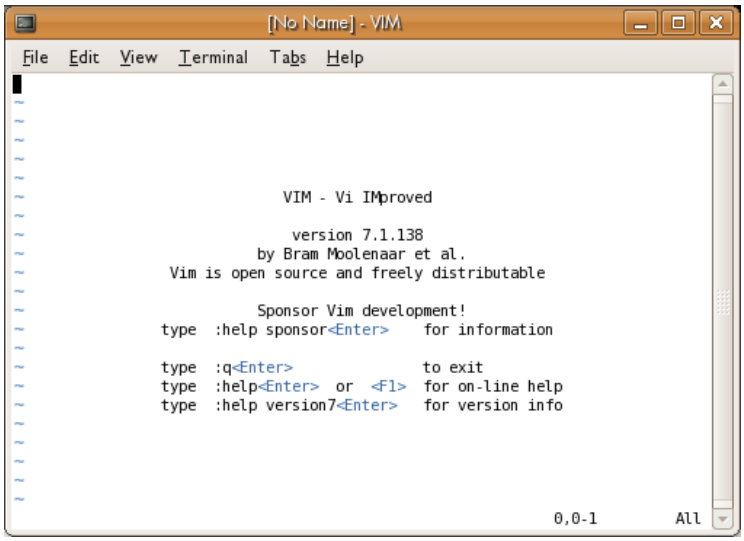
Nämä Vimin yksinkertaistetut versiot helpottavat oppimiskäyrää paljon ja näyttävät vähemmän kehittyneille käyttäjille tehokkaan editoinnin voiman, mikä puolestaan lisää tahtoa opetella tehokkaamman editorin käyttö.

PERUSKOMENNOT

Avataksesi Vimin ja aloittaaksesi uuden tekstitiedoston muokkaamisen, avaa komentorivi ja kirjoita:

```
$ vim
```

Tämä näyttää sinulle tyhjän ruudun, tai tällaisen ruudun, jolla on tietoa:



Jos tahdot avata olemassaolevan tiedoston, kirjoita se parametriksi komentoriville. Esimerkiksi seuraava avaa tiedoston nimeltä */etc/fstab*:

```
$ vim /etc/fstab
```

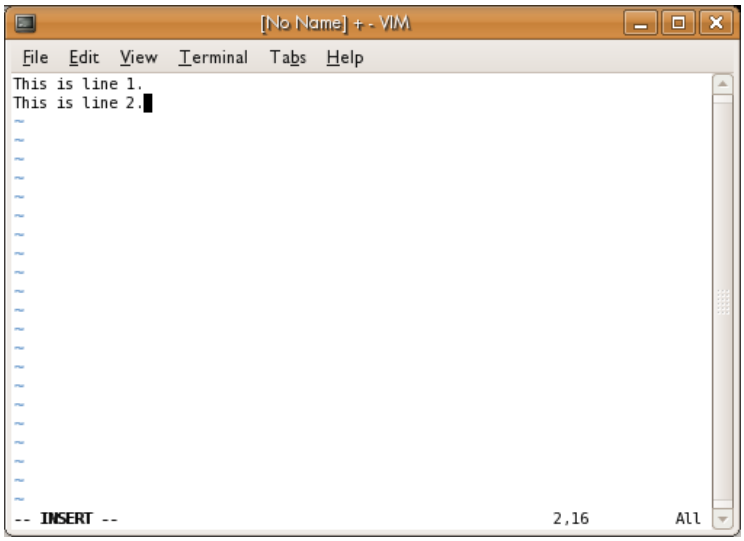
Tämä tiedosto on jo olemassa useimmissa GNU/Linux -järjestelmissä, mutta jos avaat olemattoman tiedoston, saat tyhjän ruudun. Seuraava osa näyttää sinulle, kuinka tekstiä lisätään; kun olet valmis, voit sitten tallentaa tiedoston.

Tekstin lisääminen

Olitpa sitten tyhjässä ruudussa tai tiedostossa, jossa on tekstiä, voit lisätä tekstiä *muokkaustilassa*. Paina vain **i**. Sinun pitäisi nähdä tämä ruudun alaosassa:

```
-- INSERT --
```

Kun tämä näkyy ruudun alaosassa, olet muokkaustilassa. Mitä tahansa kirjoitatkin tulee osaksi tiedostoa. Yritä esimerkiksi kirjoittaa "This is line 1." Paina sitten näppäintä **Enter** ja kirjoita "This is line 2". Tältä tämä ällistyttävä lisä maailmankirjallisuuteen näyttää Vimissä:



Kun olet lopettanut tekstin lisäämisen, paina näppäintä **Esc** (Escape) päästäksesi pois muokkaustilasta; tämä laittaa sinut normaaliin tilaan.

Vain yhdessä paikassa samaan aikaan: kursori

Vim, kuten mikä tahansa tekstieditori, pitää kirjaa sijainnistasi ja näyttää kohdistimen siinä paikassa. Se voi näyttää alleviivaukselta tai eriväriseltä laatikolta. Editointitilassa voit käyttää poistonäppäintä poistamaan merkkejä. Vim tarjoaa myös mahdollisuuden siirtyä ympäriinsä käyttäen nuolinäppäimiä ja editoida koko dokumenttia. Normaali Vi ei anna sinun liikkua ympäriinsä; se rajoittaa sinut lisäämään tekstiä tai poistamaan sitä.

Normaalissa Vissä, jos tahdot mennä takaisin muokkaamasi tekstin yli, tai siirtyä toiseen paikkaan tiedostossa, joudut painamaan näppäintä **Esc**, siirtymään takaisin paikkaan, johon tahdot lisätä tekstiä, ja siirtymään takaisin editointitilaan. Tämä voi vaikuttaa vaikealta. Mutta Vi tarjoaa niin monta tapaa siirtyä ympäriinsä ja lisätä tekstiä, että pienellä harjoittelulla huomaat, että se ei ole mikään este tuottavuudelle. Kuten aiemmin on mainittu, Vim antaa sinun liikkua ympäriinsä vapaasti ja muokata koko tiedostoa, kun olet editointitilassa - mutta huomaat lähteväsi usein editointitilasta käyttääksesi Vimin tehokkaita komentoja.

Perusliikekomennot

Harjoitellaksesi tiedostossa liikkumista, voit painaa näppäintä **Esc** päästäksesi pois editointitilasta. Jos sinulla on vain vähän tekstiä tiedostossa, voit tahtoa löytää toisen tiedoston järjestelmästäsi, joka on pidempi, ja avata sen. Muista että avatessasi tiedoston olet normaalissa tilassa, et editointitilassa.

Käytä nuolinäppäimiä siirtyäksesi ympäriinsä.

Hypätäksesi tietylle riville, paina kaksoispistettä ja kirjoita sen jälkeen rivinumero. Seuraava komento hyppää riville 20:

```
:20
```

Voit siirtyä nopeasti ylös ja alas tekstitiedostossa painamalla näppäimiä **PgUp** ja **PgDn**.

Etsi tekstiä painamalla kauttaviivaa (/) ja kirjoittamalla teksti, jonka tahdot löytää:

```
/birthday party
```

Voit yksinkertaisesti käyttää näppäintä **/** uudestaan etsiäksesi merkkijonon seuraavan toistumakerran. Etsiäksesi taaksepäin, käytä kysymysmerkkiä (?) kauttaviivan sijasta.

Tallentaminen ja poistuminen

Jos olet editointitilassa, voit tallentaa muutoksesi painamalla näppäintä **Esc** mennäksesi normaalitilaan, kirjoittaa **:w** ja painaa näppäintä **Enter**. Tämä tallentaa muutokset tiedostoon, joka on määritelty avatessasi Vimin.

> Huomaa: Vim ei oletusarvoisesti tallenna tiedoston alkuperäisen version varmuuskopioita. Komento **:w** poistaa kuitenkin vanhan sisällön. Vim voidaan kuitenkin konfiguroida tallentamaan varmuuskopiot.

Jos avasit Vimin määrittelemättä tiedostonimeä, saat virheviestin, kun painat näppäintä **:w**:

```
E32: No file name
```

Kirjataksesi tämän, määrittele komennolle **:w** tiedostonimi:

```
:w testitiedostoni.txt
```

Tämän jälkeen tarvitsee painaa näppäintä **Enter**.

Poistuaksesi Vimistä, painta **:q**. Jos sinulla on tallentamatonta tekstiä, saat virheviestin:

```
E37: No write since last change (add ! to override)
```

Kuten useimmat tekstieditorit, Vim yrittää varoittaa, mikäli olet tekemässä virheen, joka voi tuhota työsi. Kuten viesti huomauttaa, voit hylätä tekstisi ja poistua painamalla **:q!**. Tai käytä komentoa **:w** tallentaaksesi muutoksesi ja paina sitten uudestaan **:q!**. Voit yhdistää kirjoittamisen ja poistumisen millä tahansa seuraavista:

```
:wq (jonka jälkeen Enter)
:x (jonka jälkeen Enter)
ZZ
```

Hiiren käyttö Vimissä

Joskus on kätevää käyttää hiirtä Vimissä, jotta voit valita tekstiä ja siirtää nopeasti kursorin toiseen paikkaan. Saadaksesi tämän tilan käyttöön, anna Vimille komento:

```
:set mouse=a (jota seuraa Enter)
```

Useiden tiedostojen avaaminen samaan aikaan

Version 7 jälkeen Vimissä on ollut (harvoin huomattu) mahdollisuus käyttää välilehtiä, aivan kuin Firefoxissa tai muissa välilehtiä käyttävissä ohjelmissa.

Käyttääksesi välilehtiä Vimissä kirjoita komento :tabnew <tiedosto>. Esimerkiksi tiedosto foo.txt välilehteen tarvitaan komento:

```
:tabnew /path/to/foo.txt (jota seuraa Enter)
```

Siirtyäksesi edestakaisin tämän tiedoston ja sen tiedoston välillä, jonka parissa työskentelit viimeksi, käytä näppäimiä **g** ja sitten **t**. Voit avata välilehtiin niin monta tiedostoa kuin tahdot ja käyttää sitten komentoa **gt** siirtyäksesi niiden välillä. Jos olet laittanut päälle hiiren käytön (katso "Hiiren käyttö Vimissä" yläpuolella), voit yksinkertaisesti napsauttaa välilehteä.

Voit sulkea välilehden samoin kuin tekisit normaalille tiedostolle, käytä komentoa **:wq** tallentaaksesi muutokset, tai komentoa **:q** lopettaaksesi ilman tallennusta.

Kopioi, leikkaa ja liitä (nopea visuaalisen tilan esittely)

Vim käyttää tilaa jota kutsutaan *visuaaliseksi* valitsemaan tekstiä kopioitavaksi "puskuriin" (ajattele puskuria leikepöytänä), jota voidaan käyttää muualla.

Aktivoidaksesi visuaalisen tilan voit painaa näppäintä **v** ja painaa rivinvaihtoa. Kursorinäppäinten (nuolinäppäinten) avulla voit valita tekstin, jonka tahdot kopioida tai leikata.

Kopioidaksesi valitun tekstin, paina näppäintä **y**. Käytä nyt nuolinäppäimiä siirtyäksesi uuteen paikkaan tekstissäsi. Paina nyt näppäintä **p** laittaaksesi tekstin uuteen sijaintiin.

Leikataksesi valitun tekstin, paina näppäintä **x**. Siirry nyt uuteen sijantiin dokumentissasi ja paina **p** laittaaksesi tekstin siihen paikkaan.

Joskus on hyvä valita tekstiä riveissä tai pylväissä. Kokeile näitä visuaalisen tilan toimintoja painamalla näppäimiä **SHIFT+v** valitaksesi kokonaisia rivejä tai **CTRL+v** valitaksesi pylväitä. Ennen kuin huomaatkaan voit työskennellä yhtä nopeasti kuin työskentelisit hiirellä tekstinkäsittelyohjelmassa!

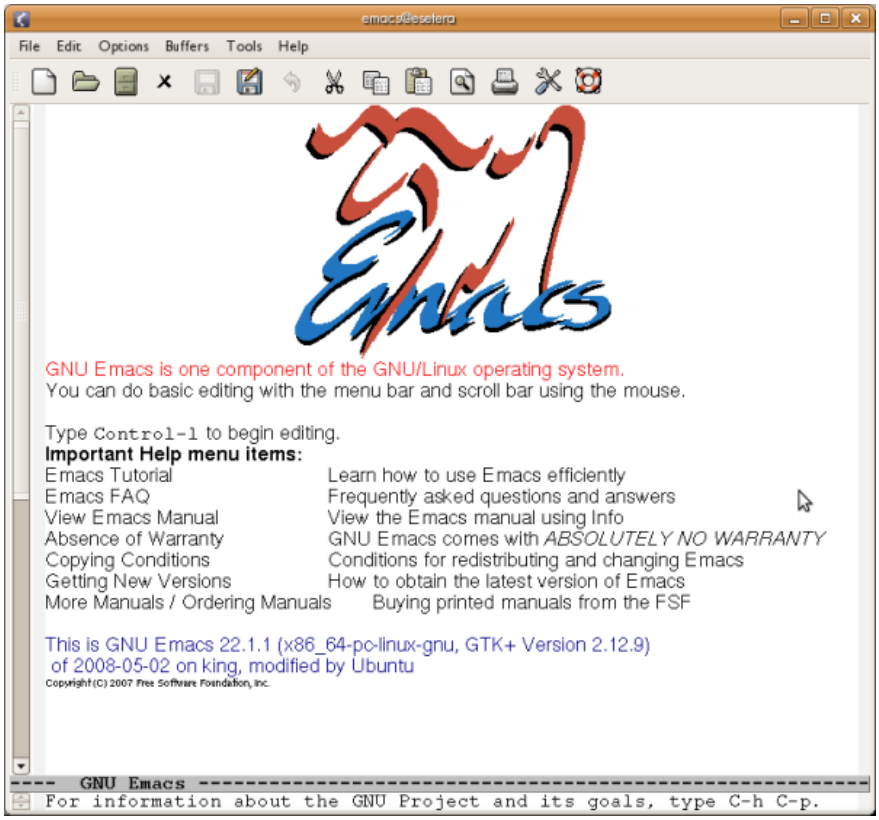
Harjoittele!

26. EMACS

Emacs on hyvin tehokas tekstieditori. Voit käynnistää Emacsin kirjoittamalla sen nimen komentoriville.

```
$ emacs
```

Jos käytät tyypillistä graafista GNU/Linux-jakelua, tämä komento avaa uuden ikkunna, jossa Emacs pyörii.



Emacs on editori, joka on niin tehokas, että monet käyttäjät avaavat Emacsin tietokoneen avattuaan ja jättävät sen päälle koko istunnon ajaksi. Jos tahdot pitää Emacsin päällä pidemmän aikaa, on hyödyllistä suorittaa Emacs taustalla, jotta komentorivi vapautuu toiselle komennolle.

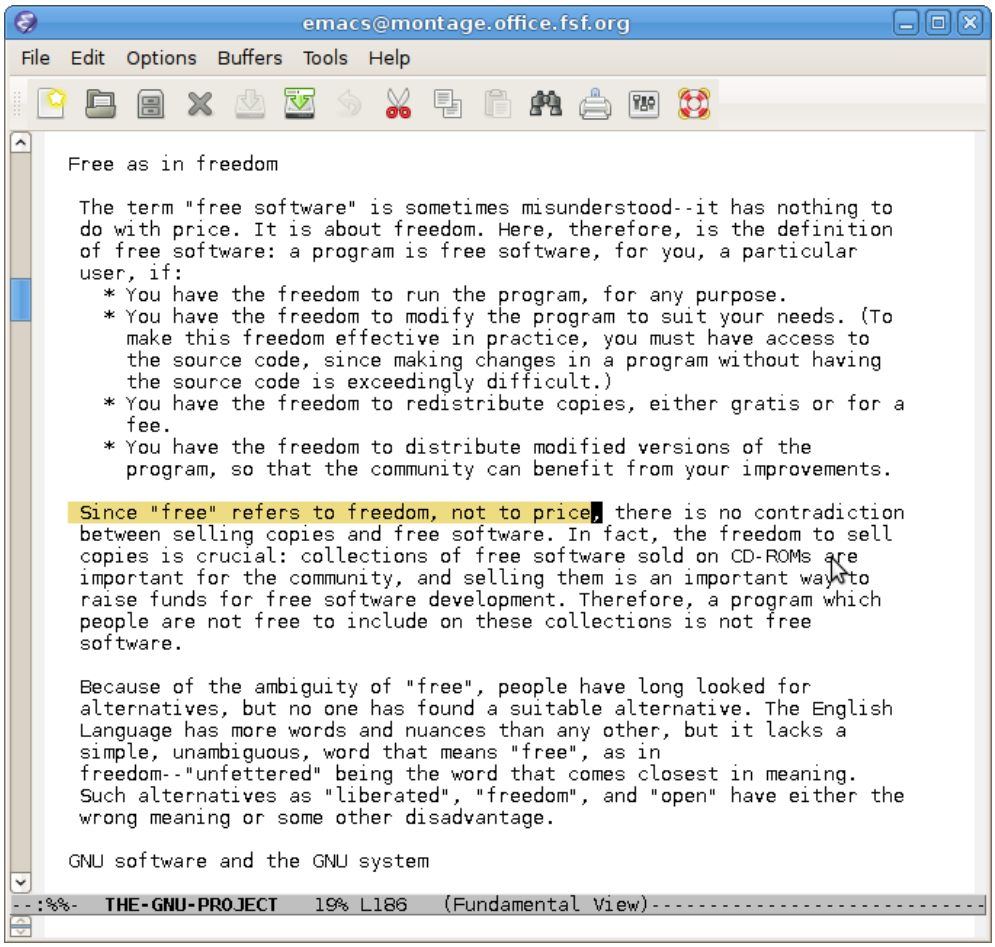
```
$ emacs &
```

Joskus tahdot ehkä ajaa Emacsin suoraan pääteikkunassa. Käytä silloin valitsinta *-nw* (ei ikkunaa). You

```
$ emacs -nw
```

Voit ladata tiedoston editoitavaksi käynnistäessäsi Emacsin antamalla tiedoston nimen *emacs*-komennon jälkeen.

```
$ emacs tiedostonimi
```



PERUSEDITOINTIKOMENNOT

Kun Emacs on päällä, on joukko peruseditointikomentoja, joita voit käyttää. Suurimmassa osassa tästä kirjasta käytämme merkintätapaa **ctrl + x** osoittamaan, että on tarpeen painaa **Ctrl** (Control) -näppäintä, ja sen jälkeen näppäintä **x** samanaikaisesti **Ctrl** -näppäimen ollessa pohjassa, jonka jälkeen molemmat näppäimet vapautetaan. Tässä Emacs-luvussa käytämme Emacs-dokumentaation merkintätapaa, joka lyhentää näppäimet **ctrl + x** muotoon **C-x**.

C-x C-f (lataa tiedosto puskuriin)

Komento **C-x C-f** (paina **Ctrl**-näppäintä, paina ja vapauta **x**, paina ja vapauta **f**, vapauta **Ctrl**) lataa tiedoston levyltä Emacsin *puskuriin* (Emacsin työalue) editointia varten. Sinulta kysytään ladattavan tiedoston nimeä. Voit sen jälkeen tehdä muutoksia puskuriin kirjoittamalla ja käyttämällä muita Emacsin komentoja. Tätä puskuria ei tallenneta levylle, ennen kuin erikseen pyydät sitä, esimerkiksi komennolla **C-x C-s**.

C-x C-s (tallenna tiedosto puskuriin)

Komento **C-x C-s** tallentaa nykyisen Emacs-bufferin levylle nykyisenä nimettynä tiedostona. Tiedoston nimi on palkissa ikkunan alaosassa.

C-x C-c (poistu Emacsista)

Tämä komento poistuu Emacsista. Jos on jäljellä tallentamattomia puskureita, Emacs kysyy tahdotko tallentaa ne.

C-h t (aloittaa Emacs-oppaan)

Komento **C-h t** (paina näppäintä **Ctrl**, paina ja vapauta **h**, vapauta **Ctrl**, paina ja vapauta **t**) aloittaa Emacs-oppaan. Tämä opas esittelee Emacsin peruskomentoja askel askeleelta.

C-h ? (yleinen ohje)

Tämä komento tarjoaa joukon ohjevaihtoehtoja.

C-k (tuhoa rivi)

Komento **C-k** tuhoaa (poistaa) nykyisen rivin nykyisessä puskurissa kursorista rivin loppuun asti.

C-y (palauta rivi)

Tämä komento palauttaa viimeksi poistetun rivin tai joukon rivejä ja liimaa sen kursorin nykyiseen sijaintiin.

MUITA EMACSIN OMINAISUUKSIA

Emacsissa on erilaisia tiloja erilaisten yleisten ja harvinaisten tiedostotyyppien muokkaamiseen. Näitä on tekstille, komentotulkkiskripteille, Python-skripteille ja niin edelleen. Jokainen tila määrittelee uudelleen esimerkiksi tabulaattorinäppäimen painalluksen vaikutukset, jotta se toimii sopivimmalla tavalla kyseiselle tiedostotyypille. Nämä tilat käynnistyvät automaattisesti tietyille tiedostotyypeille. Nämä tilat käynnistyvät automaattisesti monille tiedostotyypeille, riippuen tiedoston päätteestä tai tiedoston ensimmäisestä rivistä.

Emacsia voi laajentaa. Voit ohjelmoida sen käyttäytymään kuten tahdot, esimerkiksi käyttämällä sisäänrakennettua, helposti käytettävää skriptikieltä Emacs Lisp. Katso lisätietoa Emacs-dokumentaatiosta.

EMACS-DOKUMENTAATIO

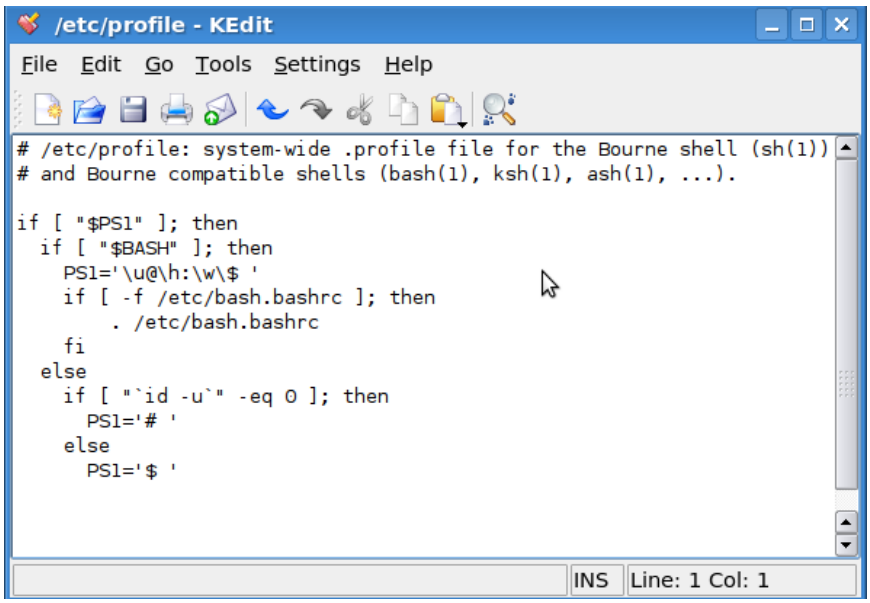
Emacs on dokumentoitu hyvin ilmaisissa lähteissä. Kirjoita `info emacs` komentorivillä (tai **C-h r** Emacsin sisältä) lukeaksesi koko virallisen dokumentaation. On myös lyhennetty opassivusto (kirjoita `man emacs` komentorivillä). Aloittelijoille paras tapa Emacsin oppimiseen on aiemmin mainittu interaktiivinen opas.

27. KEDIT, KWRITE JA KATE

Vaikka Kedit on osa KDE-ohjelmistokokoelmaa, johon kuuluu KDE-työpöytäympäristö, se ei tarvitse KDE:ta toimiakseen. Se toimii yhtä hyvin Gnomessa. KDE:ssä on joukko sisäänrakennettuja työkaluja, jotka auttavat sinua editoimaan tekstitiedostoja (myös skriptejä). Yksinkertaisin näistä on Kedit, yksinkertainen perustekstieditori. Voit käynnistää sen KDO-hakemistosta tai komentoriviltä, jos tahdot. Esimerkiksi voisit suorittaa komennon:

```
$ kedit /etc/profile &
```

Näkisit jotain tällaista.



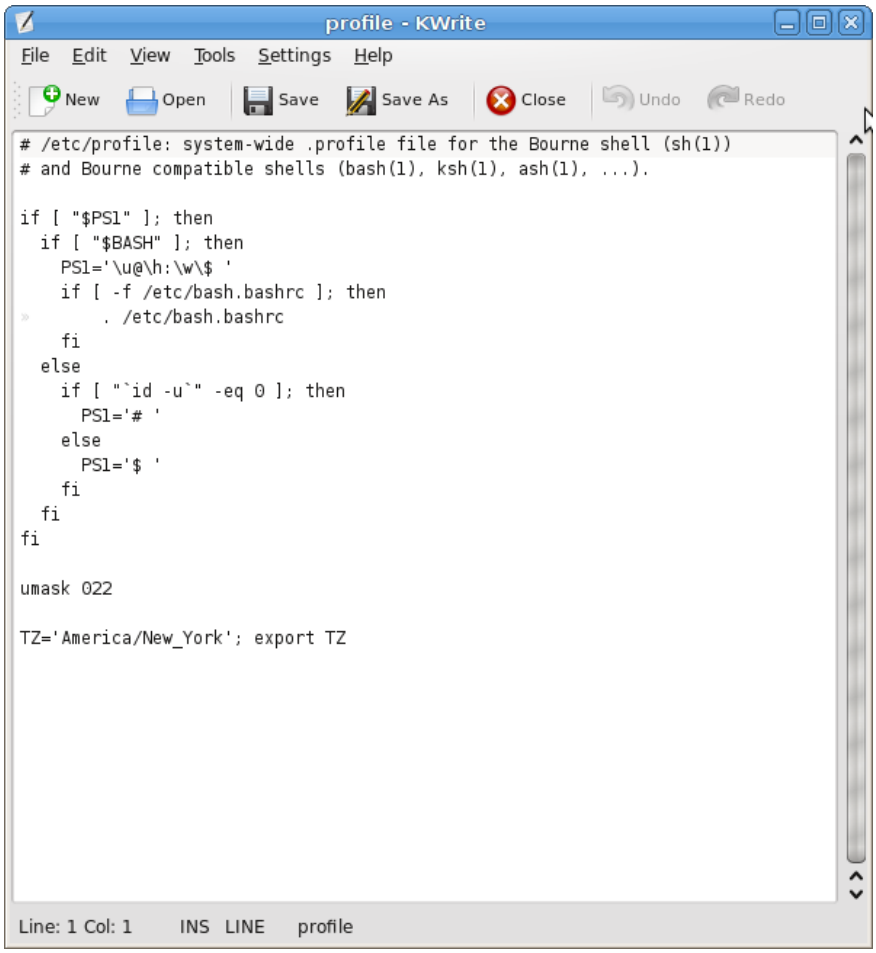
Sen käyttö on helppoa. Voit siirtyä ympäri tiedstoa nuolinäppäimillä, sekä näppäimillä Page Up (**PgUp**) ja Page Down (**PgDn**), tai hiirellä. Uuden tiedoston avaaminen tapahtuu valikosta **File->Open** ja voit oikolukea tiedostosi valikosta **Tools->Spelling**...

Ikkunan alalaidassa on hyödyllistä tietoa (jos et näe sitä, **Settings->Show Statusbar** laittaa sen näkyville). Rivi ja palkkinäyttö näyttää kohdistimen nykyisen sijainnin. "INS" merkitsee, että olet *insert* -tilassa, ja että jos oikealle kohdistimesta on tekstiä, se siirtyy eteenpäin, kun kirjoitat. Tämän vastakohta on "OVR", joka merkitsee *overtype* -tilaa, jossa teksti oikealle kursorista korvataan uudella tekstillä. Näiden välillä voit siirtyä näppäimistön **Insert** -näppäimellä.

Jos teet muutoksia nykyiseen tiedostoon, teksti "[modified]" näkyy otsikkopalkissa muistuttamassa, että joudut tallentamaan muutoksen ennen poistumista.

KWRITE

Vaikka KEdit on käyttökelpoinen, on se melko rajoittunut. KDE tarjoaa muitakin kokeilemisen arvoisia vaihtoehtoja. Kwrite näyttää melkein samalta, mutta tarjoaa lisäominaisuuksia.

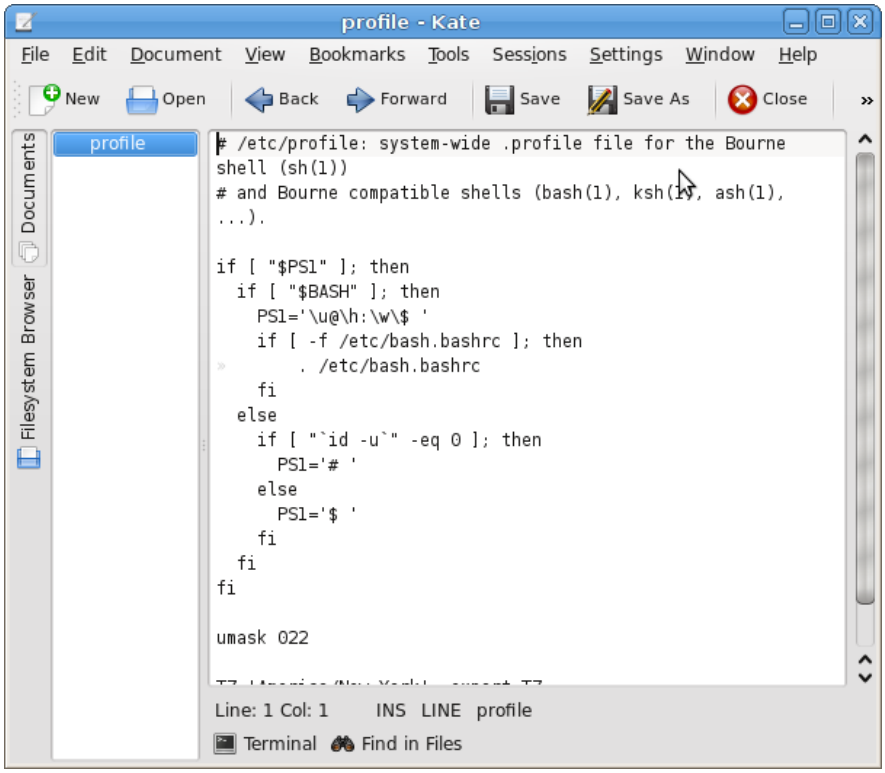


Selkein etu on *syntaksin korostaminen*. Komentotulkin skripteille, ohjelmille ja monen muun tyyppisille tiedostoille Kwrite värittää tekstin tehdäkseen sen lukemisesta helpompaa. Tässä kommentit näytetään harmaalla, parametrit vihreällä, Bash-komennot tummanpurppuralla ja muut komennot vaaleanpurppuralla. Yleensä KWrite yrittää valita korostuksen perustuen siihen, minkä tyyppisen tiedon se arvelee olevan kyseessä. Jos se arvaa väärin tai ei käytä korostusta ollenkaan, valitse vaihtoehto valikosta **Tools->Highlighting**.

Olet ehkä myös huomannut pienet miinusmerkit ja rivit vasemmassa marginaalissa. Tämä osa tunnetaan *koodin taittona*. KWrite yrittää sovittaa "if"-komennot vastaavien "fi"-komentojen kanssa, "for"-komennon komennon "done" kanssa, ja niin edelleen. Miinusmerkin napsauttaminen romahduttaa osan, mikä voi olla hyödyllistä, jos luet skriptiä, ja olet kiinnostunut katselemaan mitä on ennen blokkia ja sen jälkeen, mutta et blokin sisältöä. Kwritessä on monta hyödyllistä ominaisuutta - katso valikoita ja tutki, mitä löydät!

KATE

Viimeinen tässä katsauksessa on Kate. Se on periaatteessa sama editori kuin KWrite. Se tarjoaa kuitenkin lisätyökaluja, jotka helpottavat projektin kanssa työskentelyä, kun on kyse useammista tiedostoista eikä vain yhdestä.



Ikkunan vasemmalla puolella on välilehtiä, jotka antavat sinun lukea nykyisessä Kate-istunnossa avoimena olevia dokumentteja (KEdit ja KWrite avaavat useammat istunnot eri ikkunoissa, mutta Kate voi avata ne kaikki yhdessä ikkunassa), tai navigoida tietokoneen tiedostojärjestelmän läpi avatakseen tiedoston. Kate korostaa myös syntaksin kuten KWrite, mutta lisää myös "Pääte"-välilehden alaosaan. Tämän välilehden napsauttaminen avaa ja sulkee minipäätteen, jossa voit kirjoittaa komentoja. Jos tahdot tässä tapauksessa nähdä, mitä "id -u" skriptissä tekee, voit vain kirjoittaa sen päätteeseen nähdäksesi, mitä tapahtuu.

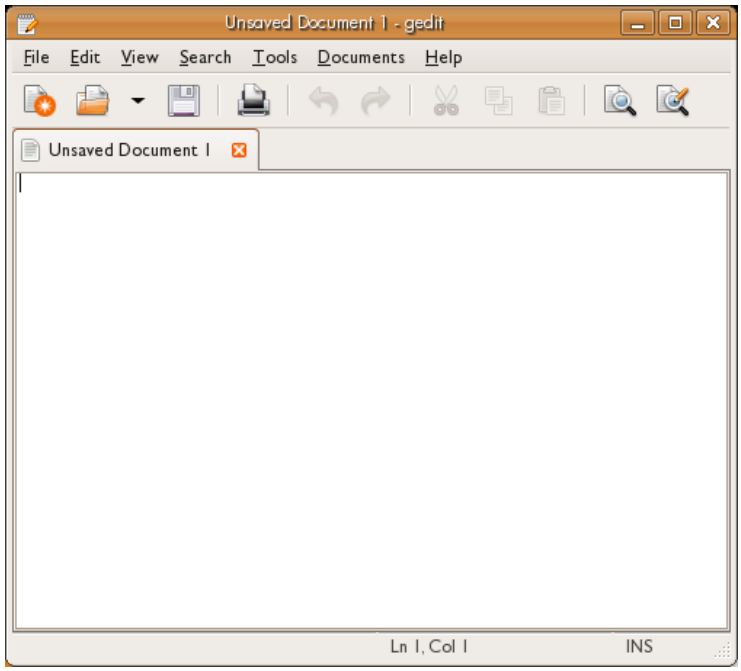
KDE-käyttäjille KEdit, KWrite ja Kate tarjoavat kolme mukavaa vaihtoehtoa tekstitiedostojen muokkaamiseen. Luultavasti kaikki kolme ovat asennettuja valmiiksi mihin tahansa KDE-järjestelmään. Pidä hauskaa niitä kokeillessasi!

28. GEDIT

Gedit, oletusarvoinen graafinen editori Gnomea käyttäessäsi, toimii myös KDE:ssä ja muilla työpöydillä. Useimmat gNewSense ja Ubuntu -asennukset käyttävät Gnomea oletusarvoisesti. Avataksesi Geditin avaa pääte ja kirjoita:

```
$ gedit &
```

Tämän pitäisi tulla näkyviin:



Tämä näyttää useimpien käyttöjärjestelmien peruseditorille. Voit käyttää ohjelmaa Gedit graafisen käyttöliittymän avulla, komennot ovat yksinkertaisia:

File->Open: Avaa olemassaolevan tiedoston

File->New: Luo uuden (tyhjän) tiedoston

File->Save: Tallentaa tiedoston

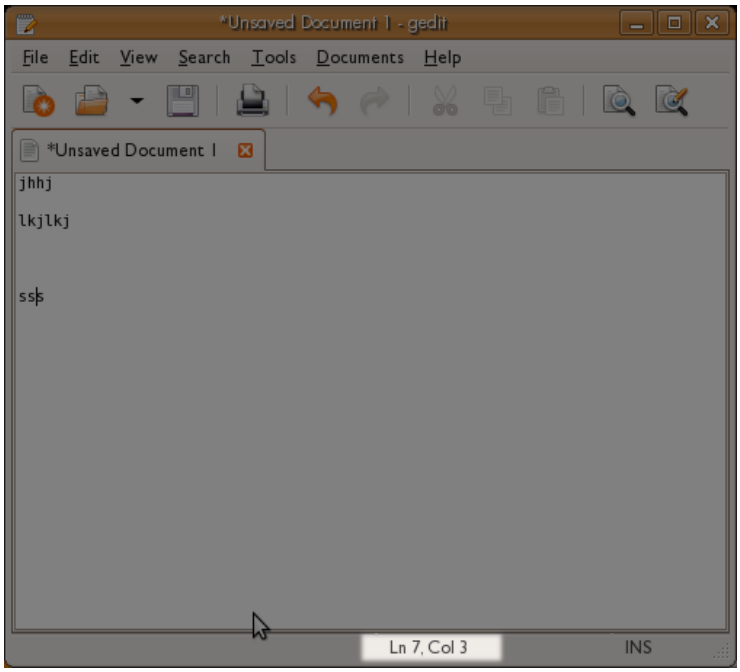
ctrl + c: Kopioi

ctrl + v: Liittää

Tässä on kaikki tarpeellinen. Lisätäksesi tekstiä voit vain kirjoittaa!

RIVINUMEROT

Gedit seuraa kohdistimen sijaintia ja näyttää sijainnin käyttöliittymän alaosassa:



Tämä voi olla hyödyllistä tietoa tiedettäväksi. Jos pidät kirjaa rivinumeroista voit käyttää näitä hyppiäksesi nopeasti tekstitiedostossa käyttäen ominaisuutta "Go to Line". Tätä voidaan käyttää käyttöliittymän kautta (**Search -> Go to Line**) tai näppäinkomennolla **ctrl** + **i**.

SKRIPTAUS

29. SKRIPTAAMINEN
30. SKRIPTIEN YLLÄPITÄMINEN
31. MUUT SKRIPTAUSKIELET
32. SED-TEKSTIOHJELMOINTIKIELI
33. AWK
34. SÄÄNNÖLLISET LAUSEKKEET
35. PERL
36. RUBY
37. PYTHON

29. SKRIPTAAMINEN

Jos sinulla on kokoelma komentoja, jotka tahtoisit suorittaa yhdessä, voit yhdistää ne skriptiin eli komentosarjaan, ja suorittaa ne kaikki kerralla. Voit myös antaa skripteille argumentteja, jotta se voi toimia eri tiedostojen tai muun sisääntulon pohjalta.

Kuten elokuvan käsikirjoitusta lukeva näyttelijä, tietokone lukee jokaisen komennon komentotulkkisi komentosarjassa, odottamatta sinua kuiskaamaan jokaista komentoa sen korvaan. Skripti on helppo tapa:

- Välttyä kirjoittamasta komentoja, jotka ajaisit usein sarjassa.
- Muistaa monimutkaisia komentoja, jotta et joudu etsimään, tai unohda, tiettyä syntaksia joka kerta sitä käyttäessäsi.
- Käyttää hallintarakenteita, kuten silmukoita ja tapauslauseita, jotta skriptisi voivat tehdä monimutkaisia toimenpiteitä. Näiden rakenteiden kirjoittaminen skriptiin tekee niistä mukavampia kirjoittaa ja helpompia lukea.

Sanotaanpa, että sinulla on kokoelma kuvia (esimerkiksi digitaalikamerasta) ja tahdot tehdä niistä esikatselukuvia. Avattuasi satoja kuvia kuvankäsittelyohjelmassasi päätät tehdä homman nopeasti komentoriviltä. Ja koska joudut ehkä tekemään saman homman tulevaisuudessa joudut kirjoittamaan komentosarjan. Tällöin esikatselukuvien tekeminen vaatii sinua kirjoittamaan vain kaksi komentoa:

```
$ cd kuvat/digikamera/lomakuvat_maaliskuu_2009
$ tee_esikatselukuvat.sh
```

Seuraava komento, `tee_esikatselukuvat.sh`, on skripti, joka tekee työt. Se voisi näyttää suunnilleen tältä:

```
#!/bin/bash
mkdir esikatselu
cp *.jpg esikatselu
cd esikatselu
mogrify -resize 400x300 *.jpg
```

Ensimmäinen rivi ei ole pakollinen. Ensimmäinen ristikkomerkki (#) muuttaa sen kommentiksi, jonka pääte jättää huomiotta ajaessaan skriptiä. Se on kuitenkin käyttökelpoinen rivi, sillä ajaessasi ohjelman rivi osoittaa, että /bin/bash -ohjelma pitäisi kutsua ajamaan skripti. Tämä rivi on myös hyödyllistä dokumentaatiota kaikille, jotka ihmettelevät mitä tiedostossa on.

Jokainen seuraavista riveistä on komento. Olemme nähneet kolme niistä aiemmin: mkdir, cp ja cd. Viimeinen komento, mogrify, on ohjelma, joka voi muuttaa kuvien kokoa (ja tehdä sen lisäksi paljon muita asioita). Lue sen opassivu oppiaksesi siitä lisää.

SKRIPTIEN MUUTTAMINEN SUORITETTAVIKSI

Tehdäksesi skriptin jollaisen juuri näytimme, avaa suosikkitekstieditorisi ja kirjoita komennot, jotka tahtoisit suorittaa. Voit kirjoittaa monta komentoa yhdelle riville, kunhan laitat puolipisteen jokaisen komennon jälkeen, jotta pääte tietää, että uusi komento alkaa.

Tallenna skripti. Yksi yleinen käytäntö on .sh -loppupäätteen käyttö - esimerkiksi *tee_esikatselukuvat.sh*.

On vielä yksi askel ennen kuin voit suorittaa skriptin: sen täytyy olla suoritettava tiedosto. Muista, että käyttöoikeuksia antaessasi oikeus ajaa tiedosto on yksi oikeuksista, joka tiedostolla voi olla, joten voit tehdä skriptistä ajettavan antamalla oikeuden execute (x). Seuraava komento antaa kenelle tahansa käyttäjälle oikeuden ajaa skripti:

```
chmod +x tee_esikatselukuvat.sh
```

Koska tahdot varmaankin käyttää skriptiä usein, voi olla hyvä idea tarkastaa PATH ja lisätä skripti yhteen sen hakemistoista (esimerkiksi */home/jdoe/bin* on hyvä valinta tässä näytetyn PATH-tiedon perusteella).

```
$ echo $PATH
/usr/bin:/usr/local/bin:/home/jdoe/bin
```

Yksinkertaista testaamista varten voit suorittaa skriptin näin, jos olet hakemistossa, joka sisältää sen:

```
$ ./tee_esikatselukuvat.sh
```

Mihin tarvitset edeltävää polkua ./? Koska useimmat käyttäjät eivät ole asettaneet nykyistä hakemistoaan PATH -ympäristömuuttujiinsa. Voit lisätä sen, mutta monet käyttäjät pitävät sitä turvallisuusriskinä.

Voit myös suorittaa skriptin, ilman että sen suorituskäyttöoikeus olisi asetettu, antamalla sen parametrinä komentotulkille:

```
bash tee_esikatselukuvat.sh
```

LISÄÄ HALLINTAA

Komentotulkki tarjoaa mahdollisuuden tehdä valintoja skriptissä ja suorittaa komennot monta kertaa eri sisääntuloilla. Tässä suhteessa komentotulkki on oikeastaan ohjelmointikieli ja mukava tapa tutustua ohjelmointikielen tarjoamiin tehokkaisiin ominaisuuksiin. Näytämme tässä perusteet komentotulkin tarjoamista hallintalauseista.

if

Tämä komento esiteltiin jo virheiden tarkistusta käsittelevässä osassa, mutta tarkastelemme sitä uudestaan tässä. if tarkoittaa jos ja se toimii aikalailla kuten olettaisit, joskin sen syntaksi on hieman erilainen kuin muissa kielissä. Se noudattaa tätä rakennetta:

```
if [ testiehto ]
then
  tee jotain
else
  tee jotain muuta
fi
```

Luet oikein: rakenne täytyy lopettaa avainsanalla *fi*. Osa *else* on vaihtoehtoinen. Varmista, että jätät tilaa aloitus- ja lopetushakasulkujen sisälle, muuten *if* ilmoittaa syntaksivirheestä.

Jos tahdot tutkia voitko lukea tiedoston, voit kirjoittaa tällaisen ohjelman:

```
if [ -r /home/joe/salatieto.txt ]
then
    echo "Voit lukea tiedoston"
else
    echo "Et voi lukea tiedostoa!"
```

if tarjoaa joukon erilaisia tarkastuksia. Voit laittaa minkä tahansa joukon komentoja *testiehdoksi*, mutta useimmat *if*-lauseet käyttävät hakasulkusyntaksin tarjoamia testejä. Nämä ovat itse asiassa synonyymi komennolle *test*. Joten ensimmäinen rivi edellisessä esimerkissä olisi yhtä hyvin voitu kirjoittaa seuraavasti:

```
if test -r /home/joe/salatieto.txt
```

Tietoja testeistä kuten *-r* löytyy komennon *test* opassivulta. Kaikkia testioperaattoreita voidaan käyttää hakasuluissa, kuten me olemme tehneet.

Joitain hyödyllisiä *test* -operaattoreita ovat:

-r	Tiedosto on luettava
-x	Tiedosto on suoritettava
-e	Tiedosto on olemassa
-d	Tiedosto on olemassa ja hakemisto

Näitä on paljon enemmänkin, ja voit jopa testata useampia ehtoja samaan aikaan. Katso komennon *test* opassivua.

WHILE (JA UNTIL)

while on silmukkarakenne. Se käy kierroksia läpi, kunnes sen testiehdot eivät enää ole totta. Se ottaa seuraavan muodon:

```
while testiehto
do
  askel1
  askel2
  ...
done
```

Voit luoda myös silmukoita, jotka toimivat, kunnes käyttäjä keskeyttää ne. Tässä on yksi tapa (ehkei kuitenkaan paras mahdollinen) katsoa kuka on kirjautunut järjestelmääsi 30 sekunnin välein:

```
while true
do
    who
    sleep 30
done
```

Tämä ei ole tyylikästä, koska käyttäjän täytyy painaa CTRL-C tai tappaa ohjelma jollain muulla tavalla. *Break*-komennolla voit kirjoittaa silmukan, joka loppuu kohdatessaan ehdon. Esimerkiksi seuraava skripti käyttää komentoa *read* (joka on erittäin hyödyllinen interaktiivisissa skripteissä) lukeakseen rivin syötettä käyttäjältä. Tallennamme syötteen muuttujaan *userinput* ja tarkastamme sen seuraavalla rivillä. Skripti käyttää jo näkemäämme *if*-komentoa *while*-blokin sisällä, mikä antaa meidän päättää koska lopetamme *while*-blokin. Komento *break* päättää *while*-blokin ja jatkaa kohti skriptin loppua (jota emme näe tässä). Huomaa, että käytämme kahta testiä valitsimella *-o*, mikä merkitsee "tai". Käyttäjä voi kirjoittaa Q joko isoin tai pienin kirjaimin lopettaakseen ohjelman.

```
while true
do
  echo "Kirjoita käsiteltävä syöte (kirjoita Q lopettaaksesi)"
  read userinput
```

```
  if [ $userinput == "q" -o $userinput == "Q" ]
  then
    break
  fi

  käsittele syöte...

done
```

until toimii täsmälleen samalla tavalla, paitsi että silmukka toimii, kunnes testiehto on totta.

CASE

case on skriptin tapa vastata pieneen joukkoon ehtoja. Se toimii samalla tavalla kuin muiden ohjelmointikielten *case*-lauseet, tosin sillä on oma omituinen syntaksinsa, joka esitellään parhaiten esimerkin avulla.

```
user=`whoami` # laittaa skriptin suorittavan käyttäjän käyttäjänimen
              # muuttujaan $user.
case $user in
    joe)
        echo "Hei Joe. Tiedän että tahtoisit tietää kellonajan, joten
näytän sen alla."
        date
        ;;
    amy)
        echo "Hyvää päivää, Amy. Tässä on tehtävälistasi."
        cat /home/amy/amy-todo.txt
        ;;
    sam|tex)
        echo "Hei hemmo. Älä unohda katsoa järjestelmäkuormitusta.
Nykyinen järjestelmäkuormitus on:"
        uptime
        ;;
    *)
        echo "Tervetuloa, kuka tahansa oletkin. Töihin siitä."
        ;;
esac
```

Jokaista tapausta täytyy seurata merkki), jonka jälkeen tulee uusi rivi, jonka jälkeen lista otettavista askelista, sitten puolipiste tuplana (;;). Ehtoon "*)" sopii kaikki, aivan kuin *default* -avainsanaan joidenkin kielten *case*-rakenteissa. Jos muut tapaukset eivät sovi, tämä lista komentoja suoritetaan. Lopulta avainsana *esac* lopettaa *case*-lauseen. Huomaaa, että näytetyssä esimerkissä on tapaus, joka sopii sekä merkkijonoon "sam" että "tex".

FOR

for on käyttökelpoinen tapa käydä läpi listaa. Se voi olla mikä tahansa merkkijonojen lista, mutta se on erityisen hyödyllinen käytäessä läpi tiedostolistaa. Seuraava esimerkki käy läpi kaikki tiedostot hakemistossa *myfiles* ja luo jokaisesta varmuuskopion. (Se tukehtuisi hakemistoihin, mutta pidetäänpä esimerkki yksinkertaisena, eikä testata onko tiedosto hakemisto.)

```
for filename in myfiles/*
do
    cp $filename $filename.bak
done
```

Kuten mikä tahansa muuttujan luova komento, ensimmäinen komento *for*-blokissa asettaa muuttujan nimeltä **filename** ilman dollarimerkkiä.

30. SKRIPTIEN YLLÄPITÄMINEN

Siirryt vähitellen ohjelmoinnin puolelle shelliskriptauksen kautta. Nyt on paras aika oppia hyvän ohjelmoijan tapoja. Koska tämä kirja on vain johdanto komentoriviin, käsittelemme muutamia tärkeitä vinkkejä, jotka pyörivät *ylläpidettävyyden* ympärillä.

Kun ohjelmoijat puhuvat ylläpidettävyydestä, he puhuvat helppoudesta, jolla ohjelmaa voidaan muutella, olipa sitten kyse virheiden korjauksesta, uusien toimintojen lisäämisestä, tai toimivuuden parantamisesta. Heikosti huollettavat ohjelmat on helppo huomata: niistä puuttuu rakenne, joten toiminnallisuus on levinnyt ympäriinsä. Kun painat tästä, se sekoaakin tuolta, mikä on todellinen painajainen. Yleisesti ottaen niitä on todella vaikea lukea. Ajattele vaikkapa tätä:

```
#!/bin/sh
identify `find ~/Kuvat/Loma/2008 -name \*.jpg` | cut -d ' ' -f 3 |
sort | uniq -c
```

Käytä suosikkieditoriasi tallentaaksesi tämän tiedoston nimellä *foo*, sitten:

```
$ chmod +x foo
$ ./foo
     11 2304x3072
     12 3072x2304
```

Tämä pieni hirviö etsii tietyn hakemiston tiedostot, jotka loppuvat päätteeseen ".jpg", ajaa komennon `identify` niille kaikille, ja raportoi sellaista tietoa, joka on jonkun mielestä ollut jossain vaiheessa erittäin hyödyllistä. Jos ohjelmoija olisi vain lisännyt jotain vihjeitä ohjelman suorittamista toimenpiteistä...

ÄLÄ KÄYTÄ PITKIÄ RIVEJÄ

Ensinnäkin huomaat, että esimerkissämme heikosti ylläpidettävä ohjelma on yksi pitkä rivi. Näin ei tarvitse olla. Entäpä, jos ohjelma näyttäisikin tältä:

```
#!/bin/sh
identify `find ~/Kuvat/Loma/2008 -name \*.jpg` |
cut -d ' ' -f 3 |
sort |
uniq -c
```

On hieman helpompaa havaita missä kukin komento alkaa ja loppuu. Se on edelleenkin sama kokoelma putkitettuja ohjelmia, mutta ne on esitetty eri tavalla. Voit katkaista pitkät rivit putkien kohdalta, mutta niiden toiminnallisuus on silti sama.

Voit myös jakaa yhden komennon useampaan riviin käyttämällä \ -merkkiä rivin lopussa liittämään sen seuraavaan riviin:

```
#!/bin/sh
echo Tämä \
     on \
     oikeasti \
     yksi \
     pitkä \
     komento.
```

NIMEÄ SKRIPTISI KUVAAVILLA NIMILLÄ

Toinen asia, jonka voit havaita, on skriptin nimi "foo". Tämä on lyhyt ja kätevä, mutta se ei tarjoa mitään vinkkiä ohjelman käyttötarkoitukseen. Entäpä tämä:

```
$ mv foo listaa_kuvien_koot
```

Nyt nimi auttaa käyttäjää ymmärtämään mitä skripti tekee. Paljon parempi, vai mitä?

KÄYTÄ MUUTTUJIA

Eräs ongelma ohjelmassa on sen tarkemerkkien (`) käyttö. Se toki toimii, mutta siinä on myös ongelmia. Ehkäpä suurin ongelma on se, jota ei ole helppo huomata: muista, että tarkemerkit laittavat sisältämänsä komennon ulostulon siihen kohtaan ohjelmassa, jossa ne ovat. Joissain ohjelmissa on rajoitettu komentorivin pituus. Tässä tapauksessa määritellyssä hakemistossa voi olla paljon kuvia, jolloin komentorivistä voi tulla epätavallisen pitkä. Tämä aiheuttaa omituisen virheviestin, kun ajat ohjelman. On monia tapoja korjata tämä ongelma, mutta tässä voimme kokeilla seuraavaa:

```
#!/bin/sh
find ~/Kuvat/Loma/2008 -name \*.jpg |
while read image ; do identify $image ; done |
cut -d ' ' -f 3 |
sort |
uniq -c
```

Nyt `find` toimii samoin kuin ennen, mutta sen ulostulo, tiedostonimien lista, putkitetaan while-silmukkaan. Tämän silmukan ehto on `read image`. `read` on funktio, joka lukee rivin kerrallaan, jakaa sen sisääntulon kenttiin ja laittaa jokaisen kentän muuttujaan, joka on tässä tapauksessa *image*. Nyt `identify` toimii kuva kerrallaan.

Huomaa, kuinka muuttujan lisääminen tekee ohjelmasta helpommin luettavan: se sanoo kirjaimellisesti, että tahdot identifioida kuvan. Huomaa lisäksi, kuinka vaikutus tuleviin ohjelmoijiin ei olisi ollut sama, jos muuttujan nimi olisi ollut vaikkapa *ovi* tai *cdrom*. Nimet ovat tärkeitä!

Mutta tässä ohjelmassa on vieläkin jotain häiritsevää: hakemiston nimi hohtaa kuin kipeä peukalo. Mitä jos muutataisimme ohjelman tällaiseksi:

```
#!/bin/sh
ALOITUS_HAKEMISTO=~/Kuvat/Loma/2008

find $ALOITUS_HAKEMISTO-name \*.jpg |
while read image ; do identify $image ; done |
cut -d ' ' -f 3 |
sort |
uniq -c
```

Tämä on hieman parempi: nyt voit muokata skriptiäsi ja muuttaa hakemistoa joka kerta, kun tahdot käsitellä jonkun toisen.

KÄYTÄ PARAMETREJA

Loppu ei tuntunut aivan oikealta. Kuitenkaan et muuta komentoa `ls` aina, kun tahdot listata eri hakemiston sisällön, ethän? Tehdäänpä ohjelmastamme yhtä joustava:

```
#!/bin/sh
ALOITUS_HAKEMISTO=$1
```

```
find $ALOITUS_HAKEMISTO -name \*.jpg |
while read image ; do identify $image ; done |
cut -d ' ' -f 3 |
sort |
uniq -c
```

Muuttuja *$1* on ensimmäinen parametri, jonka annat skriptillesi ($0 on ajamasi skriptin nimi). Nyt voit kutsua skriptiäsi näin:

```
$ ./listaa_kuvien_koot ~/Kuvat/Loma/2008
```

Tai voit listata vuoden 2007 kuvat, jos tahdot:

```
$ ./listaa_kuvien_koot ~/Kuvat/Loma/2007
```

TIEDÄ MISTÄ ALOITTAA

Ajattele, mitä tapahtuu, jos ajat skriptin näin:

```
$ ./listaa_kuvien_koot
```

Ehkäpä tahdot tämän, mutta ehkäpä et. Tapahtuu niin, että *$1* on tyhjä, joten *$ALOITUS_HAKEMISTO* on myös tyhjä ja ensimmäinen etsittävä parametri on myös tyhjä. Tämä merkitsee, että *find* etsii nykyisen työhakemistosi. Tahdot ehkä tehdä tästä käytöksestä näkyvää:

```
#!/bin/sh
if test -n "$1" ; then
    ALOITUS_HAKEMISTO=$1
else
    LOPETUS_HAKEMISTO=.
fi

find $ALOITUS_HAKEMISTO -name \*.jpg |
while read image ; do identify $image ; done |
cut -d ' ' -f 3 |
sort |
uniq -c
```

Ohjelma toimii täsmälleen samoin kuin ennenkin, erona on vain se, että kun katsot ohjelmaa seuraavan kerran kuukauden päästä, et joudu arvailemaan, miksi se antaa tuloksia, vaikka et anna sille hakemistoa parametrinä.

KATSO ENSIN

Entäpä jos annat skriptille parametrin, mutta parametri ei ole hakemisto, tai sitä ei edes ole olemassa? Kokeile.

Ei kovinkaan kaunista?

Mitäs jos teemme näin:

```
#!/bin/sh
if test -n "$1" ; then
    ALOITUS_HAKEMISTO=$1
else
    ALOITUS_HAKEMISTO=.
fi

if ! test -d $ALOITUS_HAKEMISTO ; then
    exit
fi

find $ALOITUS_HAKEMISTO -name \*.jpg |
while read image ; do identify $image ; done |
cut -d ' ' -f 3 |
sort |
uniq -c
```

Toimii paremmin. Nyt skripti ei edes yritä toimia, jos sen saama parametri ei ole hakemisto. Se ei ole kovin kohteliasta, *se poistuu hiljaisesti* ilman aavistustakaan siitä, mikä meni mynkään.

VALITA TARVITTAESSA

Se korjataan helposti:

```
#!/bin/sh
if test -n "$1" ; then
    ALOITUS_HAKEMISTO=$1
else
    ALOITUS_HAKEMISTO=.
fi

if ! test -d $ALOITUS_HAKEMISTO; then
    echo \"$ALOITUS_HAKEMISTO\" ei ole hakemisto tai sitä ei ole
olemassa. Loppu."
    exit
fi

find $ALOITUS_HAKEMISTO -name \*.jpg |
while read image ; do identify $image ; done |
cut -d ' ' -f 3 |
sort |
uniq -c
```

OLE VAROVAINEN POISTUESSASI

Ohjelma tuottaa nyt virheviestin, jos et anna sille parametrinä olemassaolevaa hakemistoa, ja poistuu ilman jatkotoimenpiteitä. Olisi mukavaa, jos antaisit muiden ohjelmien, jotka ehkä jatkossa kutsuvat ohjelmaasi, tietää että tuloksena oli virheviesti. Ohjelman olisi siis hyvä poistua virhekoodilla. Tähän malliin:

```
#!/bin/sh
if test -n "$1" ; then
    ALOITUS_HAKEMISTO=$1
else
    ALOITUS_HAKEMISTO=.
fi

if ! test -d $ALOITUS_HAKEMISTO; then
    echo \"$ALOITUS_HAKEMISTO\" ei ole hakemisto tai sitä ei ole
olemassa. Loppu.
    exit 1
fi

find $ALOITUS_HAKEMISTO -name \*.jpg |
while read image ; do identify $image ; done |
cut -d ' ' -f 3 |
sort |
uniq -c
```

Nyt, jos kohdataan virheviesti, skriptisi poistumiskoodi on 1. Jos ohjelma poistuu normaalisti, poistumiskoodi on 0.

KÄYTÄ KOMMENTTEJA

Mikä tahansa merkkiä **#** samalla rivillä seuraava jätetään huomiotta, joten voit kommentoida skriptisi toimintaa. Esimerkiksi:

```
#!/bin/sh
# Tämä skripti raportoi kaikkien JPEG-tiedostojen koot nykyisessä
hakemistossa
# tai parametrinä annetussa hakemistossa, sekä jokaisen koon kuvien
lukumäärän.

if test -n "$1" ; then
    ALOITUS_HAKEMISTO =$1
else
```

```
    ALOITUS_HAKEMISTO=.
fi

if ! test -d $ALOITUS_HAKEMISTO ; then
    echo \"$ALOITUS_HAKEMISTO\" ei ole hakemisto tai sitä ei ole
olemassa. Loppu.
    exit 1
fi

find $ALOITUS_HAKEMISTO -name \*.jpg |
while read image ; do identify $image ; done |
cut -d ' ' -f 3 |
sort |
uniq -c
```

Kommentit ovat hyviä, mutta älä kirjoita niitä liikaa. Yritä kirjoittaa ohjelma niin, että koodi itsessään on selkeää. Syy tähän on yksinkertainen: ensi vuonna, kun joku muu muuttaa skriptiäsi, tuo toinen henkilö voisi hyvinkin muuttaa komentoja ja unohtaa kommentit, mikä tekee jälkimmäisesti harhaanjohtavaa. Ajattele tätä:

```
# laske kolmeen
for n in `seq 1 4` ; do echo $n ; done
```

Kumpi se on? Kolme vai neljä? Ilmeisesti ohjelma laskee neljään, mutta kommentin mukaan se laskee kolmeen. Voisit ottaa sen kannan, että ohjelma on oikeassa ja kommentti on väärässä. Mutta entäpä, jos ohjelman kirjoittanut henkilö tahtoi laskea kolmeen ja tämä on syy siihen, miksi kommentti on olemassa? Yritetäänpä näin:

```
# On kolme pikku porsasta
for n in `seq 1 3` ; do echo $n ; done
```

Kommentti dokumentoi syyn, jonka vuoksi ohjelma laskee kolmeen: se ei kuvaa ohjelman toimintaa, se kuvaa, mitä ohjelman tulisi tehdä. Kokeillaanpa erilaista lähestymistapaa:

```
PORSAITA =3
for pig in `seq 1 $PORSAITA` ; do echo $porsas ; done
```

Sama tulos, hieman erilainen ohjelma. Jos uudelleenmuotoilet ohjelmasi, voit tehdä niin ilman kommentteja (sivuhuomautuksena hieno sana tämän kaltaisille muutoksille, joita olemme tehneet, on uudelleenfaktorointi, mutta se on tämän kirjan laajuuden ulkopuolella).

VARO TAIKANUMEROITA

Nykyisessä ohjelmassamme on taikanumero, numero joka saa ohjelman toimimaan, mutta kukaan ei tiedä miksi sen täytyy olla juuri tuo numero. Se on taikuutta!

```
...
cut -d ' ' -f 3 |
...
```

Sinulla on kaksi vaihtoehtoa: kirjoita kommentti ja dokumentoi miksi sen täytyy olla "3" eikä "2" tai "4", tai lisää muuttuja, joka selittää sen nimellään. Kokeillaanpa jälkimmäistä:

```
#!/bin/sh
# Tämä skripti raportoi kaikkien nykyisen hakemiston tai parametrinä
annetun
# hakemiston alla olevien JPEG-tieodstojen koon ja jokaisen koon
kuvien määrän.

if test -n "$1" ; then
    ALOITUS_HAKEMISTO=$1
else
    ALOITUS_HAKEMISTO=.
fi
```

```
if ! test -d $ALOITUS_HAKEMISTO ; then
    echo \"$ALOITUS_HAKEMISTO\" ei ole hakemisto tai sitä ei ole
olemassa. Loppu.
    exit 1
fi

KUVAN_KOKO=3

find $ALOITUS_HAKEMISTO -name \*.jpg |
while read image ; do identify $image ; done |
cut -d ' ' -f $KUVAN_KOKO |
sort |
uniq -c
```

Se parantaa noita asioita hieman; ainakin nyt tiedämme mistä 3 tulee. Jos ImageMagick muuttaa joskus ulostulonsa formaattia, voimme päivittää skriptiä vastaavasti.

TOIMIKO SE?

Lopuksi muttei viimeiseksi, tarkasta ajamiesi komentojen poistumisstatus. Nykyisessä esimerkissämme ei ole paljon mahdollisuuksia epäonnistumiseen. Kokeillaanpa vielä viimeistä esimerkkiä:

```
#!/bin/sh
# Kopioi kaikki HTML- ja kuvatiedostot lähdehakemistosta annettuun
kohdehakemistoon.

SRC=$1
DST=$2

if test -z "$SRC" -o -z "$DST" ; then
    cat<
```

Huomaa, että tässä esimerkissä käytetään monia asioita, jotka olet oppinut tässä kirjassa. Se ei yritä olla täydellinen; voit harjoitella parantelemalla sitä!

Sinun tulisi huomata, että ohjelma kiinnittää huomiota virhetiloihin, joita eri ohjelmat voivat tuottaa. Esimerkiksi se ei vain kutsu komentoa `mkdir` nähdäkseen toimiko ohjelma, vaan se tekee näin:

```
if ! mkdir -p "$DST" ; then
    echo Ei voi luoda päämäärätiedostoa \"$DST\".  Stop.
    exit 1
fi
```

Se kutsuu komentoa `mkdir` ehtona komennolle `if`. Jos `mkdir` kohtaa virheen, se poistuu statuksella, joka on jotain muuta kuin 0, ja `if` -ehto tulkitsee sen epätodeksi tilaksi. "!" on negaatio-operaattori, ja muuttaa sen epätoden todeksi (tai päinvastoin). Niinpä rivi sanoo periaatteesa "aja komento *mkdir*, muuta virhe arvoksi tosi operaattorilla '!', toimi mikäli virhe on olemassa." Lyhyesti sanottuna, mikäli mkdir kohtaa virheen, ohjelmavirta menee *if*-lauseen vartaloon. Tämä voisi tapahtua esimerkiksi mikäli käyttäjällä, joka ajaa skriptin, ei ole oikeuksia luoda pyydettyä hakemistoa.

Huomaa myös merkkien "&&" käyttö varmistamaan virhetilat:

```
mkdir -p "$DST/$dir" && cp -a "$filename" "$DST/$filename"
```

`Jos mkdir` epäonnistuu, komentoa `cp` ei kutsuta. Edelleenkin, mikäli joko *mkdir* tai *cp* epäonnistuu, poistumisstatus on jotain muuta kuin 0. Tuo ehto tarkastetaan seuraavalla rivillä:

```
if test $? -ne 0 ; then
```

Koska tämä voisi osoittaa jonkin epäonnistuvan hirvittävällä tavalla (esimerkiksi levy on täynnä), meidän on paras luovuttaa ja lopettaa ohjelma.

YHTEENVETO

Skriptien kirjoittaminen on taidetta. Sinusta tulee parempi taiteilija tarkastelemalla edeltäjiesi työtä ja tekemällä paljon itse. Toisin sanottuna: lue paljon skriptejä ja kirjoita itse paljon skriptejä.

Hauskaa hakkerointia!

31. MUUT SKRIPTAUSKIELET

Komentotulkki on ihmeellinen ystävä. Jos olet lukenut koko kirjan tähän asti, voit olla huumaantunut sen tarjoamista mahdollisuuksista. Mutta komentotulkki on silti hyvin rajoittunut verrattuna moniin kieliin. Annamme sinulle vain maistiaisen muista työkaluista ja kielistä, joita voit tutkia.

Kaksi perinteistä työkalua, AWK ja Sed, kutsutaan yleensä komentotulkista. Molemmat tekevät syötteelle toimenpiteitä rivi kerrallaan. Voit ajatella niitä kokoonpanolinjoina, joille työläiset lastaavat tiedoston rivi riviltä. Jokainen rivi käsitellään järjestyksessä. Nämä ovat klassisia esimerkkejä suodattimista, mikä on hyvin läheisesti GNU/Linuxiin liittyvä käsite. Suodattimet solmitaan yhteen piippuihin | -merkillä, jolloin jokaisen komennon ulostulosta tulee seuraavan komennon syöte.

Seuraava osa käsittelee säännöllisiä lausekkeita. Ne ovat oma kielensä, mutta voit tehdä paljon oppimalla muutaman yksinkertaisen ominaisuuden. Ne ilmaantuvat jatkuvasti eri paikoissa: vi-tyylisissä tekstieditoreissa, *grep*-komennossa, AWK:ssa ja Sedissä sekä kaikissa seuraavissa kielissä.

Skriptauskielet keksittiin tekemään ohjelmoinnista helppoa ja mahdollistamaan sovellusten luominen nopeasti. Toisin kuin AWK ja Sed, ne toimivat yleensä yksinään, eivät osana piippuja tai muita ympäristöjä, joissa ne vain luovat rivin ulostuloa jokaista sisään tulevaa riviä kohden. Komentotulkkiin verrattuna näiden kielten etuina ovat:

- Monipuolinen laskenta kokonais- ja liukuluvuilla
- Olio-ohjelmointi, jonka avulla voit pitää tiedon yhdessä sitä manipuloivien funktioiden kanssa
- Monimutkaisia tietorakenteita, jotka voivat varastoida niihin liittyvää erityyppistä tietoa

Tässä kirjassa on lyhyet luvut kolmesta ohjelmistomaailman tämän hetken suosituimmasta skriptauskielestä: Perlistä, Pythonista ja Rubystä. Kohtaat monia työkaluja ja tuotteita, jotka mahdollistavat laajennukset näiden kielien avulla.

32. SED-TEKSTIOHJELMOINTIKIELI

Sed (sanoista "stream editor", virrankäsittely) on hyötyohjelma, joka tekee muutoksia rivi kerrallaan. Sille annetut komennot ajetaan jokaisella sisääntulon rivillä vuorollaan. Se on hyödyllinen sekä tiedostojen käsittelyyn että prosessin ulostulon putkittamiseen muihin ohjelmiin, kuten tässä:

```
$ wc -c * | sort -n | sed ...
```

PERUSSYNTAKSI JA KORVAUS

Yleinen tapa käyttää Sediä on sanojen vaihtaminen tiedoston sisällä. Olet voinut käyttää "etsi ja korvaa" -toimintoa graafisissa tekstinkäsittelyohjelmissa. Sed voi tehdä tämän paljon tehokkaammin ja nopeammin:

```
$ sed "s/hölö/pölö/g" sisääntulotiedosto > ulostulotiedosto
```

Tutkitaanpa tätä yksinkertaista tiedostoa. Ensin kerromme komentotulkin ajaa komento `sed`. Tahtomamme käsittely on lainausmerkkien sisällä; palaamme tähän hetken päästä. Sen jälkeen kerromme Sedille *sisääntulotiedoston* ja käytämme komentotulkin uudelleensuuntausmerkkiä (>) nimetäksemme *ulostulotiedoston*. Voit määritellä tahtoessasi monta sisääntulotiedostoa; Sed käsittelee ne järjestyksessä ja luo niistä yhden ulostulovirran.

Tämä ilmaisu näyttää monimutkaiselta, mutta se on hyvin yksinkertainen, kunhan opit ymmärtämään sitä. Alussa oleva "s" merkitsee "korvaa" (englanniksi "substitute"). Tämä seuraa teksti, jonka tahdot löytää ja korvaava teksti, vinoviivat (/) toimivat erotusmerkkeinä. Niinpä Sedin täytyy löytää "hölö" sisääntulotiedostossa ja laittaa "pölö" sen tilalle. Vain ulostulotiedosto muuttuu; Sed ei koskaan muuta sisääntulotiedostoja.

Lopussa oleva "g" tarkoittaa "globaalia", mikä merkitsee, että tämä operaatio tehdään koko riville. Jos jätät pois merkin "g" ja "hölö" näkyy kaksi kertaa samalla rivillä, ainoastaan ensimmäinen "hölö" muutetaan muotoon "pölö".

```
$ cat testitiedosto
hölö pölö hölö pölö hölö pölö hölö pölö
$ sed "s/hölö/pölö/g" testitiedosto > muutettutestitiedosto
$ cat muutettutestitiedosto
pölö pölö pölö pölö pölö pölö pölö pölö
```

Kokeillaanpa tätä uudestaan ilman merkkiä `/g` komennossa ja katsotaan mitä tapahtuu.

```
$ cat testitiedosto
hölö pölö hölö pölö hölö pölö hölö pölö
$ sed "s/hölö/pölö/" testitiedosto > muutettutestitiedosto
$ cat muutettutestitiedosto
pölö pölö hölö pölö hölö pölö hölö pölö
```

Huomaa, että ilman merkkiä "g" Sed teki korvauksen vain ensimmäiseen löytämäänsä sopivaan sanaan jokaisella rivillä.

Tämä kaikki on hienoa, mutta entäpä jos tahtoisit muuttaa testitiedostossa vasta toisen sanan hölö? Muuttaaksesi tietyn sanan ilmentymän joudut määrittelemään numeron korvauskomentojen jälkeen.

```
$ sed "s/hölö/pölö/2" sisääntulotiedosto > ulostulotiedosto
```

Voit myös yhdistää tämän lippuun g (joissain Sedin versiossa) jättääksesi ensimmäisen ilmentymän entiselleen ja muuttaaksesi kaikki ilmentymät toisesta viimeiseen riviin asti.

```
$ sed "s/hölö/pölö/2g" sisääntulotiedosto > ulostulotiedosto
```

SED-ILMAISUT SELITETTYNÄ

Sed ymmärtää säännöllisiä lauseita, joita käsittelee kokonainen luku tässä kirjassa. Tässä on joitain erikoismerkeistä, joita voit käyttää sopimaan siihen, mitä tahdot korvata.

```
$ sopii rivin loppuun
^ sopii rivin alkuun
* sopii nollaan tai useampaan kappaleeseen edellistä merkkiä
[ ] kaikki hakasulkujen sisällä olevat merkit sovitetaan
```

Voisit esimerkiksi muuttaa sanat "cat", "can" ja "car" sanaan "dog" tekemällä seuraavan:

```
 $ sed "s/ca[tnr]/dog/g" sisääntulotiedosto > ulostulotiedosto
```

Poista mikä tahansa numerosarja. Ensinnäkin [0-9] varmistaa, että tarvitaan ainakin yksi numero sovitettavaksi. Seuraava [0-9] voidaan poistaa tai se voi olla peräkkäin miten tahansa monta kertaa, koska sitä seuraa metamerkki *. Lopulta numerot poistetaan, koska toisen ja kolmannen kauttaviivan välillä ei ole mitään, vaikka voisit laittaa siihen korvaavan tekstin.

```
 $ sed "s/[0-9][0-9]*//g" sisääntulotiedosto >
ulostulotiedosto
```

Jos ensimmäinen merkki ilmaisun siällä on tarke (^), Sed sovittaa ainoastaan, mikäli teksti on rivin alussa.

```
$ echo dogs cats and dogs | sed "s/^dogs/doggy/"
doggy cats and dogs
```

Dollarimerkki merkkijonon lopussa käskee Sedin sovittaa teksti ainoastaan, mikäli se on rivin lopussa.

```
$ echo dogs cats and cats | sed "s/cats$/kitty/"
dogs cats and kitty
```

Rivi muuttuu vain, jos sovitettava merkkijono on siellä, missä vaadit sen olevan, jos sama teksti on jossain muualla lauseessa, sitä ei muuteta.

POISTAMINEN

Komento "d" poistaa kokonaisen rivin, jossa on sopiva merkkijono. Toisin kuin komento "s" komento "d" ottaa koko rivin.

```
$ cat testitiedosto
line with a cat
line with a dog
line with another cat
$ sed "/cat/d" testitiedosto > uusitestitiedosto
$ cat uusitestitiedosto
line with a dog
```

Säännöllinen lause ^$ merkisee "sovita rivi, jossa ei ole mitään alun ja lopun välissä", toisin sanottuna, tyhjä rivi. Voit poistaa kaikki tyhjät rivit käyttäen komentoa "d" tuon säännöllisen lauseen kanssa:

```
$ sed "/^$/d" sisääntulotiedosto > ulostulotiedosto
```

TULOSTUKSEN HALLINTA

Oletetaanpa, että tahdot tulostaa joitain rivejä ja jättää loput tulostamatta. Sen sijaan että poistaisit rivit komennolla "d" voitkin määritellä, mitkä rivit pidetään.

Tämä voidaan tehdä kahdella ominaisuudella:

Määrittele valitsin `-n`, joka merkitsee "älä tulosta rivejä oletusarvoisesti".

Lopeta merkkijono merkillä "p" tulostaaksesi rivin, joka sopii merkkijonoon.

Näytämme tämän tiedoston, joka sisältää nimiä:

```
$ cat testitiedosto
Mr. Jones
Mrs. Jones
Mrs.
Lee
Mr. Lee
```

Olemme päättäneen käyttää aina "Ms" naisille, joten tahdomme muuttaa "Mrs." merkkijonoksi "Ms". Säännöllinen lauseke on:

```
 s/Mrs\./Ms/
```

ja tulostaaksemme ainoastaan muutetut rivit, kirjoitamme:

```
$ sed -n "s/Mrs\./Ms/p" testfile
```

MONTA HAHMOA

Sedille voidaan laittaa useampia kuin yksi operaatio samaan aikaan. Voimme tehdä tämän määrittelemällä jokaisen hahmon valitsimen `-e` jälkeen.

```
$ echo Gnus eat grass | sed -e "s/Gnus/Penguins/" -e
"s/grass/fish/"
Penguins eat fish.
```

MUOKKAUSTEN HALLINTA HAHMOILLA

Voimme myös olla tarkempia siitä, mihin riveihin hahmoja sovelletaan. Tarjoamalla hahmoa ennen operaatiota voit rajoittaa operaation riveihin, joilla on kyseinen hahmo.

```
$ cat testitiedosto
one: number
two: number
three: number
four: number
one: number
three: number
two: number
$ sed "/one/ s/number/1/" testitiedosto > muutettutestitiedosto
$ cat muutettutestitiedosto
one 1
two: number
three: number
four: number
one: 1
three: number
two: number
```

Komento `sed` sai tässä esimerkissä kaksi hahmoa. Ensimmäinen hahmo, "one", hallitsee yksinkertaisesti Sedin muuttamia rivejä. Toinen hahmo korvaa merkkijonon "number" numerolla "1" noilla riveillä.

Tämä toimii myös useampien hahmojen kanssa.

```
$ cat testitiedosto
one: number
two: number
three: number
four: number
one: number
three: number
two: number
$ sed -e "/one/ s/number/1/" -e "/two/ s/number/2/" \
 -e "/three/ s/number/3/" -e "/four/ s/number/4/" \
 < testitiedosto > muutettutestitiedosto
$ cat muutettutestitiedosto
one: 1
two: 2
three: 3
four: 4
one: 1
three: 3
two: 2
```

MUOKKAUSTEN HALLINTA RIVINUMEROIDEN AVULLA

Sen sijaan että määrittäisit hahmoja, jotka voivat toimia millä tahansa rivillä, voimme määritellä tarkan rivin tai rivejä muokattavaksi.

```
$ cat testitiedosto
even number
odd number
odd number
even number
$ sed "2,3 s/number/1/" < testitiedosto > muutettutestitiedosto
$ cat muutettutestitiedosto
even number
odd 1
odd 1
even number
```

Tämä pilkku toimii erottimena, joka käskee Sedin toimia vain riveillä kaksi ja kolme.

```
$ cat testitiedosto
even number
odd number
odd number
$ sed -e "2,3 s/number/1/" -e "1 s/number/2/" < testitiedosto
> muutettutestitiedosto
$ cat muutettutestitiedosto
even 2
odd 1
odd 1
```

Joskus et tiedä tarkalleen kuinka pitkä tiedosto on, mutta tahdot mennä määritellyltä riviltä tiedoston loppuun. Voit käyttää komentoa `wc` tai vastaavaa ja laskea rivien kokonaismäärän, mutta voit käyttää myös dollarimerkkiä ($) edustamaan viimeistä riviä:

```
$ sed "25,$ s/number/1/" < testitiedosto >
muutettutestitiedosto
```

Dollarimerkki $ on Sedin tapa sanoa: "rivin loppuun asti".

SED-KOMENTOJEN SKRIPTAAMINEN

Käyttämällä valitsinta `-f` Sed-komennossa, voit syöttää Sedille listan komennoista, joita se voi ajaa. Voit esimerkiksi laittaa seuraavat hahmot tiedostoon, jonka nimi on *sedkomennot*:

```
s/foo/bar/g
s/dog/cat/g
s/tree/houseg/
s/little/big/g
```

Voit tehdä tämän yhteen tiedostoon kirjoittamalla seuraavan komennon:

```
$ sed -f sedkomennot < sisääntulotiedosto > ulostulotiedosto
```

Jokaisen tiedostossa olevan komennon täytyy olla eri rivillä.

Sedin kaikkia ominaisuuksia ei voida kuvata tässä luvussa. Itse asiassa Sedistä on kirjoitettu kokonaisia kirjoja, verkossa on monia loistavia oppaita Sedin käyttöön.

33. AWK

AWK on tekstitiedon käsittelyyn suunniteltu kieli. Se on nimetty luojiensa Alfred **A**hon, Peter **W**einbergerin ja Brian **K**ernighanin mukaan. AWK on melko pieni kieli ja se on helppo oppia, mikä tekee siitä ihanteellisen työkalun nopeaan ja helppoon tekstinkäsittelyyn. Sen pääkäyttötarkoitus on tiedon erottaminen taulukkomaisesta sisääntulosta.

Koska AWK-kielellä kirjoitetut ohjelmat ovat yleensä aika pieniä, niitä kirjoitetaan yleensä suoraan komentoriviltä. Tietenkin suurempien skriptien tallentaminen tekstitiedostoina on myös mahdollista.

Seuraavissa kappaleissa esittelemme AWK:n perusteet kolmen yksinkertaisen esimerkin avulla. Niistä kaikki käsittelevät seuraavaa tekstitiedostoa (joka sisältää viisi kaikkien aikojen korkeinta pistetulosta Donkey Kong -videopelissä toukokuuhun 2009 mennessä):

```
1050200 Billy Mitchell 2007
1049100 Steve Wiebe 2007
895400 Scott Kessler 2008
879200 Timothy Sczerby 2001
801700 Stephen Boyer 2007
```

Tämä tiedosto on jaettu *kenttiin*. Ensimmäinen jokaisen rivin kenttä sisältää vastaavan pistemäärän, toinen ja kolmas kenttä sisältävät pistemäärän tehneen henkilön nimen, ja neljäs ja viimeinen kenttä joka rivillä sisältävät vuoden, jona pistemäärä saavutettiin. Voit kopioida ja liittää tekstin tekstitiedoston yläpuolelle ja nimetä sen vaikkapa tiedostoksi *highscores.txt,* jotta voit kokeilla seuraavia esimerkkejä.

ESIMERKKI 1

Sanotaanpa, että tahdomme tulostaa vain pistemäärät, jotka ovat suurempia kuin 1 000 000 pistettä. Lisäksi tahdomme ainoastaan pistemäärän saavuttaneiden ihmisten etunimet. AWK:lla tämä tieto saadaan näin:

```
$ awk '$1 > 1000000 { print $2, $1 }' highscores.txt
Billy 1050200
Steve 1049100
```

Kokeile!

Pieni komentorivillä syöttämämme AWK-ohjelma sisältää kaksi osaa:

1. Ensimmäinen osa ennen aaltosulkeita (*$1 > 1000000*) sanoo: "Tee tämä kaikille riveille, joiden arvo kentässä numero 1 on suurempi kuin 1 000 000."
2. Aaltosulkujen sisällä oleva osa (*print $2, $1*) sanoo: "Tulosta kenttä numero 2, jota seuraa kenttä numero 1."

Yhdistetty ohjelma sanoo: "Kaikista riveistä, joiden ensimmäisen kentän arvo on suurempi kuin 1 000 000, tulosta rivin toinen arvo, jota seuraa rivin ensimmäinen arvo." (Huomaa, että komentoriviltä syötetyt AWK-ohjelmat on yleensä suljettu heittomerkkeihin, jotta komentotulkki ei tulkitse niitä.)

Kuten olemme nähneet edellisessä esimerkissä, AWK-lauseen rakenne on seuraava:

```
rakenne { toiminta }
```

Ilmaisu *rakenne* tarkentaa ehdon, joka täytyy täyttää, jotta *toimenpide* suoritetaan. AWK-ohjelmat koostuvat mistä tahansa määrästä näitä lauseita. (Edellä esitelty ohjelma sisältää vain yhden lauseen.) AWK-ohjelma tekee periaatteessa seuraavat asiat:

1. Se lukee sisääntulon (esimerkiksi tiedoston tai tekstivirran standardisisääntulosta) rivi riviltä.
2. Jokaisella rivillä AWK tekee lauseet, joiden ehto/hahmo täyttyy.

Yksinkertaista, eikö totta?

ESIMERKKI 2

Katsotaanpa toista esimerkkiä:

```
$ awk '$4 == 2007 { print "Sija", NR, "-", $3 }'
highscores.txt
Sija 1 - Mitchell
Sija 2 - Wiebe
Sija 5 - Boyer
```

tämä ohjelma sisältää taas yhden lauseen, joka voidaan ilmaista näin: "Tulosta sana 'Sija' ja muuttujan 'NR' arvo, viiva ('-') ja kenttä numero 3 jokaiselta riviltä, jonka kentän numero 4 arvo on 2007."

Tämä pikku ohjelma tulostaa kaikkien vuonna 2007 ennätyksen tehneiden sukunimet ja heidän sijaintinsa korkeimpien tulosten listassa.

Kuinka AWK tietää kuinka hyvin ennätystuloksen tehneet ovat sijoittuneet? Kun taulukko tutkitaan, jokaisen korkeimmat pisteet tehneen pelaajan sija vastaa rivinumeroa. AWK voi päästä käsiksi jokaisen rivin numeroon sisäänrakennetun muuttujan NR (Number of Row, rivinumero) avulla. AWK:ssa on paljon hyödyllisiä sisäänrakennettuja muuttujia, jotka löydät sen dokumentaatiosta.

ESIMERKKI 3

Kolmas ja viimeinen esimerkki on hieman monnimutkaisempi kuin kaksi edellistä, sillä se sisältää kaikkiaan kolme AWK-lausetta:

```
$ awk 'BEGIN {print "Viisi parasta Donkey Kongin pelaajaa ovat
saavuttaneet yhteensä:"} {total += $1} END {print total,
"pistettä!"}' highscores.txt
```

Tämä tulostaa seuraavan:

```
Viisi parasta Donkey Kongin pelaajaa ovat saavuttaneet yhteensä:
4675600 pistettä
```

Jaetaanpa tämä ohjelma kolmeen osaan/lauseeseen (jotka olemme kirjoittaneet yhdellä komentorivillä):

Ensimmäinen lauseke

hahmo: BEGIN
toiminto: print "Viisi parasta Donkey Kongin pelaajaa ovat saavuttaneet yhteensä:"

Toinen lauseke

hahmo: ei mitään (= suorita aina *toiminto*)
toiminto: lisää kentän numero 1 arvo muuttujaan *total*

Kolmas lauseke

hahmo: END
toiminto: tulostaa muuttujan *total* arvon, jota seuraa merkkijono "pistettä"

Katsotaanpa nyt mitä tässä uudessa lyhyessä AWK-ohjelmassa on.

Ensinnäkin rakenteilla BEGIN ja END on erikoinen merkitys: BEGIN -komennon jälkeen tuleva toiminto suoritetaan ennen kuin mitään syötettä luetaan ja END-komennon jälkeen tuleva toiminto suoritetaan AWK:n lopetettua syötteen lukemisen.

Toisessa lauseessa voimme havaita, että AWK-lause ei tarvitse hahmoa, ainoastaan *toiminto* on vaadittu. Jos lause ei sisällä hahmoa, lauseen ehto on aina voimassa ja AWK suorittaa toiminnon jokaiselle sisääntulon riville.

Lopulta olemme käyttäneet omaa muuttujaamme ensimmäistä kertaa. Muuttuja on nimeltään *total*. AWK-muuttujia ei tarvitse julistaa erikseen: voit luoda uusia muuttujia yksinkertaisesti käyttämällä niitä. Esimerkkiohjelmassamme muuttujan *total* arvo alkaa nollasta ja nousee yhdellä jokaista syötteen riviä kohden. Operaattori += merkitsee "lisää oikealla oleva matemaattinen ilmaisu vasemmalla olevaan muuttujaan."

Kun kaikki sisääntulon rivit on luettu, *total* sisältää kaikkien kentän 1 arvojen summan, eli kaikkien huipputulosten summan. Lause END tulostaa muuttujan *total* arvon, jota seuraa merkkijono "pistettä.

MINNE MENEMME TÄÄLTÄ?

Olemme nähneet, että AWK on hauska ja helppokäyttöinen pikku ohjelmointikieli, jota on sovellettu suureen määrään tiedonuuttamistehtäviä. Tämä lyhyt johdanto AWK-ohjelmointikieleen ei tietenkään voi olla alkupalaa kummempi. Jos tahdot oppia enemmän, suosittelemme että katsot GAWKia, GNU-versiota AWKista. Se on yksi monipuolisimmista kielen versioista, sen mukana tulee laaja ja helppolukuinen käyttöopas (katso http://www.gnu.org/software/gawk/manual/).

34. SÄÄNNÖLLISET LAUSEKKEET

Kun yrität etsiä tekstiä tiedostoista tai muuttaa tekstiä, tarvitset usein epämääräisiä tai monimielisiä hakulausekkeita. Tyypillisiä hakuja ovat:

- Epämääräisen asioiden joukon löytäminen, kuten "yksi tai useampi nolla".
- Tekstistä on erilaisia vaihtoehtoja, kuten "väri" ja "värit" tai "rotta" ja "rotan".
- Tietyn hahmon muodostaman tekstin osan erottaminen. Esimerkiksi sinulla voi olla lista sähköpostiosoitteita, kuten joku@fsf.org tai joku@flossmanuals.net, ja tahdot erottaa sen osan, joka on @ -merkin jälkeen (fsf.org ja flossmanuals.net).

Etsiäksesi tällaisia merkkijonoja ja tehdäksesi niihin muutoksia, erityinen kieli, jota kutsutaan säännöllisiksi lausekkeiksi (englanniksi "regular expressions"), on todella hyödyllinen. Tämä osa tarjoaa nopean esittelyn säännöllisistä lausekkeista. Tämä kieli voi olla hieman uhkaava aluksi - mutta se ei ole monimutkainen, ainoastaan merkkien käytön kanssa tulee olla hyvin tarkkana. Sinun täytyy käyttää sitä jonkin aikaa, jotta aivosi tottuvat erilaisten säännöllisten lausekkeiden lukemiseen.

Helpoin tapa oppia ja harjoitella säännöllisiä lausekkeita on käyttää yksinkertaisia suodattimia, jotka tarjotaan komentotulkissa, kuten grep, Sed ja AWK. Komento grep on nähty jo muutamia kertoja tässä kirjassa. Tässä osassa käytämme "laajennettua" versiota, joka on nimeltään egrep, koska se tarjoaa enemmän ominaisuuksia, joita ihmiset käyttävät usein säännöllisissä lausekkeissa. Sed ja AWK esiteltiin aiemmissa luvuissa; käytämme tässä luvussa paljon Sediä.

Tämän osan loppuun mennessä ymmärrät komennot grep, egrep ja Sed melko hyvin. Voit sen jälkeen siirtyä eteenpäin ja käyttää säännöllisiä lausekkeita muissa tilanteissa, joissa ne ovat vieläkin tehokkaampia. Lähes jokaisessa nykyaikaisessa ohjelmointikielessä, joihin sisältyvät muut tässä oppaassa mainitut skriptauskielet Perl, Python ja Ruby, tarjotaan säännöllisiä lausekkeita. Jopa tietokannat tarjoavat säännöllisiä lausekkeita jossain muodossa.

Säännöllisten lausekkeiden yksityiskohdat vaihtelevat työkalusta toiseen, ja jopa työkalun yhdestä versiosta toiseen. Näytämme hyvin yleiset ominaisuudet, mutta ne eivät toimi hyvin kaikissa työkaluissa.

PELKKÄ TEKSTI

Säännöllisen lausekkeen ei tarvitse olla hieno. Tähän asti esittelemämme grep-komennot etsivät pelkkää tekstiä:

```
$ cat color_file
Primary colors blue and red make the color magenta
Primary colors blue and green make the colour cyan
Primary colors red and green make the colour yellow
Black and white make grey
$ grep 'colour' color_file
Primary colors blue and green make the colour cyan
Primary colors red and green make the colour yellow
```

Koska sanassa "colour" ei ole ohjausmerkkejä, jotka komentotulkki voisi tulkita, emme tarvitse heittomerkkejä, mutta käytämme niitä saadaksemme tavan iskostettua. Tässä luvussa käsitellyt egrep -komennot käyttävät paljon ohjausmerkkejä.

MÄÄRITTELEMÄTTÖMÄT JOUKOT

Yksi säännöllisten lausekkeiden yksinkertainen käyttötapa on etsiä "mikä tahansa määrä" jotain, tai sumea joukko, kuten "3-5" väliviivaa. Tässä ovat tätä tukevat ohjausmerkit. Yksinkertaisuuden vuoksi näytämme ne ensin erikseen, jonka jälkeen käytämme niitä joissain työkaluissa.

Sovita nolla tai useampi kappale merkkijonoa X	X*
Sovita yksi tai useampi kappale merkkijonoa X	X+
Sovita nollasta yhteen kappaletta merkkijonoa X	X?
Sovita 3-5 kappaletta merkkijonoa X	X{3,5}

Nämä muistuttavat paljon komentotulkin ohjausmerkkejä, mutta niissä on pieniä eroja. Keskity niiden merkitykseen säännöllisissä lausekkeissa, ja ota huomioon, että se tosiaan on komentotulkin ohjausmerkeistä eroava kieli.

Nyt voimme nähdä, kuinka voimme löytää sekä merkkijonot "color" että "colour" yhdessä haussa. Tarvitsemme joko 0 tai 1 kappaletta merkkiä "u", joten määritämme:

```
$ egrep 'colou?r' color_file
Primary colors blue and red make the color magenta
Primary colors blue and green make the colour cyan
Primary colors red and green make the colour yellow
```

Asteriski (*) on yksi yleisimmistä ja hyödyllisimmistä merkeistä säännöllisissä lausekkeissa, mutta se on myös yksi hämmentävimmistä ja väärinkäytetyimmistä. Oleta, että tahdot poistaa kaikki nollat riviltä. Voisit yrittää poistaa "minkä tahansa joukon nollia" asteriskilla:

```
$ echo "There are 40006 items" | sed "s/0*/X/"
```

Ulostuo on kuitenkin:

```
XThere are 40006 items
```

Tämä tapahtui, koska Sed korvaa vaatimasi hahmon ensimmäisen tapauksen. Pyydit "nolla tai enemmän" ja tällainen tapaus on heti rivin alussa!

Käytä tässä tapauksessa plus-merkkiä (0+), mutta monet Sedin versiot eivät tue sitä. Voit kiertää tämän rajoituksen näin:

```
$ echo "There are 40006 items" | sed "s/00*/X/"
There are 4X6 items
```

Jos laitat yhden merkin aaltosulkuihin, esimerkiksi {3}, se merkitsee "korvaa tämä numero tarkalleen". Jos lisäät mukaan pilkun ilman toista merkkiä {3,}, se merkitsee "sovita mikä tahansa numero, joka on kolme tai enemmän."

MÄÄRITTÄMÄTTÖMÄT SOVITUKSET, LUOKAT JA KANTAMAT

Sovittaaksesi mitä tahansa merkkiä, voit käyttää pistettä (.). Niinpä seuraava sopii kauttaviivaan, jota seuraa mikä tahansa merkki ja toinen kauttaviiva:

```
$ egrep '/./' file
```

Pistettä seuraa yleensä yksi sumeista määritteistä edellisestä kappaleesta. Niinpä seuraavat sopivat mihin tahansa merkkeihin kauttaviivojen välissä (merkkejä täytyy kuitenkin olla ainakin yksi):

```
$ egrep '/.+/' file
```

Seuraava on sama, mutta se löytää myös rivejä, joissa on kaksi kauttaviivaa peräkkäin (//):

```
$ egrep '/.*/' file
```

Piste on yleinen merkki tekstissä, joten tahdot usein pisteen merkitsevän pistettä - jotta sillä ei ole erikoista ohjausmerkin merkitystään. Kun tahdot etsiä merkkiä, jota työkalusi pitävät ohjausmerkkinä, laita sen eteen kenoviiva (\):

```
$ egrep '\.' file
```

Tämä komento löytää vain pisteen. Koska kenoviivan vuoksi piste pakenee ohjausmerkin tilasta, puhumme merkkien paosta kenoviivan avulla.

Jos etsit merkkijonoja, joissa on välimerkkejä etkä tahdo mitään välimerkkejä kohdeltavan ohjamerkkeinä, voi olla väsyttävää ja vaikeaa päästää jokainen merkki pakoon. Harkitse komennon fgrep käyttöä näille merkkijonoille komentojen grep tai egrep sijasta. Komento fgrep ei käsittele mitään ohjausmerkkinä. Sinun täytyy edelleenkin käyttää heittomerkkejä, jotta komentotulkki ei käsittele mitään ohjausmerkkinä.

Hakasulut antavat sinun määritellä merkkiyhdistelmiä. Etsiäksesi sekä sanoja "gray" että "grey", voit määritellä:

```
$ egrep "gr[ae]y" color_file
Black and white make grey
```

Sovittaaksesi englannin vokaaleina usein käytettyjä merkkejä, voit kirjoittaa:

```
[aeiouy]
```

Merkkien järjestys ei koskaan merkitse hakasulkujen sisällä. Voimme löytää vokaalittomia rivejä antamalla säännöllisen lausekkeen komennolle egrep. Aloitammme sekavalla tiedostolla, jonka nimi on *letter_file*:

```
This is readable text.
Ths s grbg txt.
This is more readable text.
aaaai
```

Huomaa, että toisella rivillä ei ole vokaaleja, mutta viimeisellä rivillä on pelkästään vokaaleja. Ensin etsimme vokaalia:

```
$ grep '[eauoi]' letter_file
This is readable text.
This is more readable text.
aaaai
```

Rivi ilman vokaaleja ei sovitu.

Kokeillaanpa nyt merkkejä, jotka eivät ole vokaaleja. Tämä ei merkitse pelkästään konsonantteja, mutta myös pisteitä ja välilyöntejä. Voimme kääntää merkit ylösalaisin laittamalla tarkemerkin (^) merkkien eteen. Merkkiluokassa (ja ei missään muualla) tarke merkitsee "kaikki paitsi seuraavat". Teemme sen tässä viidellä vokaalilla (sallimme merkin "y" tulla sovitetuksi, koska se voi olla myös konsonantti):

```
$ grep '[^eauoi]' letter_file
This is readable text.
Ths s grbg txt.
This is more readable text.
```

Tällä kertaa ainoastaan viimeinen rivi jätettiin pois, koska siinä oli vokaaleita eikä mitään muuta.

Merkkien luokassa voi olla myös merkkien kantamia, minkä osoitat laittamalla väliviivan kahden merkin väliin. Merkinnän [0123] sijasta voit kirjoittaa [0-3]. Ihmiset käyttävät usein seuraavia yhdistelmiä:

Mikä tahansa numero	[0-9]
Mikä tahansa pieni kirjain	[a-z]
Mikä tahansa suuri kirjain	[A-Z]
Mikä tahansa kirjain	[a-zA-Z]

Kuten taulukon viimeinen esimerkki osoittaa, voit yhdistää merkkien kantamia hakasulkujen sisällä. Voit jopa yhdistää merkkien kantamia muiden merkkien kanssa, kuten seuraavassa kirjaimien ja välimerkkien yhdistelmässä:

```
[a-zA-Z.,!?]
```

Emme joutuneet päästämään pistettä pakoon tässä esimerkissä, koska sitä ei käsitellä ohjausmerkkinä hakasulkujen sisällä.

Huomaa että merkkiluokka, olipa se miten iso tahansa, sopii aina yhteen merkkiin. Jos tahdot sen sopivan moniin merkkeihin, käytä yhtä edellisen osan määrityksistä:

```
$ egrep '\([a-zA-Z.,!?]+\)' file
```

Tämä sopii sulkuihin, joiden sisällä on mikä tahansa joukko merkkejä. Meidän täytyi päästää sulut pakoon, sillä ne ovat ohjausmerkkejä - hyvin erityisiä ohjausmerkkejä. Katsomme niitä seuraavaksi.

RYHMÄT

Sulut antavat sinun käsitellä monia merkkejä samaan aikaan. Muista että merkkiluokat hakasulkujen sisällä sopivat aina yhteen merkkiin, vaikka yksi merkki voi olla monta erilaista asiaa. Sen sijaan ryhmät voivat sopia merkkisarjaan. Jos esimerkiksi tahdot sovittaa tuhannen, miljoonan, miljardin tai biljoonan lukumääriä, voit sovittaa:

1,000

1,000,000

1,000,000,000

jne.

Voit tehdä tämän laittamalla ryhmään merkkijonon ",000" ryhmään, sulkien sen sulkuihin. Nyt mikä tahansa siihen käyttämäsi - kuten + -merkki - sovittuu koko ryhmään:

```
$ egrep '1(,000)+' file
```

Sulut tekevät vielä enemmän. Ne tallentavat sen mitä löytävät, mitä kutsutaan kaappaukseksi. Sen jälkeen voit viitata siihen myöhemmin.

Tämä on vaikeasti hahmotettava ominaisuus. Esitelläänpä sitä katsomalla tiedostoa, joka toistaa joitain merkkijaksoja:

```
This bell is a tam-tam.
This sentence doesn't appear in the egrep-generated output.
I want it quick-quick.
```

Ensimmäinen rivi sisältää sanan "tam", väliviivan ja sitten taas sanan "tam". Kolmas rivi sisältää sanan "quick", väliviivan, ja sen jälkeen taas sanan "quick". Näillä riveillä ei itse asiassa ole yhteisiä merkkijonoja, paitsi väliviiva (joka näkyy myös toisella rivillä, joten sen etsiminen ei erota ensimmäistä ja kolmasta riviä toisesta). Ensimmäisen ja kolmannen rivin yhteinen tekijä on hahmo: sana, jota seuraa väliviiva ja se itse. Niinpä voimme ottaa nuo kaksi riviä kaappaamalla sanan ja toistamalla sen (tiedoston nimi *doubles*):

```
$ egrep ' ([a-z]+)-\1' doubles
This bell is a tam-tam.
I want it quick-quick.
```

Hämmentävää? Tämä säännöllinen lauseke purettuna osiin:

(	Aloita ryhmä
[a-z]+	Mikä tahansa kirjainjoukko (yksi tai enemmän)
)	Sulje ryhmä
-	Väliviiva (yksinkertainen sovitus)
\1	Toista aiemmin kaapattu ryhmä

Merkki \1 on erikoissyntaksi, jonka tunnistavat työkalut, jotka antavat sulkeiden kaapata tekstiä. Ensimmäisellä tiedostojen rivillä se sopii merkkijonoon "tam", koska siihen merkit [a-z]+ sopivat. Kolmannella rivillä se sopii merkkijonoon "quick", koska siihen merkit [a-z]+ sopivat. Se sanoo "mitä tahansa löytyykin, tahdon sen uudelleen."

Erottaaksesi sähköpostiosoitteen toisen osan, esimerkiksi merkkijonon "fsf.org" osoitteesta "someone@fsf.org", käytä säännöllistä lauseketta, kuten:

```
([a-z._]+)@([a-z._]+)
```

Tässä tapauksessa \1 sopii osaan ennen merkkiä @, kun taas \2 sopii osaan merkin @ jälkeen. Yritä siis erottaa \2 saadaksesi merkkijonon "fsf.org".

EROTTELU

Näimme että merkkiluokat sopivat vain yhteen merkkiin samaan aikaan. Jos sinulla on kaksi tai useampia merkkijaksoja jotka voivat olla samassa paikassa, voit määritellä ne erottelun avulla. Tämä tehdään erottamalla ne pystyviivalla (|). Niinpä seuraava löytää sekä tapauksen "FSF" että tapauksen "Free Software Foundation":

```
FSF|Free Software Foundation
```

Voit laittaa erotteluun niin monta vaihtoehtoa kuin tahdot:

```
gnu.org|FSF|Free Software Foundation
```

Koska vaihtoehdot on yleensä laitettu suuremman säännöllisen lausekkeen sisään, joudut yleensä laittamaan ne sulkuihin osoittaaksesi niiden alun ja lopun:

```
The (FSF|Free Software Foundation)
```

ANKKUROINTI

Jos tahdot sovittaa jotain sen ilmaantuessa rivin alussa, mutta ei missään muualla, laita säännölliseen lausekkeeseen tarke (^). Näin voit käyttää seuraavaa ottamaan kiinni rivit, jotka alkavat pienellä merkillä:

```
^[a-z]
```

Tällä tarkkeen käytöllä ei ole mitään tekemistä sen tarkkeen kanssa, jonka näimme aikaisemmin hakasulkujen sisällä. Tarke merkitsee "rivin alussa" kun se on säännöllisen lausekkeen ensimmäinen ilmaus, mutta vain siinä sijainnissa.

Vastaavasti voit sovittaa jotain rivin lopussa lisäämällä dollarimerkin ($) säännöllisen lausekkeen loppuun:

```
[0-9]$
```

Lauseessa "I added 3 and 5 to make 8" edellinen säännöllinen lauseke tulee sopimaan numeroon 8, koska se on rivin lopussa.

Kun etsit rivejä, jotka sopivat säännölliseen lausekkeeseen tarkasti (ei erillistä testiä alussa eikä lopussa), käytä molempia ankkureita. Jos tahdot varmistaa, että rivissä on vain numeroita, kirjoita:

```
^[0-9]+$
```

Merkit [0-9]+ tarkoittavat "yksi tai enemmän numeroita" ja merkit ^ ja $ varmistavat, että säännöllinen lauseke ottaa koko rivin.

Tahdoimme näyttää sinulle mitä säännöllisillä lausekkeilla voi saada aikaan. On paljon enemmän mahdollisuuksia, joista jotkin ovat käytännöllisiä vain ohjelmointikielissä. Kuten aiemmin varoitimme, eri työkalut tukevat eri ominaisuuksia, joten joudut lukemaan dokumentaation egrepille, Sedille tai mille tahansa muulle käyttämällesi työkalulle tai kielelle saadaksesi selville mikä toimii.

Kun testaat säännöllisiä lausekkeita, voit kokeilla monia verkossa olevia tai painettuja kirjoja (joista jotkin ovat ilmaisia ja jotkin maksullisia) jotka auttavat ratkaisemaan ongelmia, kuten väärin käytettyjä hakasulkuja tai sulkuja. Sellaisten työkalujen avulla voit oppia monimutkaisemmat ominaisuudet ja kirjoittaa helpommin monimutkaisia säännöllisiä lausekkeita.

35. PERL

Perl on ohjelmointikieli, jolla voidaan tehdä tehtäviä, jotka olisivat vaikeita tai hankalia komentorivillä. Perl on oletusarvoisesti mukana useimmissa GNU/Linux -jakeluversioissa. Usein Perliä käytetään kirjoittamalla tekstieditorissa tiedosto ja antamalla se sitten ohjelman `perl` suoritettavaksi.

Perl -skriptien nimi voi olla mitä tahansa, mutta yleensä ne loppuvat päätteeseen ".pl". Voit käyttää mitä tahansa tekstieditoria luodaksesi tämän tiedoston - Emacs, Vim, Gedit, tai mikä tahansa suosikkisi onkin. Skripti voisi näyttää tältä:

```
my $a = 1 + 2;
print $a,"\n";
```

Tässä esimerkissä luomme muuttujan (käyttäen komentoa *my*) jota kutsutaan nimellä *$a* (dollarimerkki on Perlin tapa ilmaista muuttuja), joka tallentaa laskutoimituksen "1 + 2" tuloksen. Se käyttää funktiota *print* tulostamaan tuloksen, jonka pitäisi olla 3. Pilkku yhdistää kaksi tai useampia merkkijonoja. Tässä tapauksessa uusi rivi liitetään tulostetun merkkijonon loppuun. Kaikki Perl-lauseet loppuvat puolipilkkuun, vaikka ne olisivatkin eri riveillä. Jos me tallennamme tämän tiedoston nimellä *eka.pl*, voimme ajaa sen komentoriviltä.

```
$ perl eka.pl
3
```

Ohjelma tulosti "3", aivan kuin odotimme. Tietenkin voimme käyttää Perliä tekemään hyödyllisempiä asioita. Voimme esimerkiksi katsoa nykyisen hakemiston kaikkia tiedostoja.

```
my $filename;
opendir DH, "." or die "Ei voi avata hakemistoa!";
while( $filename = readdir(DH) ){
  print $filename,"\n";
}
```

```
$ perl eka.pl
perl-skripti.perl
muu-skripti.perl
dokumentti.odt
kuva.png
```

Tässä käytämme funktiota *opendir* avaamaan hakemiston lukemista varten. "DH" on hakemistokahvasi, jolla voimme viitata avoinna olevaan hakemistoon lukemista varten. Hakemistokahvaa ei julisteta kuin muuttujaa, se luodaan vain funktiota *opendir* luotaessa. Siirrämme hakemistonimen tai merkkijonon (lainausmerkkien sisällä); yksi piste viittaa nykyiseen hakemistoon. Komennot *or* tai *die* käskevät Perlin lopettaa toiminta, mikäli hakemistoa ei voida avata. Lopussa oleva merkkijono kertoo Perlille mitä sen täytyy tulostaa virheviestinä.

Silmukan *while* sisällä, funktio *readdir* ottaa hakemistokahvamme ja palauttaa seuraavan tiedostonimen hakemistossa, tallentaen sen oletusmuuttujaan "$_".

Tehdäänpä jotain näillä tiedostoilla - tässä on tapa löytää kaikki hakemiston ".pl" -tiedostot.

```
opendir DH, "." or die "Ei voi avata hakemistoa!";
while( $_ = readdir(DH) ){
  print $_,"\n" if /.pl$/;
}
```

Yllä käytämme Perlin lyhyttä ilmaisua pakataksemme tulostuksen ja evaluaation yhdelle riville, sillä sekä *print* että *if* ottavat oletusmuuttujan "$_" parametrikseen, mikäli sellaista ei ole määritelty. Parametri "/.pl$/" sanoo: sovita mikä tahansa sana, jonka loppupääte on ".pl". Alla on yksinkertaisempi mutta monisanaisempi tapa valita kaikki tiedostot, joissa on merkkijono ".txt".

```
my $filename;
opendir DH, "." or die "Ei voi avata hakemistoa!";
while( $filename = readdir(DH) ){
  if ($filename =~ /.txt$/){
    print $filename,"\n";
  }
}
```

Perl käyttää aaltosulkuja ({ }) ryhmittämään ilmaisuja haarautumiin ja silmukoihin, kuten *if* ja *while*. Operaattori "=~" kertoo komennoille *if*, kun ".txt" ilmaantuu muuttujan tai merkkijonon loppuun.

Voimme käyttää Perlissä myös komentorivin koodia käyttämällä funktiota *system*. Jos tahdomme esimerkiksi poistaa kaikki ".txt" -tiedostot, voimme käyttää seuraavaa:

```
my $dir = "./";
system("rm $dir*.txt");
```

Yllä komento *system* siirtää parametrinsa komentoriville, joka suorittaa sen täsmälleen sellaisena kuin se olisi, jos kirjoittaisimme sen. Jos nyt katsomme mitä tahansa tiedostoja, jotka loppuvat ".txt", emme löydä yhtään.

```
system("ls *.txt");
```

LISÄÄ TIETOA PERLISTÄ

Perlin verkkosivu osoitteessa http://www.perl.org sisältää vakuuttavan määrän tietoa ja dokumentaatiota Perl-kielestä.

36. RUBY

Ruby on ohjelmointikieli, jolla voidaan tehdä tehtäviä, joiden tekeminen olisi liian vaikeaa tai kömpelöä komentoriviltä käsin. Joudut asentamaan Rubyn aloittaaksesi sen käytön. Yleensä koneellasi on pakettienhallintaohjelma, jolla voit helposti asentaa Rubyn. Ubuntussa tällainen ohjelma on Synaptic. Voit myös mennä sivulle http://www.ruby-lang.org, josta voit ladata Rubyn ja löytää siitä lisätietoa.

Aivan kuin komentorivillä, voit käyttää Rubyä kirjoittamalla yksittäisiä komentoja, tai voit luoda skriptitiedoston. Jos tahdot kirjoittaa yksittäisiä komentoja, asenna "irb"-ohjelma ja käytä "irb"-komentoa komentorivillä:

```
$ irb
> 10 + 10
=> 20
> exit
$
```

Kuten näet, voit käyttää Rubyä peruslaskutoimituksiin. Tärkeä huomio Rubystä on, että se ei tulosta arvoa ainoastaan käyttäessäsi komentoa "echo" (joka on "puts" Rubyssä), Ruby tulostaa tuloksen mistä tahansa komennosta, jonka kirjoitat -- tätä "=>" tarkoittaa. Kun kirjoitat komennon "10+10", tulos on "20". Muista myös, että komennolla "exit" pääset pois irb-ohjelmasta.

Kirjoittaaksesi monen rivin skriptin Rubyssä, luot tiedoston ja tallennat sen loppupäätteellä ".rb". Voit käyttää mitä tahansa tekstieditoria luodaksesi tämän tiedoston -- Emacsia, Vimiä, Gedittiä, mikä tahansa suosikkisi onkaan. Skripti voi näyttää tältä:

```
a = 1 + 2
puts a
```

Tässä esimerkissä luomme muuttujan "a", joka tallentaa komennon "1 + 2" tuloksen. Sen jälkeen se käyttää komentoa "puts" tulostaakseen tuloksen, jonka pitäisi olla 3. Jos tallennamme tämän tiedostona "ruby.rb", voimme ajaa sen komentoriviltä:

```
$ ruby ruby.rb
3
```

Ruby-ohjelma tulosti "3", aivan kuin odotimme. Tietenkin voimme käyttää Rubya tekemään hyödyllisempiä asioita. Voimme esimerkiksi katsoa kaikkia tiedostoja hakemistossa:

```
$ irb
> Dir.entries('hakemistoni')
=> ["ruby-skripti.rb", "toinen-skripti.rb", "dokumentti.odt", "kuva.png"]
```

Käytämme "Dir.entries" -metodia katsomaan tiedostoja "hakemistoni" -hakemistossa. Huomasit luultavasti, että annamme Rubyssä parametrit eri tavalla. Sen sijaan että erottaisimme ne välilyönneillä, laitamme ne heittomerkkeihin. Joudumme myös sulkemaan kaikki sanat heittomerkkeihin - ei ainoastaan niitä, joissa on erikoismerkkejä.

Tehdäänpä jotain näillä tiedostoilla -- tässä on tapa löytää kaikki ".rb"-tiedostot hakemistossa:

```
> files = Dir.entries('hakemistoni')
=> ["ruby-skripti.rb", "toinen-skripti.rb", "dokumentti.odt", "kuva.png"]
> for file in files
```

```
> puts file if file.include?('.rb')
> end
ruby-skripti.rb
toinen-skripti.rb
```

Ensin käytämme "for"-komentoa käydäksemme läpi kaikki tiedostot. Pääsemme sitten työskentelemään jokaisen tiedoston parissa. Seuraava rivi sanoo, että tahdomme tulostaa tiedoston, jos siihen sisältyy teksti ".rb". Lopuksi lopetamme silmukan.

Voimme myös käyttää komentorivin koodia Rubyssä sulkemalla sen "`"-merkkeihin. Jos esimerkiksi tahdomme poistaa kaikki ".rb"-tiedostot, voimme käyttää seuraavaa ohjelmaa:

```
> files = Dir.entries('hakemistoni')
=> ["ruby-skripti.rb", "toinen-skripti.rb", "dokumentti.odt", "kuva.png"]
> for file in files
> `rm #{file}` if file.include?('.rb')
> end
```

Huomaa, miten suljimme "rm"-komennon "`"-merkkeihin. Käytimme myös merkkejä "#{}" sulkeaksemme Ruby-muuttujan, joten se on laitettu tähän komentoon oikein, eikä kirjaimelliseen tekstitiedostoon.

OPI LISÄÄ RUBYÄ

Jos tahdot oppia enemmän Rubystä, http://www.ruby-lang.org on Rubyn kotisivu, ja http://www.ruby-doc.org on mahtava paikka löytää käyttöoppaita ja dokumentaatiota.

37. PYTHON

Python on ohjelmointikieli, jolla voi tehdä tehtäviä, jotka olisivat vaikeita tai hankalia komentorivillä. Python on oletusarvoisesti mukana useimmissa GNU/Linux -jakeluissa. Aivan kuin komentorivillä, voit käyttää Pythonia kirjoittamalla yksittäisi komentoja, tai voit luoda skriptitiedoston. Jos tahdot kirjoittaa yksittäisi komentoja, aloita Python-tulkki kirjoittamalla komento "python".

```
$ python >>> 10 + 10
20
```

Poistuaksesi interaktiivisesta Python-istunnosta, käytä näppäinkomentoa **Ctrl + d**.

Kirjoittaaksesi monen rivin skriptin Pythonissa, jonka voit suorittaa interaktiivisen Python-konsolin ulkopuolella, joudut ensin luomaan tiedoston ja tallentamaan sen ".py"-loppupäätteellä tiedostonimen lopussa. Voit käyttää mitä tahansa tekstieditoria luodaksesi tämän tiedoston -- Emacsia, Vimiä, Gedittiä, tai mikä tahansa suosikkisi onkin. Skripti voi näyttää tältä:

```
a = 1 + 2
print a
```

Tässä esimerkissä luomme muuttujan "a", joka tallentaa laskutoimituksen "1 + 2" tuloksen. Sen jälkeen se käyttää "print"-komentoa tulostaakseen tuloksen, jonka pitäisi olla 3. Tallennamme tämän tiedoston nimellä "eka.py", ja voimme suorittaa sen komentoriviltä.

```
$ python eka.py
3
```

Python-ohjelma tulosti "3", aivan kuin odotimme. Tietysti voimme tehdä Pythonilla hyödyllisempiä asioita. Voimme esimerkiksi katsoa kaikkia nykyisen hakemiston tiedostoja.

```
$ python
>>> import os
>>> os.listdir('.')
['muistio.txt', 'lueminut.txt', 'eka.py']
```

Tässä tuomme ohjelmaan standardikirjaston "os", jossa on käyttöjärjestelmän funktioita. Kutsumme listdir-funktiota palauttaaksemme listan nykyisen hakemiston tiedostoista. Tuomme hakemiston nimen merkkijonona (suljettuna heittomerkkeihin); yksittäinen piste viittaa nykyiseen hakemistoon.

Yritetäänpä tehdä jotain näillä tiedostoilla -- tässä on tapa löytää kaikki hakemiston ".py"-tiedostot.

```
>>> files = os.listdir('.')
>>> files ['muistio.txt', 'lueminut.txt', 'eka.py']
>>> [file for file in files if '.py' in file] ['eka.py']
```

Yllä käytämme voimakasta rakennetta, jota kutsutaan listan ymmärtämiseksi, luodaksemme uuden listan muuntamalla ja suodattamalla listan. Alla on yksinkertainen mutta monisanaisempi tapa valita kaikki tiedostot, joissa on merkkijono ".txt".

```
>>> for file in files: ... if '.txt' in file: ... print file ...
```

```
muistio.txt
lueminut.txt
```

Sisennystä vaaditaan Pythonissa. Sisennys kertoo Python-tulkille, mitä "for"-silmukkaan sisällytetään ja mitä sisällytetään "if"-lauseeseen. Lisäksi sinun täytyy painaa rivinvaihtoa viimeisten kolmen pisteen lopussa kertoaksesi Python-tulkille, että olet kirjoittanut komennot.

Voimme myös käyttää komentorivin koodia Pythonissa lähettämällä sen "os.system" -funktiolle. Jos esimerkiksi tahdomme poistaa kaikki ".txt"-tiedostot, voimme tehdä näin:

```
>>> for file in files: ... if '.txt' in file: ... cmd = 'rm ' +
file ... os.system(cmd)
...
```

Yllä loimme päätekomennon "cmd" Python-merkkijonolla yhdistämällä (käyttämällä "+"-operaattoria) merkkijonot "rm" ja tiedostonimen, jonka jälkeen lähetämme sen "os.system"-funktiolle. Nyt voimme tarkastaa että tiedostot on poistettu.

```
>>> os.system('ls')
eka.py
```

LISÄÄ TIETOA PYTHONISTA

Pythonin verkkosivu on http://www.python.org ja se sisältää paljon tietoa ja dokumentaatiota Python-kielestä. Jos olet aloitteleva ohjelmoija, Jeffrey Elknerin, Allen B. Downeyn ja Chris Meyersin kirja "How to Think Like a Computer Scientist" sivulla http://openbookproject.net/thinkCSpy/index.html on hyvä paikka aloittaa.

KOMENNOT NOPEASTI

38. KOMENTOJEN PIKAOPAS

38. KOMENTOJEN PIKAOPAS

Jokaisessa tämän luvun komentoesimerkissä dollarimerkki ($) rivin alussa on minimaalinen GNU/Linux komentokehote. (Oletusarvoinen komentokehotteesi on yleensä monimutkaisempi.)

Loppurivillä on komento, jolla on valitsimia ja parametrejä. Seuraavassa ovat oppaan käytännöt.

- Rivit, jotka eivät ala dollarimerkillä "$" ovat komennon suorittamisesta tulevaa palautetta.
- Merkki "|" putkittaa komennon ulostulon toiselle komennolle.
- Merkki ">" uudelleenohjaa komennon ulostulon tiedostoon.
- Lisätäksesi ulostulon tiedostoon, käytä merkkejä ">>".
- Monilla komennoilla ei ole ulostuloa. Ne onnistuvat tai epäonnistuvat hiljaa, mutta palauttavat virhekoodin, jota skripti voi käyttää päättääkseen mitä tekee seuraavaksi.
- Merkki ";" jakaa komennot samalla rivillä. Ne tehdään järjestyksessä, vasemmalta alkaen.
- Merkki "&" komentorivin lopussa suorittaa komennon taustalla ja antaa käyttäjälle heti kehotteen seuraavan komennon antamiseksi.
- Merkki "\" rivin lopussa tarkoittaa, että komento jatkuu seuraavalla rivillä. Vaikka tässä oppaassa annetut komennot ovat aika lyhyitä, on tapauksia, joissa skriptitiedoston komento voi olla viiden tai kuuden rivin pituinen.
- Tapauksissa '*tekstiä välilyönneillä* ' ja "*lisää tekstiä välilyönneillä*" heitto- ja lainausmerkit merkitsevät, että teksti on yksi parametri, mukaan lukien ensimmäisen esimerkin lopussa olevan välilyönnin. Muuten komentotulkki tulkitsisi jokaisen sanan erilliseksi parametriksi ja hylkäisi välilyönnit.
- Tarkemerkit "`" merkitsevät komennot suoritettaviksi. Lopputulos korvaa komennon.
- Monilla komennoilla on valitsin, joka kirjoitetaan joko `-r` tai `-R`, jolla komento toimii jokaiseen alihakemiston. Tämä tapahtuu jokaisessa alkihakemistossa, joka kuuluu sille hakemistolle, jossa komento suoritetaan. Koska näitä valitsimia voidaan käyttää muille toiminnoille, sinun tulisi tarkastaa jokaisen komennon dokumentaatio, jolla tahdot käyttää tätä toimintoa.

Muutos, jonka monet ihmiset tekevät, on laittaa tämä komento käynnistystiedostoon, joka on *.bashrc* tai *.profile*.

$ **PATH=$PATH:.**

Tämä lisää nykyisen hakemiston (.) polkuun, joten voit käynnistää omat komentosi hakemistostasi siirtymällä hakemistoon ja kirjoittamalla komennon nimen. Muuten joudut kirjoittamaan `./komento` suorittaaksesi komennon nykyisessä hakemistossa. Jotkut ihmiset pitävät tätä turvallisuusriskinä. Joidenkin mielestä on parempi käyttää komentoa

$ **PATH=**

~/bin:$PATH

jotta he voivat suorittaa komentoja valitsemissaan ja omistamissaan hakemistoissa, ja nämä skriptit toimivat muiden saman nimisten sijasta. Kuitenkaan he eivät voi käyttää skriptejä satunnaisesti missä tahansa ohjelmistojärjestelmässä, jossa he sattuvat olemaan.

LS

`ls` on komento tiedostonimien listaamiseen.

Siinä voi olla valitsimia (lippuja) lisättynä komennon jälkeen miinusmerkillä "-".

Parametrejä voidaan myös lisätä.

$ ls	tavallinen tiedostolistaus
$ ls | less	putkittaa listauksen komennolle less, joka näyttää sivun kerrallaan
$ ls > tiedostolista	uudelleensuuntaa listauksen tiedostoon, jonka nimi on "tiedostolista"
$ ls -l	pitkä tiedostolista (tiedostojen koot, muutosajat, ...)
$ ls -a	listaa myös "."-alkuiset tiedostonimet, joita ei normaalisti lista
$ ls -l -a	käyttää molempia yllä mainittuja valitsimia
$ ls -la	vastaa edellistä komentoa
$ ls *.fort	listaa tiedostot, joiden nimi on "mitätahansa.fort"
$ ls -lat *.fort	t merkitsee tiedostojen listaamista aikajärjestyksessä eikä aakkosjärjestyksessä

Nämä esimerkit näyttävät kaikkien GNU/Linux -komentojen tärkeimmät piirteet. Ne ovat lyhyitä, ne ottavat valitsimia, ne ottavat parametrejä, voit käyttää useita valitsimia yhdessä, voit ketjuttaa komentoja yhteen putkilla, voit suunnata komentojen ulostulon tiedostoon.

Käytä valitsinta -R listataksesi kaikki tiedostot tietyn hakemiston alla.

```
$ ls -R
kuvat1 kuvat2 kuvat3

./kuvat1:
rautatieasema.jpg    makkaratalo.jpg

./kuvat2:
```

```
suomenlinna.jpeg     korkeasaari.jpeg

./kuvat3:
kansallisteatteri.jpeg   kaupunginteatteri.jpeg    junassa.odt
```

"R" tarkoittaa "rekursiivista". Huomaa, että rekursiivinen valitsin on -R kirjoitettuna isolla kirjaimella komennossa ls. (-r kääntää hakemiston sisällön toisin päin). Komennossa rm -R voi olla isolla tai pienellä kirjaimella.

MAN, INFO

Nyt kun tunnet päätetulkin komentojen perusteet, käytä näitä komentoja usein; voit oppia melkein mitä tahansa niiden tarjoamasta runsaasta (joskin niukkasanaisesta) dokumentaatiosta.

Voit saada yleiskuvan melkein mistä tahansa GNU/Linux -komennosta käyttämällä komentoa `man` tai uudempaa komentoa `info`. Jos `man`-komento ei toimi, se johtuu luultavasti siitä, että komentotulkkisi ei löydä käyttöopassivuja. Tämä vaatii sinua asettamaan muuttujan MANPATH oikeisiin hakemistoihin. Löytääksesi käyttöopassivuja sisältävät hakemistot, yritä seuraavaa komentoa. Se tarkoittaa: "tulosta sivut, jotka sisältävät sanan man":

```
$ find / -type d -name man -print
```

Opassivut noudattavat melko tiukkaa muotoilua. Ne alkavat komennon synopsilla, jonka jälkeen ne sisältävät kaikki komennon valitsimet ja parametrit. Mukana voi olla myös lyhyt kuvaus, esimerkkejä, yleiskatsaus, oletusarvot, lopetustila, ympäristömuuttujat ja tunnetut viat.

Info esittää vastaavaa materiaalia, mutta usein yksityiskohtaisemmin, ja se on jaettu useisiin sivuihin, joissa on navigointilinkkejä.

APROPOS

Komento `apropos` kertoo sinulle opastiedostoista, joissa on määrittelemäsi avainsana, sisältäen myös opassivut, jotka käsittelevät muita asioita kuin komentoja. Sinun voisi olla vaikea arvata joidenkin näiden sivujen nimet ilman `apropos`-komennon apua.

```
$ apropos -a samba password
smbpasswd (5)        - The Samba encrypted password file
```

PWD

GNU/Linuxissa on hakemistoja, jotka auttavat sinua järjestämään tiedostosi. Komento `pwd` kertoo sinulle "nykyisen työskentelyhakemistosi" eli CWD:n (englanniksi "Current Working Directory"). Hakemistojen määrittelyissä "." tarkoittaa nykyistä hakemistoa ja ".." sen ylähakemistoa. Merkillä "/" alkavat polut ovat absoluuttisia, eivätkä ole riippuvia nykyisestä hakemistosta. Ilman merkkiä "/" alkavat hakemistot ovat suhteessa nykyiseen hakemistoon.

Esimerkkihakemisto voisi olla */home/tomi/kivat/skriptit*. Koko nimi alkaa kauttaviivalla "/" ja tiedostojen nimet erotetaan kauttaviivoilla. Tämä eroaa Windowsista, joka käyttää kenoviivaa "/" ja Macintoshista, joka käyttää kaksoispistettä ":".

```
$ pwd
/home/minunnimeni
```

CD

Muuttaa työskentelyhakemistosi.

$ cd	siirtää sinut kotihakemistoosi
$ cd ..	vie sinut nykyisen hakemiston ylähakemistoon, esimerkiksi hakeistosta /usr/lib hakemistoon /usr
$ cd Dokumentit	siirtyy nykyisen hakemiston alla olevaan Dokumentit-hakemistoon, jos sellainen on olemassa (käyttää suhteellista polkua)
$ cd /usr/lib	vie sinut hakemistoon "/usr/lib" mistä tahansa (käyttää absoluuttista polkua)
$ cd /	juurihakemisto - tämän yläpuolella ei ole mitään.

"Kotihakemisto" on hakemisto, jossa olet kirjautuessasi sisään. Se määritellään tiedostossa */etc/passwd* (paitsi jos joku erikoinen verkkojärjestelmä on käytössä, jolloin se löytyy komennon `ypcat passwd` ulostulosta).

MKDIR

Tee hakemisto.

```
$ mkdir hakemistoni
```

RMDIR

Poista hakemito

```
$ rmdir hakemistoni
```

TOUCH

Jos tiedosto on olemassa, tämä päivittää sen muutosajan ja -päivämäärän. Jos sellaista tiedostoa ei ole, tämä komento luo sen tyhjänä.

```
$ touch uusitiedosto
```

RM

Tämä komento poistaa tiedostoja (ja hakemistoja).

Oletusarvoisesti et saa toista mahdollisuutta, sillä poistettuja tiedostoja ei voi palauttaa. Ilkeä uusille käyttäjille tehty jekku on neuvoa heitä tekemään komento "`rm -r *`" korjauksena mihin tahansa heidän ongelmaansa. Älä tee tätä, ellet tahdo asentaa uutta käyttöjärjestelmää joka tapauksessa, muista myös tehdä varmuuskopio kaikista tiedostoistasi.

```
$ rm tiedosto(t)
```

`$ rm -r tiedosto(t)`	Poistaa hakemistot ja kaiken niiden sisällön.
`$ rm -i tiedosto(t)`	Kysyy jokaisen tiedoston poistamista erikseen (suositeltu!)
`$ rm -f tiedosto(t)`	Ei valita, jos tiedostoa ei ole olemassa. Ohittaa myös -i -valinnan.

LN

Tee kovia linkkejä tai symbolisia (eli "pehmeitä") linkkejä tiedostoihin. Sekä kovat että pehmeät linkit ovat viitteitä toisiin tiedostoihin. Jos et tiedä mikä "inode" on, käytä vain pehmeitä linkkejä. (Vaikka tietäisit mikä "inode" on, käyttäisit silti luultavasti pehmeitä linkkejä suurimman osan ajasta).

Pehmeä linkki on erityislaatuinen tiedosto, joka toimii toisen tiedoston (tai hakemiston) peitenimenä - perusajatus on sama kuin Windowsin "työpöydän pikakuvakkeissa", eli muualla tietokoneellasi olevaan tiedostoon, hakemistoon tai ohjelmaan osoittavissa kuvakkeissa työpöydälläsi. Tiedosto tai hakemisto, johon linkki osoittaa, on linkin "kohde".

Käyttäessäsi komentoa `ln` muista liittää mukaan `-s` valitsin valitaksesi pehmeän linkin ja laita polku kohteeseen ennen uuden linkin kohdepolkua.

Luodaksesi pehmeän linki, jonka nimi on "linkkinimi" osoittamaan kohteeseen, jonka nimi on "tiedosto":

```
$ ln -s tiedosto linkkinimi
```

```
$ ln -s ../lib/*.so .
```

CP

Tämä komento kopioi tiedostoja.

$ cp tiedosto1 tiedosto2	Kopioi tiedosto1 tiedosto2 päälle, jos se on olemassa. Luo tiedosto2, jos sitä ei ole olemassa.
$ cp /etc /home/tomi	Kopioi hakemiston /etc hakemistoon home/tomi/etc
$ cp -r hakemistoni uusihakemistoni	Valitsimella -r kopioi myös tiedostot.

MV

Tämä siirtää ja nimeää uudelleen tiedostoja ja hakemistoja.

$ mv tiedosto2 tiedosto3	Nimeää tiedoston tai hakemiston uudelleen
$ mv /home/tomi/etc /tmp	Siirtää tiedoston tai hakemiston

WHOAMI

Tämä kertoo käyttäjätunnuksesi, esimerkiksi "tomi".

Voit siis kirjoittaa

```
$ grep `whoami` /etc/passwd
```

nähdäksesi kotihakemistosi ja muita asioita.

```
tomi:!:5037:1:P. Tomi:/u/tomi:/bin/csh
```

Komentoa `grep` käytetään etsimään merkkijonoja tekstitiedostoista.

Tarkemerkit `` aiheuttavat komennon suorittamisen sisällä. Tulos, tässä tapauksessa "tomi", korvataan ulompaan komentoon, aivan kuin käyttäjä olisi kirjoittanut `grep tomi /etc/passwd`.

Merkki ! toisessa kentässä kertoo tietokoneelle, että salattu salasana on toisessa tiedostossa. Tämän vuoksi epärehellisten ihmisten on vaikeampi löytää salasanasi.

PASSWD

Tämän pitäisi olla yksi ensimmäisistä komennoista, jotka suoritat saatuasi uuden käyttäjätunnuksen. Se asettaa salasanasi ja voi tehdä muutamia muitakin asioita. Sinulta kysytään vanha salasana (jos se on olemassa) ja sinulta kysytään uutta salasanaa kaksi kertaa. (Pituus on 8 merkkiä.)

ESIMERKKI

```
$ passwd
Changing password for "tomi"
tomi's Old password:
tomi's New password:
Enter the new password again:
$
```

On olemassa rajoituksia sille, mitä voit valita, mutta ne ovat hakemistossa */etc/security* ja niitä ei voi lukea!

Hyvä salasana on vaikeasti arvattava, mutta se on myös vaikea muistaa. On paras käyttää kaikki 8 merkkiä ja käyttää ainakin yhtä merkeistä [a-z], [A-Z] ja [0-9]. On olemassa ohjelmia, jotka arvaavat monia salasanoja, joten on epäviisasta käyttää nimiin, oikeisiin sanoihin, käyttäjätunnukseen, koneen nimeen tai muihin arvattaviin asioihin liittyviä salasanoja.

EXIT

Tämä lopettaa istuntosi tai vain yhden ikkunan (jos olet ikkunassa). `exit` löytää työt, jotka ovat pysäytettyinä (esimerkiksi painamalla **Ctrl + z**), koska ne kuolisivat, jos käyttäisit komentoa `exit`. Kirjoita `exit` uudelleen, jos olet tyytyväinen siihen. Taustatöiden suorittaminen jatkuu.

Kun käytät graafista käyttöliittymää, hiirellä käytettävää valikkoa voi käyttää lopettamaan koko istuntosi. Ikkunoiden hallintaohjemia ei ole rakennettu GNU/Linuxin sisään, ja niiden toiminta vaihtelee suuresti.

PS

Näyttää listan toimivista prosesseista. Oletusarvoisesti ps näyttää vain nykyisen komentotulkin prosessit. Nähdäksesi kaikki prosessisi käytä komentoa `ps -e`.

```
$ ps
  PID TTY          TIME CMD
29477 pts/0    00:00:00 bash
29811 pts/0    00:00:00 ps
```

KILL

Lopettaa toimivan prosessin antamatta sen päättyä. Erityisen hyödyllinen, mikäli ohjelma on joutunut loputtomaan silmukkaan, jossa se ei saa tai käsittele näppäinkomentoja.

```
$ kill 29477
```

Tappaa prosessin, jonka prosessitunnus on 29477. Edellisessä ps-komennon esimerkissä tämä olisi nykyinen komentotulkki.

CHOWN

Jokaisella tiedostolla on omistaja ja ryhmä, kuten voit nähdä komennolla `ls`. Sinulla tulisi olla ryhmä, joka on nimetty samoin kuin tunnuksesi. Vaihda käyttäjää -komento `chown` antaa sinun asettaa nämä omistajuusparametrit tiedostoille, joihin sinulla on oikeus kirjoittaa.

Voit antaa kokonaiselle ryhmälle oikeuden lukea, kirjoittaa tai suorittaa tiedostoja yhdellä komennolla. Jos tahdot pitää tiedostot yksityisenä, aseta ryhmä sisältämään vain itsesi, ja käytä `chmod` -komentoa (alla) rajoittaaksesi muiden pääsyä tiedostoon. Seuraava olettaa, että kun käyttäjätunnuksesi luotiin, sinua varten luotiin ryhmä samalla nimellä. Tämä on totta monissa nykyaikaisissa järjestelmissä, mutta ei kaikissa.

```
$ chown käyttäjä:käyttäjä tiedostoni
```

Jos olet tuonut jotain hakemistoja, jotka tulivat väärällä omistajuudella, voit muuttaa ne kaikki yhdellä komennolla. Käytä komentoa `cd` päästäksesi ylätason hakemistoon ja tee tämä komento, vaihda siihen vain oikeat käyttäjän ja ryhmän nimet. Valitsin -R kertoo komennolle `chown`, että sen täytyy käydä jokaisen alihakemiston läpi, ja jokerimerkki * kertoo sille, että jokaisen sen löytämän tiedoston ja hakemiston omistajuutta on muutettava.

```
$ chown -R käyttäjä:ryhmä *
```

CHMOD

Jokaisella tiedostolla tai hakemistolla on "tila", joka koostuu joukosta parametrejä.

Voit nähdä tämän komennolla `ls -l`.

```
$ cd ; ls -la
```

Merkki ";" erottaa kaksi komentoa samalla rivillä.

Tämä näyttää sinulle tiedostojen tilat kotihakemistossasi.

Listalla tulisi olla kaksi kohtaa, joissa lukee "." ja "..", jotka viittaavat nykyiseen hakemistoosi ja sen ylähakemistoon.

Näet listan merkkejä, joka näyttää tiedoston käyttöoikeudet, jota seuraa omistaja (luultavasti oma käyttäjätunnuksesi kaikissa kohdissa, paitsi kohdassa "..") ja ryhmät (ehkä ryhmä "staff"), tiedoston koko, viimeisen muutoksen aika ja päivämäärä ja tiedoston nimi. Tämä lista muistuttanee seuraavaa:

```
total 312
drwxr-x---  16 allen    staff       1024 Oct 21 14:07 .
drwxr-xr-x  21 sys      sys          512 Oct 13 16:25 ..
-rwxr-----   1 allen    staff        896 Oct 20 14:44 .cshrc
drwx------   2 allen    staff        512 Oct  6 08:51 .elm
-rw-r--r--   1 allen    staff         59 Oct 17 13:59 .exrc
-rwxr-----   1 allen    staff        461 Oct 17 12:18 .login
...
```

Kirjain "d" ensimmäisessä kohdassa merkitsee, että kyseessä on hakemisto. Merkki "-" merkitsee tiedostoa.

Katso mahdollisten tiedostotyyppien lista ls-komennon oppaasta komennolla man. Se on valitsimen -l alla.

Seuraavana on yhdeksän muuta merkkiä kolmessa kolmen merkin ryhmässä.

Merkit 2-4 ovat tiedoston omistajalle.

Merkit 5-7 ovat tiedoston ryhmälle.

Merkit 8-10 ovat kaikille muille, paitsi pääkäyttäjälle, jolla on aina kaikki käyttöluvat.

- "r" antaa lukuoikeuden.
- "w" antaa kirjoitusoikeuden.
- "x" antaa suoritusoikeuden. Hakemiston tapauksessa tämä tarkoittaa hakuoikeutta.
- "-" ei anna oikeuksia kyseiseen sijaintiin.

Katso `man ls` -komennolla käyttöopasta valitsimen `-l` alta nähdäksesi muut mahdolliset tilat. Huomaa, että kirjoitusoikeus hakemistossa antaa sinun poistaa tiedostoja tuosta hakemistosta, vaikka et omista tiedostoja.

```
chmod 640     tiedosto1   # asettaa tiedosto1 tilaksi -rw-r-----
chmod 755     tiedosto2   # asettaa tiedosto2 tilaksi -rwxr-xr-x
chmod go=     tiedosto3   # asettaa tiedosto3 tilaksi -???-------
(poistaa ryhmän ja muut)
chmod -R go-w $HOME       # HYVÄ TURVALLINEN KOMENTO: vain sinä voit
kirjoittaa kotihakemistoosi
```

MORE, LESS, PG, CAT

Nämä komennot antavat sinun katsoa tekstitiedostoja.

```
$ more .bashrc
$ less /etc/motd
$ cat /etc/fstab
```

Komennot `more`, `less` ja `pg` antavat sinun käydä tiedoston läpi ja etsiä merkkijonoja. Hupaisasti komennolla `less` on enemmän vaihtoehtoja kuin komennolla more.

Kun olet komennossa `more`, `less` tai `pg`, paina **h**-kirjainta nähdäksesi, mitä komentoja voit käyttää.

Komento `cat` näyttää koko tiedoston, joten se on hyvä vain tiedostoille, jotka sopivat kahteen tai useampaan ruutuun. Komennon `cat` alkuperäinen tarkoitus on liittää tiedostot yhteen, mutta se on erittäin käyttökelpoinen lyhyiden yksittäisten tiedostojen näyttämiseen.

```
$ cat tiedosto1 tiedosto2 > tiedosto3
```

GREP

Etsii ja näyttää rivejä yhdessä tai useammassa tiedostossa. Ensimmäinen versio, grep, on nimetty yleisen komentomuodon mukaan varhaisessa rivieditorissa. Komento etsii koko tiedoston läpi ja näyttää rivit, joilla kirjainjono on. Säännölliset lausekkeet, jotka on kuvailtu tämän oppaan toisessa osassa, antavat menetelmiä tekstin hahmottamiseen. Valitsimet mahdollistavat tiedostoryhmien etsimisen, sisältäen kaikki aloituspisteen alihakemistot. Yksi grepin yleisimmistä käyttötavoista on etsiä tietyn käyttäjän tai ohjelman rivejä asetus- ja lokitiedostoista.

```
$ grep `whoami` /etc/passwd
```

etsii whoami-komennon tuloksen salasanatiedostosta ja palauttaa käyttäjän tunnuksen tiedot, tosin salasana on salattuna.

DF

Vapaa tila kovalevyllä. Näyttää kovalevyjen koot, käytetyn tilan ja vapaan tilan. Näyttää koon blokkeina (koko, joka vaihtelee tiedostojärjestelmästä toiseen) oletusarvoisesti, joten käytä -h valitsinta nähdäksesi koot ihmisillä luettavassa muodossa: K on tuhat tavua, M on miljoona tavua ja G on miljardi tavua.

```
$ df -h
Filesystem            Size  Used Avail Use% Mounted on
/dev/sda1             143G   41G   96G  30% /
tmpfs                 941M     0  941M   0% /lib/init/rw
varrun                941M  380K  941M   1% /var/run
varlock               941M     0  941M   0% /var/lock
udev                  941M  2.7M  939M   1% /dev
tmpfs                 941M  844K  940M   1% /dev/shm
lrm                   941M  2.4M  939M   1% /lib/modules/2.6.27-11-
generic/volatile
```

ECHO

Toista jälkeeni, korvaavia komentoja voidaan käyttää tarvittaessa. Echo-komennot ovat melko hyödyllisiä skripteissä.

```
$ echo "Hei, maailma."
Hei, maailma.
$ echo $PATH
/home/nimeni/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbi
```

FILE

Tämä yrittää tunnistaa tiedostoja, luokitellen ne tekstiksi, suoritettaviksi tiedostoiksi tai dataksi.

Kirjoita:

```
$ file * | less
```

nähdäksesi joitain esimerkkejä.

DIFF

Erot kahden tekstitiedoston välillä. Kertoo sinulle myös onko kahdella binääritiedostolla eroa.

```
$ diff .profile~ .profile
 23a22
> scim -d &
```

Tämä sanoo, että rivi 23 on lisätty (tässä tapauksessa useamman kielen ja kirjainmerkistön käytön tuen lisäämiseksi), ja näyttää rivin uuden sisällön.

WC

Laskee sanat, sekä rivit ja merkit.

```
$ wc .login 6       7     461 .login
```

Tiedostossa *.login* on 6 riviä, 7 sanaa ja 461 merkkiä.

Ohjelma laskee minkä tahansa merkkijonon, jossa on tulostettavia merkkejä välilyöntien välissä sanaksi, joten se voi antaa outoja tuloksia ihmisten näkökulmasta.

FIND

Listataksesi kaikki nykyisen työhakemiston alla olevat tiedostot ja hakemistot, joiden nimi on core:

```
$ find . -name core -ls
```

Poistaaksesi nykyisen työhakemiston alta kaikki tiedostot, joiden nimi on core (hyödyllistä, sillä ohjelmat luovat näitä tiedostoja, kun ne lopettavat toimintansa ohjelmointivirheen vuoksi, mutta tiedostoista on sinulle vähän hyötyä, ellet osaa lukea ohjelman lähdekoodia):

```
$ find . -name core -exec rm {} \;
```

Listataksesi kaikki tiedostot ja hakemistot, joiden nimi on *"jotain*.core" nykyisen työhakemiston alla:

```
$ find . -name '*.core' -ls
```

Nimetäksesi kaikki nimellä *man* kutsutut tiedostot hakemiston */usr* alla:

```
$ find /usr -name man -type d -print
```

Seuraavat kaksi esimerkkiä ovat käyttökelpoisia sellaisten tilapäisten tiedostojen poistamiseen, joita et enää tarvitse.

```
$ find /tmp /var/tmp -mtime +3 -type f -user allen -exec rm {}
\;
$ find /var/preserve -mtime +8 -type f -user allen -exec rm {}
\;
```

Seuraavassa esimerkissä nimetään tiedostot ja hakemistot, jotka ovat joko muuttuneet edellisten kahden päivän aikana, tai joiden omistajalla on suoritusoikeudet.

```
$ find . \( -mtime -2  -o -perm -100 \) -print
```

Kahden edellisen päivän aikana tapahtuneiden muutosten tarkastaminen on hyvä tapa saada selville, miksi tietokoneesi ei toimi enää!

FTP

Lataa tiedostoja File Transfer Protocol -tiedonsiirtoprotokollan avulla. Käytä:

```
$ ftp verkkonimi
```

yhdistääksesi ftp-arkistoon osoitteessa *verkkonimi*. FTP-protokolla määrittelee kuinka kirjaudutaan sisään, navigoidaan arkiston hakemistoissa, ladataan tiedostoja tai tiedostoryhmiä ja paljon muuta. Tämä esitys antaa sinulle tarpeeksi komentoja, jotta voit navigoida arkistossa ja ladata tiedostoja.

Kun otat yhteyden arkistoon, näet komentokehotteen >, johon voit kirjoittaa komentoja.

```
> binary
```

Siirtyy binääritilaan, joka siirtää jokaisen tiedoston tavun muuttamattomana. Tämän pitäisi olla ensimmäinen komento, jonka annat, paitsi jos olet varma, että lataat vain tekstitiedostoja.

```
> ascii
```

Siirtyy tekstitilaan, joka muuttaa rivinvaihdot. Tämä on oletustila, kun FTP alkaa. Se on turvallinen ASCII-tekstille, muille kahdeksanbittisille koodauksille, sekä Unicode UTF-8 -koodaukselle. **Älä käytä tätä ohjelmille, kuville, musiikille ja muille binääritiedostoille.**

```
> dir
```

Listaa tiedostot isäntäkoneen nykyisessä hakemistossa.

```
> cd hakemisto
```

Vaihtaa hakemistoa.

```
> cdup
```

Siirtyy nykyisen hakemiston ylätason hakemistoon.

```
> get etätiedosto [paikallistiedosto]
```

Lataa tiedoston ja nimeää sen tarvittaessa uudelleen.

```
> put paikallistiedosto [etätiedosto]
```

Lataa tiedoston etäkoneelle ja nimeää sen tarvittaessa uudelleen.

```
> bye
```

Lopettaa `ftp`-istunnon.

WGET

Lataa tiedostoja netistä luotettavasti, jopa epäluotettavilla yhteyksillä. Jos yhteys katkeaa latauksen aikana, wget jatkaa yhteyden palatessa.

```
$ wget URL
```

lataa tiedoston URL-osoitteesta nykyiseen hakemistoon, käyttäen samaa nimeä. Vaihtoehtoja on paljon, kuten voit kuvitella.

TAR

Tar-komento luo yhden tar-tiedoston, joka sisältää yhden tai useamman tiedoston sisällön, tai purkaa tiedostot tar-tiedostosta. Vaikka se suunniteltiin alunperin luomaan tiedostojen yhdistelmiä varmuuskopionauhalle, se on edelleenkin standardiohjelma tiedostojen pakkaamiseen, jotta voit siirtää niitä ympäriinsä yhtenä yksikkönä. Vaihtoehtoisesti tiedostot voidaan pakata tgz-tiedostoon.

```
$ tar -cf nimi.tar hakemisto/
```

Tallentaa hakemiston *hakemisto* tiedostoon *tiedosto.tar*. Valitsin `-c` luo tar-tiedoston. Valitsin -f määrää käyttäämään annettua tiedostonimeä. Ilman valitsinta -f tulos menee standardiulostuloon. Voit käyttää tätä muotoa putkessa käyttääksesi erilaista pakkausmenetelmää, kuten bzip2, joka kuvaillaan alla.

```
$ tar -cvfz tiedosto.tgz hakemisto/
```

Varastoi hakemiston *hakemisto* pakatun sisällön tiedostoon *tiedosto.tgz*, ja antaa melko monisanaisen ulostulon konsoliin kun jokainen tiedosto käsitellään. Monisanaisempi ulostulo on saatavilla vv-valitsimella.

```
$ tar -xf tiedosto.tar
```

Purkaa tiedoston *tiedosto.tar* nykyiseen hakemistoon.

```
$ tar -xzf tiedosto.tgz
```

Purkaa gzip-tiedoston *tiedosto.tgz*. Loppupääte tgz on lyhenne aiemmin yleisestä tiedostomuodosta .tar.gz, jota käytettiin, kun tiedostojen yhdistäminen ja lopputuloksen pakkaaminen vaati tar-komennon putkittamista gzip-komennon sisääntuloon.

GZIP, GUNZIP, ZCAT, BZIP2, BUNZIP2, BZCAT

Komennot zip ja bzip2 ovat tiedostonpakkausalgoritmeja, jotka ovat suosituimpia monista tiedostojen pakkaustavoista (alunperin näitä käytettiin tiedostonsiirtoon hitailla modeemiyhteyksillä), joten nämä käänteiset algoritmit purkavat alkuperäiset tiedostot muuttumattomina. Vaikka bzip2-menetelmä saavuttaa paremman pakkauksen kuin zip, se vaatii myös enemmän prosessoriaikaa. Näytämme molempia varten komennon pakkaamiseen, purkamiseen ja useamman tiedoston yhdistämiseen. Tar-ohjelma voi käyttää gzip-pakkausta.

```
$ gzip tiedosto
```

tai

```
$ bzip2 tiedosto
```

pakkaa tiedoston arkistoon. Arkistotiedoston nimi on sama kuin alkuperäinen tiedosto, loppupäätte ".gz" tai ".bz2" lisätään.

```
$ gunzip tiedosto.gz
```

Purkaa arkiston *tiedosto.gz* alkuperäiseen tiedostoon.

Komento `gunzip` vastaa komentoa `gzip -d`, jossa valitsin `-d` tarkoittaa purkamista. Vastaavasti komento `bunzip2` vastaa komentoa `bzip2 -d`.

Komento `zcat` vastaa komentoa `gunzip -c`, jossa valitsin `-c` käskee kirjoittamaan purkamattomat tulokset yhteen tiedostoon. Tämä on usein hyödyllistä, kun hakemisto sisältää joukon pakattuja tekstitiedostoja. Esimerkiksi:

```
$ zcat *.txt.gz
```

Ulostulon yhdistäminen yhteen tiedostoon tekee siitä helpommin etsittävän. Voisit putkittaa tämän komennon ulostulon komentoon `grep`. Vastaavasti `bzcat` vastaa komentoa `bzip2 -c`.

LYNX

World-Wide Web on niin yleinen, että monet ihmiset luulevat sen olevan koko internet, mutta alunperin Web suunniteltiin sisältämään ainoastaan tekstiä, ja käytössä on edelleenkin monia tekstiselaimia. Lynx on edelleenkin suosittu niiden keskuudessa, jotka käyttävät enemmän aikaa komentorivillä tai Emacsissa kuin graafisessa käyttöliittymässä.

Tekstiselain voi vain jättää huomiotta kaiken materiaalin, joka on merkitty HTML-tageilla, jotka käsittelevät graafista materiaalia, tai se voi näyttää ALT-tekstin, joka on usein liitetty kuviin, sekä tekstiselainten vuoksi että huononäköisille ihmisille.

```
$ lynx url
```

aloittaa Lynxin. Jos määrittelet URL-osoitteen, Lynx aloittaa sillä sivulla. Muuten se menee nykyiselle oletusarvoiselle kotisivulleen, jonka voit laittaa osoittamaan minne tahansa.

Seuraava taulukko antaa sinulle tarpeeksi komentoja Lynxin käyttöön, mutta on myös monia muitakin.

?	Apua
K	Listaa komennot
+	Vieritä yksi ruutu alas
-	Vieritä yksi ruutu ylös
↓	Seuraava linkki
↑	Edellinen linkki
Return tai →	Seuraa linkkiä
←	Takaisin
a	Lisää nykyinen linkki kirjanmerkiksi
d	Lataa nykyinen linkki tiedostoon
g	Mene URL-osoitteeseen
o	Vaihda asetukset
p	Tulosta
s	Etsi
v	Katso kirjanmerkit
q	Poistu Lynxistä.

EMACS, VI, NANO, PICO

On olemassa monta erilaista tekstinkäsittelyohjelmaa eri käyttötarkoituksiin. Eräs vanhimmista on Emacs, joka on kirjoitettu LISP-ohjelmointikielellä, jolloin siihen voi lisätä komentoja LISPin avulla, sekä muuttaa näppäinkomentoja. Siinä on mahdollisuus suorittaa ulkoisia komentoja, kuten sähköpostia, joten jotkin käyttäjät tekevät kaiken Emacsin sisältä. Jotkut käyttävät mielummin vi-tekstinkäsittelyohjelmaa, jossa on samankaltainen mahdollisuus lisätä komentoja ja muuttaa näppäinkomentoja, mutta se ei korvaa komentoriviä käyttäjilleen. Monet muut, varsinkin sellaiset, jotka eivät ohjelmoi työkseen, käyttävät mielellään nanon ja picon kaltaisia yksinkertaisempia tekstinkäsittelyohjelmia.

Joka tapauksessa tekstinkäsittelyohjelman voi käynnistää ja siihen ladata tiedoston seuraavan muotoisella komennolla:

```
$ tekstinkäsittelyohjelma tiedostonimi
```

Katso eri ohjelmien dokumentaatiota nähdäksesi muita komentorivin valitsimia ja tekstinkäsittelyohjelman komentoja. Esimerkiksi:

http://www.gnu.org/software/emacs/

http://www.ccsf.edu/Pub/Fac/vi.html

http://www.nano-editor.org/

http://www.itd.umich.edu/itcsdocs/r1168/

PR

Tämä lisää otsikot ja sivunumerot tekstitiedostoihisi.

```
$ pr /etc/sendmail.cf | less
```

Paina rivinvaihtoa nähdäksesi tiedoston vierivän hitaasti. Voit käyttää myös näppäimiä **Page Up** ja **Page Down** tai nuolinäppäimiä.

LPR

tulostaa tiedoston. Hyödyllinen pelkälle tekstille, mutta voi tulostaa myös muita tiedostomuotoja (kuten PostScript ja PDF), jos järjestelmässä on tulostinajurit, jotka ymmärtävät näitä tiedostomuotoja.

```
$ lpr .profile
```

SPLIT

Oletetaanpa, että sinulla on 600MB kokoinen ISO-tiedosto, jonka tahdot jakaa useampaan osaan helpompaa varastointia varten. Voit tehdä niin komennolla:

```
$ split -b 200m kuva.iso kuva_iso_
```

Tämä esimerkki luo kolme tiedostoa, joiden nimet ovat kuva.*iso_aa*, kuva.*iso_ab* ja kuva.*iso_ac*, joista jokaisen koko on 200MB. Jos tahdot liittää ne takaisin yhteen, käytä komentoa:

```
$ cat kuva.iso_* > uusi-kuva.iso
```

Muista: mitä enemmän harjoittelet, sitä helpommin ja tehokkaammin voit työskennellä. Kokeile näillä komennoilla - ainoastaan harjoitus tekee mestarin!

LIITTEET

39. TÄSTÄ KÄYTTÖOPPAASTA
40. LISENSSI

39. TÄSTÄ KÄYTTÖOPPAASTA

Tämä käyttöopas kirjoitettiin alunperin LibrePlanet-tapahtumassa, joka oli GNU/Linux-konferenssi. Se pidettiin Harvard Science Centerissä, Cambridgessa, Massachusetsissa, 21.-22. maaliskuuta 2009. LibrePlanetia sponsoroi Free Software Foundation (FSF) ja se organisoitiin kolmeen osaan: vapaiden ohjelmistojen aktivismi, verkkopalveluiden vapaus ja korkean tärkeysasteen vapaat ohjelmistoprojektit. Tämän käyttöoppaan kehittäminen oli osa vapaiden ohjelmistojen aktivismia käsittelevää konferenssin osuutta, ja se tehtiin yhteistyössä FLOSS Manualsin ja FSF:n välillä. Kirjan kirjoistuspyrähdyksen organisoivat Andy Oram ja Adam Hyde, avustajina olivat Peter Brown, Deb Nicholson ja Danny Clark.

FREE SOFTWARE FOUNDATION

Tapahtuman osallistujamäärä oli suuri. Ensimmäistä kertaa FLOSS Manualsin kirjapyrähdykseen osallistui enemmän ihmisiä verkon kautta kuin fyysisesti. LibrePlanetin toisena päivänä, osana konferenssiin liittymätöntä ohjelmaa, sivuilla työskenteli säännöllisesti 4-5 ihmistä.

Tämä käyttöopas on kirjoitettu osana FLOSS Manuals-projektia.

Opasta kirjoitetaan wikimäisesti yhteistyönä. Sen uusin versio löytyy osoitteesta: http://fi.flossmanuals.net/komentorivin-perusteet/

Voit parannella tätä käyttöopasta. Näin voit muokata käyttöopasta:

1. REKISTERÖIDY

Rekisteröidy FLOSS Manualsin kirjoitusalustalle:

http://fi.flossmanuals.net/kirjoita/

2. KIRJOITA

Valitse käyttöopas http://fi.flossmanuals.net/kirjoita/komentorivin-perusteet/ ja luku, jota tahdot parannella.

Lisätietoa FLOSS Manualsin käytöstä löydät myös käyttöoppaastamme:

http://fi.flossmanuals.net/floss-manuals/

3. KESKUSTELU

On hyvä idea keskustella kanssamme, jotta voimme koordinoida kaikkia kirjoittajia. Meillä on kansainvälinen kanava IRCissä.

Jos osaat käyttää IRC-keskusteluohjelmaa, voit liittyä seuraavalle kanavalle:

palvelin: irc.freenode.net
kanava: #flossmanuals

4. POSTILISTA

Kaikista FLOSS Manualsiin liittyvistä asioista voit keskustella liittymällä postilistallemme:

https://list.kapsi.fi/listinfo/flossmanuals

40. LISENSSI

Kaikki kappaleet ovat kirjoittajien tekijänoikeuden alaisia. Jos muuten ei sanota, kaikki luvut tässä käyttöoppaassa on lisensoitu **GNU General Public License version 2** mukaisesti.

Tämä dokumentaatio on vapaata dokumentaatiota: voit jakaa sitä eteenpäin ja/tai muokata sitä Free Software Foundationin GNU General Public License mukaisesti; joko lisenssin version 2, tai (tahtoessasi) minkä tahansa myöhemmän version.

Dokumentaatiota jaellaan siinä toivossa, että se on käyttökelpoisa, mutta ILMAN MITÄÄN TAKUUTA; ilman edes MYYTÄVYYDEN tai TIETTYYN KÄYTTÖÖN SOPIVUUDEN oletettua takuuta. Katso lisätietoja GNU General Public Licensestä.

Tämän dokumentaation mukana olisi pitänyt tulla kopio GNU General Public Licensestä, mikäli sitä ei tullut kirjoita osoitteeseen Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.

ALKUPERÄISEN ENGLANNINKIELISEN VERSION KIRJOITTAJAT

Osa materiaalista otettu Gareth Andersonin hienosta oppaasta:

http://tldp.org/LDP/GNU-Linux-Tools-Summary/html/

Ja täältä:

http://www.gnu.org/software/bash/manual/bashref.html

ABOUT THIS MANUAL
© Free Software Foundation 2009
Modifications:
adam hyde 2009
Peter Brown 2009
Scott Walck 2009
Viktor Becher 2009

AWK
© Viktor Becher 2009
Modifications:
adam hyde 2009
Andy Oram 2009
Edward Cherlin 2009
Freaky Clown 2009
Vitor Baptista 2009

BASIC COMMANDS

Modifications:
adam hyde 2009
Andy Oram 2009
Ben Woodacre 2009
Dennis Kibbe 2009
edoardo batini 2009
Freaky Clown 2009
Marc Mengel 2009
Marcelo Magallon 2009
Max Newell 2009
Peter Davies 2009
Sameer Thahir 2009
Steffen Schaumburg 2009

BEGINNING SYNTAX

Modifications:
adam hyde 2009
Andy Oram 2009
Ben Weissmann 2009
Colin Williams 2009
Jason Woof 2009
Johannes Becher 2009
Leif Biberg Kristensen 2009
Marc Mengel 2009
Marcelo Magallon 2009
Peter Davies 2009
Viktor Becher 2009
William Merriam 2009

CHECKING EXIT

Modifications:
adam hyde 2009
Andy Oram 2009
Jason Woof 2009
Tim Goh 2009

COMMAND HISTORY

Modifications:
adam hyde 2009
Andy Oram 2009
Federico Lucifredi 2009
Freaky Clown 2009
hari raj k 2009
Steffen Schaumburg 2009
Tim Goh 2009

INTERACTIVE EDITING

Modifications:
adam hyde 2009
Andy Oram 2009
Dennis Kibbe 2009

COMMAND QUICKIE
© Free Software Foundation 2009
Modifications:
adam hyde 2009
Andy Oram 2009
Edward Cherlin 2009
Freaky Clown 2009
John Sullivan 2009
S. Lockwood-Childs 2009

CREDITS
© adam hyde 2006, 2007, 2009

CUSTOMISATION
© Steve Revilak 2009
Modifications:
adam hyde 2009
Andy Oram 2009
Edward Cherlin 2009
Freaky Clown 2009
John Sullivan 2009
Vitor Baptista 2009

CUT DOWN ON TYPING
© Free Software Foundation 2009
Modifications:
adam hyde 2009
Andy Oram 2009
Dennis Kibbe 2009
hari raj k 2009
Pierre Fortin 2009

MOVING AGAIN
© Andy Oram 2009
Modifications:
adam hyde 2009
Freaky Clown 2009

EMACS
© Free Software Foundation 2009
Modifications:
adam hyde 2009
John Sullivan 2009
Matt Lee 2009
Scott Walck 2009
Viktor Becher 2009

FILE STRUCTURE

Modifications:
Andy Oram 2009
Freaky Clown 2009
Jason Woof 2009
John Sullivan 2009
Marcelo Magallon 2009
Peter Davies 2009

GNUSCREEN

Modifications:
adam hyde 2009
Andy Oram 2009
Darren Hall 2009
Edward Cherlin 2009
Freaky Clown 2009
Ntino Krampis 2009
S. Lockwood-Childs 2009

GEDIT

Modifications:
adam hyde 2009
Edward Cherlin 2009
John Sullivan 2009

GETTING STARTED

Modifications:
adam hyde 2009
Andy Oram 2009
Barbaros Catkan 2009
Freaky Clown 2009
John Sullivan 2009
Matt Lee 2009
S. Lockwood-Childs 2009
Viktor Becher 2009

INSTALLING SOFTWARE

Modifications:
adam hyde 2009
Andy Oram 2009
Edward Cherlin 2009
Jarkko Oranen 2009
John Sullivan 2009

INTRODUCTION

Modifications:
adam hyde 2006, 2007, 2009
Andy Oram 2009
Ben Weissmann 2009
Freaky Clown 2009
Michel Barakat 2009
Peter Davies 2009
S. Lockwood-Childs 2009
Scott Walck 2009
Viktor Becher 2009
William Merriam 2009

KEDIT

Modifications:
adam hyde 2009
Edward Cherlin 2009
Matt Lee 2009
Vance Kochenderfer 2009

MAINTAINING SCRIPTS

Modifications:
adam hyde 2009
Andy Oram 2009
Edward Cherlin 2009
Freaky Clown 2009
Marcelo Magallon 2009
Michael Gauland 2009
Vitor Baptista 2009

MOVING AROUND

Modifications:
adam hyde 2009
Andy Oram 2009
Edward Cherlin 2009
Freaky Clown 2009
Peter Davies 2009
Steffen Schaumburg 2009

MULTIPLE FILES

Modifications:
adam hyde 2009
Andy Oram 2009
Darren Hall 2009
Edward Cherlin 2009
hari raj k 2009
Kent Tenney 2009
Marcelo Magallon 2009
Peter Davies 2009
Steffen Schaumburg 2009

NANO
© Free Software Foundation 2009
Modifications:
adam hyde 2009
Andy Oram 2009
Scott Wells 2009

OTHER LANGUAGES
© Andy Oram 2009
Modifications:
adam hyde 2009
John Sullivan 2009

OUTLINE
© Andy Oram 2009
Modifications:
adam hyde 2009
Edward Cherlin 2009

PARAMETER SUBSTITUTION
© Free Software Foundation 2009
Modifications:
adam hyde 2009
Andy Oram 2009
Darren Hall 2009
Freaky Clown 2009
Jason Woof 2009
Michael Gauland 2009
Viktor Becher 2009

PERL
© Free Software Foundation 2009
Modifications:
adam hyde 2009
Andy Oram 2009
Darren Hall 2009
Stephen Compall 2009

PERMISSIONS
© Andy Oram 2009
Modifications:
adam hyde 2009
Freaky Clown 2009

PIPING
© Free Software Foundation 2009
Modifications:
adam hyde 2009
Andy Oram 2009
Edward Cherlin 2009
Leif Biberg Kristensen 2009
Vance Kochenderfer 2009

PYTHON
© Free Software Foundation 2009
Modifications:
adam hyde 2009
Freaky Clown 2009
Scott Walck 2009

REGULAR EXPRESSIONS
© Andy Oram 2009
Modifications:
adam hyde 2009
Darren Hall 2009
Freaky Clown 2009

RUBY
© Free Software Foundation 2009
Modifications:
adam hyde 2009
Ben Weissmann 2009
Edward Cherlin 2009
Freaky Clown 2009
Vitor Baptista 2009
Xie Haichao 2009

SED
© Freaky Clown 2009
Modifications:
adam hyde 2009
Andy Oram 2009
Darren Hall 2009

SSH
© Free Software Foundation 2009
Modifications:
adam hyde 2009
Andy Oram 2009
Freaky Clown 2009
Scott Walck 2009

SCRIPTING
© Free Software Foundation 2009
Modifications:
adam hyde 2009
Andy Oram 2009
Beth Skwarecki 2009
edoardo batini 2009
Max Newell 2009

STANDARD FILES
© Free Software Foundation 2009
Modifications:
adam hyde 2009
Andy Oram 2009
Dev Devarajan 2009
Edward Cherlin 2009
Freaky Clown 2009
Marc Mengel 2009
Michael Gauland 2009
Michel Barakat 2009
Viktor Becher 2009
Vitor Baptista 2009

SUB COMMANDS
© Free Software Foundation 2009
Modifications:
adam hyde 2009
Andy Oram 2009
Freaky Clown 2009
John Sullivan 2009

SUPERUSERS
© Andy Oram 2009
Modifications:
adam hyde 2009
Edward Cherlin 2009
John Sullivan 2009

TEXT EDITORS
© Free Software Foundation 2009
Modifications:
adam hyde 2009
Andy Oram 2009
Edward Cherlin 2009
John Sullivan 2009
mike mcneely 2009
Renato Golin 2009

VIM
© Free SOftware Foundation 2009
Modifications:
adam hyde 2009
Andy Oram 2009
Freaky Clown 2009
John Sullivan 2009

SUOMENKIELISEN VERSION KÄÄNTÄJÄT JA KIRJOITTAJAT

OTA KOMENTO

Kirjoittajat:

2011

TomiToivio

Tomi Toivio

tomi@flossmanuals.net

,

2012

TomiToivio

Tomi Toivio

tomi@flossmanuals.net

KÄYTÖN ALOITTAMINEN

Kirjoittajat:

2012

TomiToivio

Tomi Toivio

tomi@flossmanuals.net

,

2011

TomiToivio

Tomi Toivio

tomi@flossmanuals.net

SYNTAKSI

Kirjoittajat:

2011

TomiToivio

Tomi Toivio

tomi@flossmanuals.net

,

2012

TomiToivio

Tomi Toivio

tomi@flossmanuals.net

YMPÄRIINSÄ SIIRTYMINEN

Kirjoittajat:

2012

TomiToivio

Tomi Toivio

tomi@flossmanuals.net

TIEDOSTORAKENNE

Kirjoittajat:

2012

TomiToivio

Tomi Toivio

tomi@flossmanuals.net

PERUSKOMENNOT

Kirjoittajat:

2011

TomiToivio

Tomi Toivio

tomi@flossmanuals.net

,

2012

TomiToivio

Tomi Toivio

tomi@flossmanuals.net

JÄRJESTELMÄNVALVOJA

Kirjoittajat:

2012

TomiToivio

Tomi Toivio

tomi@flossmanuals.net

USEAMMAT TIEDOSTOT

Kirjoittajat:

2011

TomiToivio

Tomi Toivio

tomi@flossmanuals.net

,

2012

TomiToivio

Tomi Toivio

tomi@flossmanuals.net

KIRJOITA VÄHEMMÄN

Kirjoittajat:

2012

TomiToivio

Tomi Toivio

tomi@flossmanuals.net

PUTKITUS

Kirjoittajat:

2011

TomiToivio

Tomi Toivio

tomi@flossmanuals.net

,

2012

TomiToivio

Tomi Toivio

tomi@flossmanuals.net

KOMENTOHISTORIA

Kirjoittajat:

2012

TomiToivio

Tomi Toivio

tomi@flossmanuals.net

STANDARDITIEDOSTOT

Kirjoittajat:

2011

TomiToivio

Tomi Toivio

tomi@flossmanuals.net

,

2012

TomiToivio

Tomi Toivio

tomi@flossmanuals.net

KÄYTTÖOIKEUDET

Kirjoittajat:

2012

TomiToivio

Tomi Toivio

tomi@flossmanuals.net

INTERAKTIIVINEN MUOKKAUS

Kirjoittajat:

2012

TomiToivio

Tomi Toivio

tomi@flossmanuals.net

POISTUMISEN TARKASTAMINEN

Kirjoittajat:

2012

TomiToivio

Tomi Toivio

tomi@flossmanuals.net

ALIKOMENNOT

Kirjoittajat:

2012

TomiToivio

Tomi Toivio

tomi@flossmanuals.net

HAKEMISTOT

Kirjoittajat:

2012

TomiToivio

Tomi Toivio

tomi@flossmanuals.net

KUSTOMISOINTI

Kirjoittajat:

2012

TomiToivio

Tomi Toivio

tomi@flossmanuals.net

PARAMETRIEN KORVAAMINEN

Kirjoittajat:

2012

TomiToivio

Tomi Toivio

tomi@flossmanuals.net

GNUSCREEN

Kirjoittajat:

2012

TomiToivio

Tomi Toivio

tomi@flossmanuals.net

SSH

Kirjoittajat:

2011

TomiToivio

Tomi Toivio

tomi@flossmanuals.net

,

2012

TomiToivio

Tomi Toivio

tomi@flossmanuals.net

OHJELMIEN ASENTAMINEN

Kirjoittajat:

2011

TomiToivio

Tomi Toivio

tomi@flossmanuals.net

,

2012

TomiToivio

Tomi Toivio

tomi@flossmanuals.net

TEKSTIEDITORIT

Kirjoittajat:

2012

TomiToivio

Tomi Toivio

tomi@flossmanuals.net

NANO

Kirjoittajat:

2012

TomiToivio

Tomi Toivio

tomi@flossmanuals.net

VI JA VIM

Kirjoittajat:

2012

TomiToivio

Tomi Toivio

tomi@flossmanuals.net

EMACS

Kirjoittajat:

2012

TomiToivio

Tomi Toivio

tomi@flossmanuals.net

KEDIT

Kirjoittajat:

2012

TomiToivio

Tomi Toivio

tomi@flossmanuals.net

GEDIT

Kirjoittajat:

2012

TomiToivio

Tomi Toivio

tomi@flossmanuals.net

SKRIPTAUS

Kirjoittajat:

2012

TomiToivio

Tomi Toivio

tomi@flossmanuals.net

SKRIPTIEN HUOLTAMINEN

Kirjoittajat:

2012

TomiToivio

Tomi Toivio

tomi@flossmanuals.net

MUUT KIELET

Kirjoittajat:

2012

TomiToivio

Tomi Toivio

tomi@flossmanuals.net

SED

Kirjoittajat:

2012

TomiToivio

Tomi Toivio

tomi@flossmanuals.net

AWK

Kirjoittajat:

2012

TomiToivio

Tomi Toivio

tomi@flossmanuals.net

SÄÄNNÖLLISET LAUSEKKEET

Kirjoittajat:

2012

TomiToivio

Tomi Toivio

tomi@flossmanuals.net

PERL

Kirjoittajat:

2012

TomiToivio

Tomi Toivio

tomi@flossmanuals.net

RUBY

Kirjoittajat:

2012

TomiToivio

Tomi Toivio

tomi@flossmanuals.net

PYTHON

Kirjoittajat:

2012

TomiToivio

Tomi Toivio

tomi@flossmanuals.net

KOMENTOJEN PIKAOPAS

Kirjoittajat:

2012

TomiToivio

Tomi Toivio

tomi@flossmanuals.net

,

2011

TomiToivio

Tomi Toivio

tomi@flossmanuals.net

TÄSTÄ KÄYTTÖOPPAASTA

Kirjoittajat:

2012

TomiToivio

Tomi Toivio

tomi@flossmanuals.net

TEKIJÄT

Kirjoittajat:

2012

TomiToivio

Tomi Toivio

tomi@flossmanuals.net

FLOSS MANUALS (SUOMI)

FI.FLOSSMANUALS.NET

Vapaat oppaat vapaille ohjelmille!

GENERAL PUBLIC LICENSE

Version 2, June 1991

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Lesser General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

> **a)** You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.
> **b)** You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.
> **c)** If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

> **a)** Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms

of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS