Sekalaisia selvennyksiä

Panu Kalliokoski ja Jussi Syrjänen

Sisältö

Tässä dokumentissa annetaan esimerkkejä ja tarkennuksia sekalaisista aiheista.

1 Komento vai tiedosto

Unix-komento muodostuu sanoista. Esimerkiksi komento

```
cat tied1 tied2
```

koostuu kolmesta sanasta: “cat”, “tied1” ja “tied2”.

Milloin sana tulkitaan tarkoittamaan jonkin komennon nimeä, milloin jotain tiedostoa?

Nyrkkisääntö on se, että ensimmäinen sana on komennon nimi ja loput ovat tiedostoja. (Itse asiassa komennon nimi “päättää”, miten muut sanat tulkitaan — esimerkiksi `cat` tulkitsee ne tiedostoiksi, joiden sisältö pitää näyttää.)

Kun mukaan tulevat uudelleenohjaukset ja putket, asia monimutkaistuu hieman. Tiedostoon ohjauksen (> tai ”>>”) jälkeen tuleva sana tulkitaan *aina* tiedostoksi, johon komennon tulostama teksti pannaan. Putken (|) jälkeen tulevat sanat taas tulkitaan uudeksi komennoksi, jolle edellisen komennon tulostama teksti annetaan syötteenä.

Seuraavissa esimerkkikomennoissa tiedostoiksi tulkitut sanat ovat *kursiivilla* ja komentojen nimiksi tulkitut sanat **lihavoidulla**. (Kokeile myös, mitä nämä komennot tekevät!)

- **date** -I > *tämä_päivä*
- **date** -I | **cut** -d- -f3

- **date** -I | **cut** -d- -f3 > *päivänro*
- **date** -I | **cut** -d- -f3 | **rev**
- **cat** *cat*
- **tips** tops tups

Lisää esimerkkejä.

```
cat -n lord_of_the_rings.txt | grep -i 'tengwar' | grep 'Rumil' | less
```

tulkitaan seuraavasti:

`cat -n lord_of_the_rings.txt` → **cat**-ohjelma tulostaa tiedoston `lord_of_the_rings.txt` (Mitä **-n**-valitsin mahtaa tehdä? Käy katsomassa **cat**:in ohjeista!)

| → edellisen komennon tulostama teksti annetaan seuraavalle komennolle syötteenä

`grep -i "tengwar"` → **grep** etsii ja tulostaa syötteestään ne rivit, joilla on sana "tengwar" (Mitäköhän **-i**-valitsin oikein tekee?)

| → edellisen komennon tulostama teksti annetaan seuraavalle komennolle syötteenä

`grep "Rumil"` → **grep** etsii ja tulostaa syötteestään ne rivit, joilla on sana "Rumil" (Kaikilla näillä riveillä on myös sana "tengwar"... Miksi?)

| → edellisen komennon tulostama teksti annetaan seuraavalle komennolle syötteenä

`less` → **less**-ohjelma näyttää syötteensä ruudullinen kerrallaan. Käyttäjä voi myös selata syötettä ja tehdä kaikkea muutakin kivaa.

Sen, mitä sanat "Rumil" ja "tengwar" tarkoittavat ja miten ne liittyvät toisiinsa, saatte selville tutkimalla *Sormusten Herraa* — tai, jos teillä on se tekstitiedostona, suorittamalla esimerkkikomennon oikeasti. :)

2 Unix-komennon kommunikaatiomalli

Maailma on Unix-ohjelman kannalta omituinen. Komento kommunikoi muun maailman kanssa perin yksinkertaisen mallin kautta. Se saa "syötettä", eli matskua / tietoa / ohjeita käsiteltäväkseen, kolmea kautta; ja se tuottaa "tulostetta", eli ulkomaailmalle tuotettua tietoa tms., myös kolmea kautta. Lisäksi se voi käyttää käyttöjärjestelmän palveluita, kuten lukea tiedostoja.

syötteet:		tulosteet:
valitsimet ja argumentit →		→ vakiotuloste
vakiosyöte →	Komento	→ vakiovirhe
ympäristömuuttujat →		→ palautusarvo
	käyttöjärjestelmän palvelut:	

- tiedostot
- verkkoyhteydet
- jne.

Komento voi käyttää näitä kolmea syöte- ja tulostetapaa haluamallaan tavalla. Yleensä kuitenkin niillä on seuraavat merkitykset:

- valitsimet ja argumentit — kertovat komennolle, mitä ja miten tehdään.
- vakiosyöte — tarjoaa komennolle materiaalia käsiteltäväksi (ainakin siinä tapauksessa, ettei sitä tarjota muualta: esim. ei kerrota mitään tiedoston nimeä, minkä sisältöä on tarkoitus käsitellä.)
- ympäristömuuttujat — kertovat komennolle, miten sen tulisi toimia. Erona valitsimiin on se, ettei käyttäjän tarvitse erikseen asettaa ympäristömuuttujaa joka kerta ohjelmaa käynnistäessään. Monet ohjelmat tarjoavat jonkin asetuksen muuttamiseen sekä valitsimia että ympäristömuuttujan, josta ne lukevat oletusarvon.
- vakiotuloste — ohjelman tuottama, "tavallinen" lopputulos.
- vakiovirhe — virheilmoitukset yms. suoraan käyttäjälle tarkoitetut lausahdukset. Erillään, jotteivät esim. varoitukset menisi putkilinjassa seuraavalle komennolle prosessoitaviksi vaan tulisivat suoraan käyttäjän luettavaksi.
- palautusarvo — karkea raportti siitä, "onnistuiko" ohjelma tehtävässään vai tuliko esim. jokin virhe. Rivikäyttäjälle tällä ei ole paljon merkitystä (virheistä kerrotaan joka tapauksessa virheilmoituksilla), mutta esimerkiksi tehtäessä ohjelmia, jotka kutsuvat toisia ohjelmia, on tärkeää tietää yleisluontoisesti, mitä kutsutulle ohjelmalle kävi.

Monet ohjelmat jättävät käyttämättä jotain tai useita näistä. Esimerkiksi on paljon ohjelmia, jotka eivät piittaa yhdestäkään ympäristömuuttujasta (toim.huom.: ei pidä paikkaansa nykyaikaisessa Unixissa, sillä kieliasetukset (locale) vaikuttavat jotakuinkin joka ohjelmaan). `echo` ei tee vakiosyötteellään mitään, eikä `rm`. Jotkin erikoislaatuiset, vain yhteen asiaan tarkoitetut ohjelmat, kuten `whoami`, eivät tee valitsimilla ja argumenteilla (melkein) mitään. Useat Unix-ohjelmat eivät normaalitilanteessa sano yhtään mitään.

3 Merkkien lainaaminen

Komentotulkki, shell, on monipuolinen ohjelma, ja siinä on paljon merkkejä, joilla on jonkinlainen erikoismerkitys. Näitä ovat ainakin välilyönti, sen seuralainen sarkainmerkki ja rivinvaihtomerkki, sekä kaikki nämä merkit:

```
$ #   ] [ ( ) | &amp; ; < > ! ? *  " ' w_bq ~
```

Jos minkään näistä merkeistä haluaa sisällyttää komentoon sellaisenaan, ilman erityismerkitystä, se pitää "lainata". Komentotulkissa on kolme mekanismia merkkien lainaamiseen:

- Kenoviiva (\), joka lainaa vain yhden merkin, seuraavan merkin. Esimerkiksi `echo \*` näyttää tähden. (Kokeile myös ilman kenoviivaa!)
- Lainausmerkit ("), joiden välissä olevassa tekstissä erityismerkityksensä menettävät kaikki erikoismerkit paitsi $, ‘ (tämä ei ole heittomerkki vaan ns. kääntöhipsu *backtick*), \ ja "(joka siis lopettaa lainauksen). Esimerkiksi `grep -i "'[a-zåääö]*'"` etsii rivejä, joilla on heittomerkeissä oleva sana.

- Heittomerkit (’), joiden välissä olevassa tekstissä erityismerkityksensä menettävät kaikki erikoismerkit paitsi ’ (joka siis lopettaa lainauksen). Esimerkiksi `sed 's/\([A-ZÅÄÖ]\)[a-zääö]* \([A-ZÅÄÖ][a-zääö]*\)/\1. \2/g'` muuttaa tekstistä nimet sellaisiksi, että etunimestä sanotaan vain alkukirjain.

Lainausmerkkejä ja heittomerkkejä kutsutaan yhdessä hipsuiksi. Tavallisen käyttäjän kannalta on melko samantekevää, kumpaa niistä käytetään. Jos lainattu teksti sisältää heittomerkkejä, lainausmerkit ovat kätevämmät, ja vastaavasti toisin päin. Jos tarvitsee lainata vain yksi yksittäinen merkki, kenoviiva on kätevin. Jos lainattu teksti sisältää paljon kenoviivoja, heittomerkki on ainoa kätevä tapa.

4 Mikä on muuttuja?

Muuttuja on tiedon paikka: lokero, jolla on nimi, ja johon on talletettu jokin arvo. Unixissa on ympäristömuuttujia. Muuttujan käsite on hankala, koska se on niin yleispätevä: muuttuja sinänsä ei sisällä mitään tietoa siitä, mihin sitä käytetään.

Ajatellaanpa vaikka, että meillä on seuraavat muuttujat:

muuttuja	arvo
LAMPAAT	8
POSSUT	2
LEHMÄT	1

On helppoa arvata, että tässä muuttujia on käytetty kotieläinten lukumäärän muistissa pitämiseen. Jokaiselle kotieläintyypille on oma muuttujansa, jonka arvo on kyseisen tyyppisten eläinten lukumäärä.

Toisaalta meillä voi olla myös seuraavat muuttujat:

muuttuja	arvo
Hikka	paha
Torst_ruoka	kukkakaalikeitto
HILSETTÄ	35.8
foo	bar

Näistä muuttujista ei ole välttämättä helppoa arvata, mitä tarkoitusta ne ajavat.

Ympäristömuuttujat ovat ikään kuin tietovarasto, joka on olemassa käynnistettäviä ohjelmia varten. Jokainen ohjelma päättää ihan itse, mitä muuttujia (jos mitään) se tarkastelee ja mihin/miten niiden arvot vaikuttavat. Esimerkiksi `man`-komento katsoo MANPATH-muuttujan perusteella, mistä hakemistoista man-sivuja tulee etsiä. Jos asentelee itselleen ohjelmia, niiden man-sivut sisältävä hakemisto kannattaa lisätä MANPATH-muuttujan arvoon.

Mutta koska ympäristömuuttujia voi asettaa niin kuin haluaa, yhtä hyvin niitä voi käyttää esim. muistikirjana. (Koska ohjelman ympäristömuuttujien arvot häviävät, kun ohjelma (esim. komentotulkki) loppuu, tiedosto on kuitenkin ehkä parempi muistikirja.)

5 Absoluuttiset ja relatiiviset polut

Kaikille komennoille, joille ylipäänsä voi antaa argumentiksi tiedostoja (esim. `less`-komennolle tiedosto, jota halutaan lueskella), voi määrittää tiedoston kahdella tavalla: absoluuttisesti tai relatiivisesti.

Tiedoston *absoluuttinen* nimipolku on olemassa sitä varten, että jokaiseen tiedostoon olisi yksiselitteinen ja pysyvä tapa viitata. Niin kauan kuin tiedostoa ei siirretä paikasta toiseen, absoluuttinen nimipolku pysyy aina samana. Käyttöjärjestelmä tunnistaa absoluuttisen nimipolun siitä, että se alkaa kauttaviivalla (/); esimerkiksi nimipolku /usr/include/netinet/in.h osoittaa tietyssä koneessa aina samaan, tiettyyn tiedostoon.

Relatiivinen nimipolku on suhteessa *työhakemistoon*. Tämä mekanismi on olemassa sitä varten, ettei usein pitkähköjä ja hankalahkoja absoluuttisia nimipolkuja tarvitsisi aina käyttää. Koska relatiiviset nimipolut ovat suhteessa työhakemistoon, työhakemiston vaihtuminen muuttaa jokaisen tiedoston relatiivista nimipolkua. Tästä aiheutuu joskus sekaannusta, jos käyttäjä ei muista, missä vaiheessa on vaihtanut työhakemistoa tms.

Esimerkki: kotihakemistossani (joka olkoon /home/atehwa) on alihakemisto proj, jossa on alihakemistot stx ja piki. Lisäksi kotihakemistossani on alihakemisto tmp. Alkuun työhakemistoni on kotihakemistoni. Tällöin:

absoluuttinen nimi	relatiivinen nimi
/home	..
/home/atehwa	.
/home/atehwa/proj	proj
/home/atehwa/proj/stx	proj/stx
/home/atehwa/proj/piki	proj/piki
/home/atehwa/tmp	tmp

Jos vaihdan työhakemistokseni /home/atehwa/proj komennolla "'cd proj"', tilanne muuttuu tällaiseksi:

absoluuttinen nimi	relatiivinen nimi
/home	../..
/home/atehwa	..
/home/atehwa/proj	.
/home/atehwa/proj/stx	stx
/home/atehwa/proj/piki	piki
/home/atehwa/tmp	../tmp

Jos vaihdan työhakemistokseni /home/atehwa/proj/stx komennolla `cd stx`, tilanne muuttuu taas:

absoluuttinen nimi	relatiivinen nimi
/home	../../..
/home/atehwa	../..
/home/atehwa/proj	..
/home/atehwa/proj/stx	.
/home/atehwa/proj/piki	../piki
/home/atehwa/tmp	../../tmp

Vielä yksi esimerkki: jos vaihdan työhakemistokseni /home/atehwa/tmp komennolla `cd ../../tmp` tai `cd /home/atehwa/tmp`, tilanne muuttuu tällaiseksi:

absoluuttinen nimi	relatiivinen nimi
/home	../..
/home/atehwa	..
/home/atehwa/proj	../proj
/home/atehwa/proj/stx	../proj/stx
/home/atehwa/proj/piki	../proj/piki
/home/atehwa/tmp	.

6 Mikä on www-sivun olemus?

www-sivu on tiedosto. Sen sisältö on yleensä HTML-kuvauskieltä, joka on raakatekstiä pienin lisäyksin: siellä on merkintöjä siitä, mikä on otsikko, mikä on lihavoitu, jne. (Tämän www-sivun "todellisen" sisällön saa useimmissa selaimissa näkyviin jostain valikosta löytyvällä "view source" -toiminnolla.)

Kun www-selaimella aletaan katsella www-sivua, selain tekee kaksi asiaa:

1. se hakee kyseistä www-sivua vastaavan tiedoston
2. se esittää tämän tiedoston haluamallaan, mielellään mahdollisimman selkeällä tavalla.

Näin ollen selaimessa näkyvä kuva *ei ole sivu itse*, vaan se on selaimen sivusta laatima *esitystapa*. Eri selaimet saattavat antaa samalle sivulle huomattavan erilaisia esitystapoja, esimerkiksi tekstipohjainen selain kuten `lynx` tai `w3m` näyttää sen varsin eri tavalla kuin esimerkiksi Internet Explorer, kännyköiden WAP-selaimet näyttävät sen vielä eri tavalla, ja jotkin "selaimet" ovat oikeasti puhesyntetisaattoreita, jotka pajattavat sivun sisällön ääneen. HTML-kuvauskieli yrittää tehdä mahdolliseksi esittää sivun mielekkäästi kaikissa näissä erilaisissa medioissa.

Ensimmäinen kohtakaan (sivun hakeminen) ei ole yksiselitteinen. www-selaimella on useita tapoja saada kyseinen tiedosto. Yleisin tapa on se, että selain ottaa Internetin kautta johonkin palvelimeen yhteyttä (HTTP-protokollalla), pyytää tätä lähettämään kyseisen sivun, ja näyttää sitten vastaanottamansa sisällön. Mutta selain osaa näyttää myös paikallisessa tiedostojärjestelmässä olevia sivuja. Selaimen ylälaidassa yleensä näkyvä "sivun osoite", URI, kertoo yksiselitteisesti, miten ja mistä sivu haetaan / on haettu.

7 Xargs-komennon käyttöesimerkkejä

`xargs` on näppärä komento. Se liittyy olennaisesti Unix-komennon kommunikaatiomalliin: se muuntaa yhdenlaista syötettä, eli vakiosyötteestä tulevaa tekstiä, toisenlaiseksi, eli komentoriviargumenteiksi. Sitä tarvitaan, jos jokin komento X *tulostaa* jotain, mikä pitää antaa toiselle komennolle Y *argumenteiksi* (eikä syötteeksi, jolloin riittää yksinkertainen putki).

Annetaan esimerkkinä komento, jolla haluamme lopettaa kaikki omat emacs-prosessimme. Näiden prosessien pid-numerot saa irti `ps`-komennon tulostetta muokkaamalla:

```
$ ps
PID TTY          TIME CMD
```

```
22119 tty2     00:00:00 bash
22154 tty2     00:00:00 emacs
22160 tty2     00:00:00 ps
$ ps | grep emacs | cut -d' ' -f1
22154
```

Hyvä, nyt meillä on komento, jolla saadaan pid:t selville (useista järjestelmistä löytyvä `pidof` tekee muuten saman homman). Mutta miten tämä tieto välitetään `kill`-komennolle? Sehän ei lue lopetettavien prosessien numeroita syötteestä, vaan ne pitää antaa komentolinja-argumentteina:

```
$ ps | grep emacs | cut -d' ' -f1 | kill
kill: usage: kill [-s sigspec | -n signum | -sigspec] [pid | job]...
```

Ratkaisu on tietenkin `xargs`, joka ottaa argumentikseen komennon ja antaa sille syötteensä argumenteiksi. Oikea komento on

```
$ ps | grep emacs | cut -d' ' -f1 | xargs kill
```

Toinen esimerkki: oletetaan, että meidän pitää poistaa kaikki tekstitiedostomme, joissa esiintyy (poliittisesti epäkorrekti?) sana "homppeli". Saadaksemme tämän aikaan meidän täytyy (1) luoda lista kaikkien tekstitiedostojemme nimistä:

```
$ find . -name '*.txt'
./examples/Stx-doc.txt
./examples/Stx-ref.txt
./examples/artikkeli.txt
./examples/stx2any.txt
./juttu.txt
```

(2) suodattaa näistä `grep`:llä ne, jotka sisältävät kyseisen sanan (valitsin -l on tarkoitettu tähän: sillä `grep` näyttää vain niiden tiedostojen nimet, joista löytyy kyseinen teksti, ei kohtia, josta se löytyy):

```
$ find . -name '*.txt' | xargs grep -l homppeli
./juttu.txt
```

(3) ohjata tulos `rm`:lle. `rm`:lle annetaan poistettavat tiedostot argumentteina, joten taas tarvitaan `xargs`:a:

```
$ find . -name '*.txt' | xargs grep -l homppeli | xargs rm
```

Ilman ensimmäistä `xargs`:a `grep` etsisi homppeli-sanaa suoraan `find`:n tulosteesta, siis tiedostojen *nimistä*. `xargs` ohjaa ne argumenteiksi, jolloin `grep` käsittää ne tiedostoiksi, joiden *sisällöstä* pitää etsiä. Ilman toista `xargs`:a `rm` ei saisi yhtäkään tiedostonimeä argumentikseen ja valittaisi. (`rm` ei lue syötettään.)