10 Awk

Awk on ohjelmointikieli, joka on erityisen tehokas käsiteltäessä järjesteltyä tietoa ja tulostettaessa raportteja. *Awk* voi lukea syötteen joko tiedostosta tai *stdin*:sta. Kuten *sed*kin *awk* lukee syötettä rivi kerrallaan. Syötteestä *awk* etsii rivit, jotka täsmäävät annettuun merkkijonolausekkeeseen ja suorittaa riveille halutut toimenpiteet.

10.1 Perusteita

*Awk*in käynnistys komentoriviltä tapahtuu seuraavasti:

awk [-v var=value] [-Ffs] 'program' file ...

tai

awk [-v var=value] [-Ffs] -f progfile file ...

Awk-ohjelman suorituksen voidaan ajatella koostuvan yhdestä "syötesilmukasta". Silmukassa luetaan rivi tiedostosta ja mahdollistetaan luetun rivin prosessointi. Tämä kaikki on *awk*ssa sisäänrakennettuna, joten käyttäjän ei tarvitse huolehtia prosessoitavan tiedoston avaamisesta ja rivienlukemisesta. Normaalisti edellämainittua syötesilmukkaa suoritetaan niin kauan kun syötetiedostossa on rivejä.

Itse *awk*-ohjelma (program) koostuu yhdestä tai useammasta *pattern*- ja *action*-parista:

```
pattern {action}
pattern {action}
...
```

Pattern-osa voi olla esim. merkkijonolauseke tai ehtolauseke, johon tietueen, yleensä rivin, sisältöä verrataan. Jos ehto tai lauseke täsmää, suoritetaan *action*-osan sisältämät komennot.

Esim.

```
awk '/^$/ {print "Empty line"}' file
```

Jos *pattern*-osa jätetään pois suoritetaan *action*-osan komennot kaikille tietueille.

Awk mahdollistaa myös tiettyjen rutiinien suorittamisen ennenkuin yhtään syöteriviä on luettu. Samoin *awk* mahdollistaa tiettyjen rutiinien suorittamisen, kun kaikki syöterivit on luettu. Kyseiset rutiinit määritellään *awk*-ohjelmassa **BEGIN** ja **END** avainsanoilla. Tällöin ohjelman rakenne voi olla seuraavanlainen.

```
BEGIN {action}
pattern {action}
pattern {action}
...
END {action}
```

Awk olettaa, että syöte on jollain tavalla järjestettyä eikä vain päättymätön merkkijono. Perusoletuksena *awk*ssa on että, syöte jakautuu tietueisiin (record) ja tietue jakautuu kenttiin (field). Tietueet erotellaan normaalisti rivinvaihdolla (rivit ovat siis tietueita) ja kentät joko välilyönneillä tai tabulaattoreilla. Kenttäerotin voidaan vaihtaa komentorivioptiolla *-Ffs*, missä *fs* tarkoittaa uutta kenttäerotinta.

*Awk*ssa kenttiin voidaan viitata kenttäoperaattorilla $. Operaattoria seuraa numero tai muuttuja joka kertoo kentän paikan. $1 tarkoittaa ensimmaistä kenttää, $2 tarkoittaa toista kenttää jne.

Esim.

```
awk '{print $1, $3, $2}' tied
```

Tulostaa tiedoston `tied` riveiltä kentät yksi, kolme ja kaksi. $0 tarkoittaa koko syöteriviä.

10.2 Muuttujat

Muuttujat *awk*ssa voivat olla joko numero- tai merkkijonoarvoisia. Muuttujan tyyppiä ei määritellä erikseen vaan se määräytyy muuttujan arvon / sisällön perusteella. Muuttujia ei tarvitse alustaa erikseen, sillä kaikki *awk*ssa esiintyvät omat muuttujat ovat oletusarvoisesti alustettu tyhjiksi numeroarvolla nolla.

Muuttujien nimet saavat sisältää sarjan kirjaimia, numeroita ja alaviivoja (_). Muuttujan nimi ei kuitenkaan saa alkaa numerolla. Isoilla ja pienillä kirjaimilla on oma merkityksensä eli `a` ja `A` ovat eri muuttujia.

*awk*ssa on lisäksi joukko systeemimuuttujia, joilla on omat merkityksensä ja jotka ovat ns. "varattuja nimiä".

10.2.1 Systeemimuuttujat

Awk tuntee joukon systeemimuuttujia (sisäisiä muuttujia). Nämä muuttujat voidaan jakaa kahteen tyyppiin. Ensimmäisen tyypin muuttujat ovat sellaisia joiden oletusarvoja on mahdollista muuttaa, tällaisia ovat esimerkiksi kenttä- ja tietue-erottimet. Toisen tyypin muuttujia käytetään lähinnä raporttien tulostuksessa ja prosessoinnissa. Näiden muuttujien arvot *awk* ylläpitää automaattisesti. Tällaisia muuttujia ovat esimerkiksi kenttien määrä nykyisellä rivillä sekä nykyisen rivin numero.

Tärkeimmät systeemimuuttujat:

FS määrittelee kenttäerottimen (field separator). Oletuksena kenttäerotin on välilyönti tai tabulaattori.

NF määrittelee kenttien määrän nykyisessä syötetietueessa. Muuttujan NF arvon muuttaminen saattaa aiheuttaa yllättäviä sivuvaikutuksia.

NR sisältää nykyisen tietueen numeron.

FNR Kuten muuttuja NR, mutta mikäli on määritelty useita syötetiedostoja niin muuttuja FNR sisältää tietuenumeron senhetkisessä syötetiedostossa.

RS määrittelee tietue-erottimen (record separator). Muuttujan RS oletusarvo on rivinvaihto (newline). Joissakin *Awk*-versioissa huomioidaan vain ensimmäinen merkki, joissakin RS voi olla myös merkkijonolauseke.

OFS määrittelee tulostuksessa käytettavan kenttäerottimen. Oletusarvo on välilyönti.

ORS määrittelee tulostuksessa käytettävän tietue-erottimen. Oletusarvo on rivinvaihto.

OFMT määrittelee numeroiden tulostuksessa käytettävän formaatin. Oletusarvo on %.6g

FILENAME sisältää nykyisen syötetiedoston nimen.

ARGC sisältää komentoriviargumenttien lukumäärän.

ARGV Taulukko joka sisältää komentoriviargumentit.

ENVIRON Assosiatiivinen taulukko ympäristömuuttujista.

CONVFMT POSIX:n mukana tullut muuttuja joka kontrolloi numeroiden muuttamista merkkijonoiksi. Oletusarvo on %.6g. Kokonaisluvut pysyvät muunnoksessa aina kokonaislukuina riippumatta CONVFMT tai OFMT muuttujista.

RLENGTH *match()* funktion asettama muuttuja joka sisältää täsmänneen jonon pituuden.

RSTART *match()* funktion asettama muuttuja joka sisältää täsmänneen jonon alkupaikan.

SUBSEP Katso luku Moniulotteiset taulukot (10.4.1)

10.2.2 Operaattorit

Aritmeettiset operaattorit

$+$ Lisäys

$-$ Vähennys

$*$ Kertominen

/ Jakaminen

`%` Modulo

ˆ Potenssiin korotus

Sijoitusoperaattorit

$++$ Kasvatetaan muuttujan arvoa yhdellä.

$--$ Vähennetään muuttujan arvoa yhdellä.

$+=$ Lisätään muuttujaan, esim. $x += 2$ tarkoittaa $x = x + 2$

$-=$ Vähennetään muuttujaa, esim. $x -= 2$ tarkoittaa $x = x - 2$

$*=$ Asetetaan muuttujaan kertolaskun tulos, esim $x *= 2$ tarkoittaa $x = x * 2$

$/=$ Asetetaan muuttujaan jakolaskun tulos, esim $x /= 2$ tarkoittaa $x = x/2$

$\%=$ Asetetaan muuttujaan modulo, esim $x \% = 2$ tarkoittaa $x = x \% 2$

$\hat{}=$ Asetetaan muuttujaan potenssiinkorotuksen tulos, $x\hat{}= 2$ tarkoittaa $x = x\hat{}2$

Boolen operaattorit

`||` Looginen TAI (OR)

`&&` Looginen JA (AND)

`!` Looginen EI (NOT)

Vertailuoperaattorit

< Vähemmän kuin.

> Enemmän kuin.

<= Vähemmän tai yhtäpaljon kuin.

>= Enemmän tai yhtäpaljon kuin.

== Arvot samat.

!= Erisuuri kuin

~ "Täsmää" merkkijonolausekkeeseen, esim $5~/[0-9]/ on tosi, jos nykyisellä rivillä viides kenttä sisältää numeron.

!~ Ei "täsmää" merkkijonolausekkeeseen, esim $5 !~/[0-9]/ on tosi, jos nykyisellä rivillä viides kenttä ei sisällä numeroa.

10.2.3 Formatoitu tulostus *printf()*

Tulostus *awk*ssa tapahtuu `print`-komennolla, mutta `print` komento ei tarjoa mitään valmista keinoa tulostuksen formatointiin. Formatoituun tulostukseen käytetään C-kielestä lainattua *printf* funktiota, jonka syntaksi on

printf (formaatti **[,** argumentit **])**

Esim. `'BEGIN { printf ("Terve maailma!\n") }'`

On muistettava että *printf* ei automaattisesti tulosta rivinvaihtoa kuten *print. Printf* funktiossa on rivinvaihto tulostettava erikseen käyttämällä "*n*" merkintää.

10.3 Ehtorakenteet ja silmukat

Awk-kieli tarjoaa myös mahdollisuuden ehtorakenteiden ja silmukoiden käyttöön. *Awk*ssa näiden rakenteiden syntaksi on lainattu C-kielestä.

10.3.1 Ehtorakenteet

*Awk*ssa on kaksi ehtorakennetta, *if* ja C-kielestä tuttu ehto-operaattori. *If*-lauseen syntaksi on seuraavanlainen:

if (ehto)

komento1
[else

komento2 **]**

Jos ehto toteutuu (eli on eri kuin nolla tai ei-tyhjä), niin suoritetaan komento1. Mikäli *else* osa on määritelty komento2 suoritetaan jos ehto on epätosi (nolla tai tyhjä). Ehto voi sisältää aritmeettisia-, boolean- tai vertailuoperaattoreita. Mikäli komeentoja on useampia on ne sijoitettava kaarisulkuihin { }.

Esim. `if ( x ) print x`

Mikäli x:llä on nollasta poikkeava arvo se tulostetaan.

Toinen ehtorakenne on niinsanottu ehto-operaattori.

ehto ? toimenpide1 **:** toimenpide2

esimerkiksi seuraava rakenne:

```
if ( x > y )
  string = "Suurempi"
else
  string = "Pienempi"
```

voitaisiin kirjoittaa lyhyemmin muotoon:

```
string = ( x > y ) ? "Suurempi" : "Pienempi"
```

10.3.2 Silmukat

Silmukka voidaan *awk*ssa määritellä kolmella tavalla, käyttäen joko *while*-, *do(-while)*- tai *for*-rakennetta.

While-silmukan rakenne on seuraavanlainen:

while (ehto **)**

komennot

Silmukan aluksi *ehto* tutkitaan ja mikäli se on tosi suoritetaan komennot. Mikäli ehto ei ole koskaan tosi ei silmukan runkoa koskaan suoriteta. Jos komentoja on useampia on ne suljettava kaarisulkuihin { }

```
i=1
while ( i <= 4 ) {
  print $i
  i++
}
```

Do-silmukka (tai *do-while*) on muunnelma *while*-silmukasta, ja sen rakenne on seuraavanlainen:

do

komennot
while (ehto **)**

Do-silmukan ero *while*-silmukkaan on siinä, että tutkittava ehto on silmukan lopussa. Tämä aiheuttaa sen että silmukassa määritellyt komennot suoritetaan ainakin kerran. Kuten *while*-silmukassakin, kaarisulkuja { } on käytettävä, jos suoritettavia komentoja on useampia.

Kolmas *awk*n tuntema silmukka on *for*-silmukka. *For*-silmukan rakenne on:

for (lask_alustus; lask_testaus; lask_kasvatus **)**

komennot

lask_alustus tarkoittaa laskuri muuttujan alustamista.

lask_testaus tarkoittaa laskuri muuttujan testausta silmukan alussa.

lask_kasvatus tarkoittaa laskuri muuttujan kasvatusta silmukan lopussa juuri ennen laskurin testausta *lask_testaus*.

```
for ( i = 1; i <= NF; i++ )
  print $i
```

10.4 Taulukot

Taulukko on muuttuja, johon on mahdollista tallettaa joukko arvoja. Taulukossa oleviin arvoihin voidaan viitata taulukon *indekseillä*. *Awk*ssa taulukon kokoa ei tarvitse määritellä – muuttujaa vain käytetään taulukkona. Taulukkoon sijoitetaan arvoja seuraavalla tavalla.

taulukko[indeksi**]** = arvo

Esim.

```
taulu[1] = "Maanantai"
```

Taulukon arvo voidaan tulostaa seuraavasti:

```
print taulu[1]
```

*Awk*ssa kaikki taulukot ovat assosiatiivisia, mikä tarkoittaa, että taulukon indeksit voivat olla joko numeroita tai merkkijonoja.

Esim.

```
taulu[Ma] = "Maanantai"
```

```
print taulu[Ma]
```

For-silmukasta on olemassa on variantti assosiatiivisten taulukoiden käsittelyyn. Syntaksi on seuraavanlainen:

for (muuttuja **in** taulukko **)**

tehdaan jotain *taulukko[muuttuja]*:lle

Avainsanaa *in* voidaan käyttää myös testattaessa löytyykö tietylle indeksille arvoa taulukosta.

Syntaksi:

indeksi **in** taulu

palauttaa ykkösen (1) jos *taulu[indeksi]* löytyy ja nollan (0) jos ei löydy.

Taulukon luomiseen voidaan käyttää *awk*n sisäistä funktiota *split()*. *Split()* funktion syntaksi on

n = **split(**jono, taulukko, erotin**)**

Missä *jono* on on merkkijono joka jaetaan taulukon *taulukko* elementeiksi. *Jono* jaetaan elementeiksi käyttäen erottimena *erotin*ta. Erotin voi olla kokonainen merkkijonolauseke. Mikäli erotinta ei määritellä käytetään erottimena muuttujaa FS. Taulukon indeksointi alkaa ykkösestä (1) ja jatkuu n:ään.

Esim.

```
BEGIN {
  split (" Ma, Ti, Ke, To, Pe, La, Su", viikon_p, ",")
  for ( paiva in viikon_p )
    print paiva, viikon_p[paiva]
}
```

Taulukosta voidaan poistaa arvoja komennolla:

delete taulukko[indeksi]

10.4.1 Moniulotteiset taulukot

Awk ei suoraan tue moniulotteisia taulukoita, mutta se tarjoaa tavan simuloida moniulotteisia taulukoita.

Esim.

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

```
  tied_taul[NR, i] = $i
  suur_nf = ( NF > suur_nf ) ? NF : suur_nf
}
END {
for (j=1; j <= NR; j++) {
  for (k=1; k <= suur_nf; k++) {
    printf("%s ", tied_taul[j, k])
  }
  printf("\n")
}
printf("3. rivin 2. alkio on: %s \n", tied_taul[3,2])
}
```

Oletetaan että em. esimerkki on tiedostossa *mdarray.awk*, jolloin sitä voidaan käyttää seuraavasti:

```
awk [-Ffs] -f mdarray.awk file
```

mlarray.awk skripti muodostaa tiedostosta *file* 2-ulotteisen taulukon, käyttäen kenttäerottimena *-F* optiolla annettua *fs* merkkijonolauseketta. Muuttujaan *suur_ nf* talletetaan suurin kenttien määrä. END osa tulostaa taulukon rivi kerrallaan.

Em. esimerkki ei luo "aitoa" moniulotteista taulukkoa, vaan siinä konvertoidaan indeksit jonoksi (esim "3" ja "2"), jotka liitetään yhteen käyttäen erottimena systeemimuuttujaa SUBSEP ("indeksierotin", oletuksena \034). Eli [3, 2] tarkoittaa oikeasti "[3\0342]".

10.5 Funktiot

Usein toistuvat ja pitkät komentorutiinit kannattaa määritellä omiksi funktioikseen. Funktion määrittely *awk*ssa on seuraavanlainen.

function funktion_nimi (parametrilista) {
  funktion komennot
}

Itse määritelty funktio voi sisältää *return*-lauseen:

return expr

Funktio voi palauttaa (nimessään) arvon käyttämällä *expr*-osaa. Ilman *expr*-osaa funktio palaa (ilman ennustettavaa arvoa) jatkamaan komentoja funktion kutsun jälkeen. Esim.

```
BEGIN{
  a=1; b=2
```

```
  print ynnaa(a,b)
}
function ynnaa(a,b) {
  return a+b
}
```

Awk tarjoaa myös joukon sisäisiä funktiota. Nämä funktiot voidaan jakaa kahteen tyyppiin: *aritmeettisiin-* ja *merkkijono*funktioihin.

10.5.1 Aritmeettiset funktiot

*Awk*ssa on yhdeksän aritmeettista funktiota, jotka on esitelty seuraavassa.

cos(x) Paluttaa luvun x kosinin (x on radiaaneissa).

sin(x) Paluttaa luvun x sinin (x on radiaaneissa).

exp(x) Palauttaa e potenssiin x.

log(x) Palauttaa luvun x luonnollisen logaritmin.

int(x) Palauttaa luvun x kokonaisosan.

sqrt(x) Palauttaa luvun x neliöjuuren.

atan2(y,x) Palauttaa y/x:n argtangentin väliltä $[-\pi, \pi]$

rand() Palauttaa (pseudo) satunnaisluvun r, missä $0 \leq r < 1$.

srand(x) Alustaa uuden siemenluvun *rand()*-funktiolle. Mikäli siemenlukua ei alusteta, käytetään siemenlukuna aikaa. Palauttaa vanhan siemenluvun.

10.5.2 Merkkijonofunktiot

*Awk*n sisäiset merkkijonofunktiot.

gsub(r,s,t) Vaihtaa jonon s kaikkiin merkkijonolausekkeen r osumiin jonossa t. Palauttaa suoritettujen vaihtojen määrän. Mikäli parametri t puuttuu käytetään $0:aa.

index(s,t) Palauttaa merkkijonon t paikan merkkijonossa s tai nollan mikäli jonoa t ei löydy jonosta s.

length(s) Paluttaa jonon s pituuden, ilman parametriä paluttaa $0:n pituuden.

match(s,r) Palauttaa paikan jonossa *s* josta merkkijonolauseke *r* alkaa tai nollan mikäli merkkijonolauseketta *r* ei löydy. Asettaa arvot systeemimuuttujille RSTART ja RLENGTH.

split(s,a,sep) Jakaa merkkijonon *s* taulokoksi *a* käyttäen kenttäerottimena parametriä *sep*, jos parametri *sep* puuttuu käytetään muuttujan FS arvoa kenttä erottimena. Palauttaa taulukon alkioiden lukumäärän.

sprintf("fmt",expr,...) Formatoi *expr*-lauseet *fmt*n määräämään asuun. Palauttaa tuloksen merkkijonona.

sub(r,s,t) Vaihtaa jonon *s* ensimmäisen merkkijonolausekkeen *r* osuman paikalle jonossa *t*. Mikäli vaihto onnistuu palauttaa ykkösen (1) muuten palauttaa nollan (0). Mikäli parametri *t* puutuu vaihto suoritetaan $0:ssa.

substr(s,p,n) Palauttaa alimerkkijonon merkkijonosta *s* alkaen paikasta *p* maksimissaan pituuteen *n* asti. Mikäli parametri *n* puuttuu palautetaan loppujono.

tolower(s) Muuttaa kaikki isot kirjaimet pieniksi jonossa *s* ja palauttaa uuden jonon.

toupper(s) Muuttaa kaikki pienet kirjaimet isoiksi jonossa *s* ja palauttaa uuden jonon.

10.6 Muita funktioita

Seuraavassa vielä pari muuta hyödyllistä funktiota.

10.6.1 *Getline* funktio

Getline funktiota käytetään uuden syöterivin lukemiseen, sillä voidaan lukea paitsi tavallisesta syöte virrasta myös syöte tiedostosta tai putkesta. *Getline* funktion toimii samaan tapaan kuin *awk*n *next* komento. Molemmat lukevat seuraavan syöterivin, mutta *next* komento palauttaa ohjelman suorituksen skriptin alkuun, kun taasen *getline* lukee rivin muuttamatta suorituksen paikkaa.

Funktion *getline* mahdollisia palautusarvoja ovat:

1 *Getline* onnistui lukemaan rivin.

0 Tiedosto loppui (end-of-file).

-1 Lukemisessa tapahtui virhe.

Esimerkki käyttäjän syötteen lukemisesta:

```
BEGIN { printf ("Anna nimesi: ")
  getline < "-"
  print
}
```

10.6.2 *System()* funktio

System() funktio suorittaa parametrinä annetun systeemikomennon. Suoritetun komennon tulostetta ei kuitenkaan voi käyttää ohjelmassa, vaan *system* funktio palauttaa suoritetun komennon palautusarvon. Ohjelma odottaa komennon suorituksen päättymisen ennenkuin suoritusta jatketaan.

Esimerkki *system* funktiosta:

```
BEGIN { if (system ("mkdir tmp") != 0 )
  print "Komennon suoritus ei onnistunut"
}
```

10.6.3 *Close()* funtio

Close() funktiota voidaan käyttää avoimien tiedostojen ja putkien sulkemiseen. Syitä *close()* funktion käyttöön: Putkia voi olla avoinna vain tietty määrä samanaikaisesti, systeemi rajoittaa avoimien tiedostojen määrän jne.

Esimerkki putken sulkemisesta:

```
BEGIN {
  while ("who" | getline)
    who_out[++i] = $1
  close("who")
  for ( kuka in who_out )
    print who_out[kuka]
}
```