

# Parsing Dostu: AWK!

**Özkan KIRIK**

ozkan ~ enderunix.org

**Parsing Dostu: AWK!**

Özkan KIRIK

Telif Hakkı © 2005 Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.

# İçindekiler

1. Awk'ya bir bakış .....	1
2. Awk'da Kod Blokları .....	2
3. Örneklerle Tekrarlı Blok.....	4
4. Regular Expressions ve Awk.....	5
5. Diziler ve Awk.....	8

# Bölüm 1. Awk'ya bir bakış

Awk, genelde giriş metnini inceleyip, metin içerisinde istediğinin verilerin, belirlediğiniz biçimde düzenler ve STDOUT'a basar.

Awk, giriş metnini bir veri tablosu gibi kabul eder. Tablonun her satırı (Row) bir kayıt gibi yorumlanır. Her sütunu ise Veri Alanı (Field) olarak kabul yorumlanır. Sütunları ayırmak için Alan Ayırıcı (Field Seperator) kullanılıyor. Aksi belirtilmediği halde Field Seperator (FS) boşluk karakteridir.

Bir **ls -l** çıktısı alıp inceleyelim,

```
# ls -l
total 6
-rw-r--r--  1 ryland   wheel      767 Jun  6 00:04 .cshrc
-rw-----  1 ryland   wheel      276 Nov 23 10:16 .history
-rw-r--r--  1 ryland   wheel      248 Jun  6 00:04 .login
```

Bu sonucu, awk'ya girdi olarak verirsek, 4 Kayıttan (Row) oluşan bir veriyi inceliyor olacağız. Awk ile giriş metni incelenirken, her satır için Alan Sayısı (Number Of Fields) tekrar hesaplanır. Bu durumda ilk kayıttan *NR* (Number Of Row) değişkeni 1'i gösterirken, *NF* (Number Of Fields) değişkeni ise 2'yi gösterecektir. İkinci kayıt için, NR = 2, NF = 9 olacaktır

Awk, Alan ayırıcının tekrarlanması durumunda tek bir alan ayırıcı varmış gibi davranır. Örneğin:

```
"a b" ile "a      b" girdileri awk için aynıdır.
```

Awk, alanları 1'den başlayarak numaralandırır. Alanlara erişmek için alan numarasının önüne **\$** işareti eklenir. Örneğin birinci alana erişmek için, **\$1** ifadesi kullanılıyor. Bütün satırı temsilen **\$0** ifadesi kullanılabilir. Yukarıdaki **ls -l**'de ilk iki satırı inceleyelim;

```
NR = 1, NF = 2, $1="total", $2="6"
```

```
NR = 2, NF = 9, $1="-rw-r--r--", $2="1", $3="ryland", $4="wheel", $5="767", $6="Jun", $7="6", $8="00"
```

## Bölüm 2. Awk'da Kod Blokları

Şimdi de, awk'nın, kodlarını nasıl yorumladığından bahsedelim,

Awk, kodlarını 3 aşamada inceler:

1. Giriş Metni İncelenmeden önce çalışacak kod (BEGIN)
2. Metnin Her Satırı için tekrar edilecek kod
3. İnceleme tamamlandıktan sonra çalışacak kod (END)

Örnek bir awk scripti ele alalım:

```
-- ornek.awk --
# Giriş incelenmeden önce çalışacak bölüm
BEGIN{
print "Örnek AWK Programına Hoşgeldiniz."
print "Aşağıda, giriş metninin sadece birinci sütunları görüntülenecektir."
}

# Metnin Her Satırı için tekrar edilecek kod
{
print $1
}

# İnceleme tamamlandıktan sonra çalışacak kod
END{
print "İnceleme tamamlandı."
}
-----
```

Scripti çalıştırsak;

```
# ls -l | awk -f ../ornek.awk
Örnek AWK Programına Hoşgeldiniz.
Aşağıda, giriş metninin sadece birinci sütunları görüntülenecektir.
total
-rw-r--r--
-rw-----
-rw-r--r--
-rw-r--r--
İnceleme tamamlandı.
#
```

Aynı işi **awk** ile commandline olarak yapmak istersek, scripti tek tırnak tırnak içine alarak yazın. Mesela;

```
# ls -l | awk '{print $1}'
```

Başka bir örnek ele alalım. **ls -l** nin çıktısında 5. sütun dosya boyutlarını gösteriyor. Bu değerleri toplayıp, toplam dosya sayısını ve boyutunu yazdıralım.

```
-- ornek2.awk --
BEGIN {
```

```
toplamboyt=0
}

{
if (NR != 1) {
toplamboyt=toplamboyt+$5
}
}

END {
dosyasayisi=NR-1 # 1 çıkartmamızın nedeni, ls -l 'de ilk satırın (total ile başlayan satır) dosya o
print "Toplam,"
print dosyasayisi" dosya, "toplamboyt" byte."
}
-----
```

Scripti çalıştıralım,

```
# ll | awk -f ../ornek2.awk
Toplam,
4 dosya, 2018 byte.
#
```

## Bölüm 3. Örneklerle Tekrarlı Blok

Bu sefer biraz daha farklı bir örnek ele alalım. Giriş metninde kayıt, tek bir satırda değil de, 2 ya da daha fazla satıra yayılmışsa bu kayıtların incelenmesi olayını inceleyelim;

```
-- giris.txt --
Arayan numara 02121112233
5 cevapsiz cagri
Arayan numara 02164445566
3 cevapsiz cagri
Arayan numara 02127778899
19 cevapsiz cagri
-----
```

Yukarıdaki şekilde bir girişimiz olsun. Bizden istenen çıktı ise, "**02121112233 numarasından 5 çağrı var**" şeklinde her arama tek satıra gelecek şekilde bir çıktı.

```
-- ornek3.awk --
{
if ( (NR%2)==1 ) {
telno=$3
} else {
}
aramasayisi=$1
print telno" numarasından "aramasayisi" çağrı var."
}
}
-----
```

% operatörü, soldaki sayının, sağdaki sayıya bölümünden kalan sayıyı veriyor (*modulus*). Bir sayının ikiye bölümünden kalan o sayının tek mi çift mi olduğuna dair fikir edinmemizi sağlıyor. Bu durumda tek satırlarda telefon numarası var, çift satırlarda ise cevapsız arama sayısı yer alıyor. Tek satırlarda telefon numarasını bir değişkene atıyoruz, çift satırlarda ise arama sayısını öğrenip, sonucu ekrana yazdırıyoruz.

Şimdi scripti çalıştıralım,

```
# cat giris.txt | awk -f ornek3.awk
02121112233 numarasından 5 çağrı var.
02164445566 numarasından 3 çağrı var.
02127778899 numarasından 19 çağrı var.
#
```

# Bölüm 4. Regular Expressions ve Awk

Regex'ler awk'da yazılırken // arasına alınarak yazılır. (Örnek: /^[0-9]/ gibi).

Awk'da, bir ifadenin verilen regex'le eşleşip eşlemediğini kontrol etmek için, ~ (tilda) operatörünü kullanıyoruz. Mesela giriş metninde, rakam ile başlayan satırları gösteren, diğerlerini göstermeyen bir awk kodu yazalım:

```
-- giris.txt --
EnderUNIX Yazılım Geliştirme Takımı
  3numaralı sayfa
1209348 nolu evrak
Bu satir yazilmayacak degil mi
-----
```

```
-- ornek4.awk --
{
if ( $1 ~ /^[0-9]/ ) {
print
}
}
-----
```

Scripti çalıştırdığımızda;

```
# cat giris.txt | awk -f ornek4.awk
  3numaralı sayfa
1209348 nolu evrak
#
```

Kodları incelersek, **if** satırında, (**\$1 ~ /^[0-9]/**) koşulu, 1.ci alanda (**\$1**) **^[0-9]** regex'inin uyup uymadığını kontrol ediyor.

Regex'in kullanıldığı bir başka nokta ise, değiştirme işlemi (string substitution). Awk, değiştirme işlemi için 2 fonksiyona sahip; **sub** ve **gsub** fonksiyonları.

"**sub**" fonksiyonu verilen ifade içerisinde, aranan regexi ilk bulduğunda değiştirme işlemi yapar ve aramayı durdurur. Aynı ifade içerisinde daha sonra aynı regex'e uyan başka bir kelime varsa onlar değiştirilmez. "**gsub**" fonksiyonu ise verilen ifade içerisinde eşleşen bütün regexleri değiştirir. Her iki fonksiyonda da eşleşen kelime & işareti ile temsil edilir.

Örneğin, **ls -l** nin çıktısında bir takım değiştirme işlemlerini yapalım:

```
# ls -l
total 6
-rw-r--r--  1 ryland   wheel      767 Jun  6 00:04 .cshrc
-rw-----  1 ryland   wheel      276 Nov 23 10:16 .history
-rw-r--r--  1 ryland   wheel      248 Jun  6 00:04 .login
#
```

```
-- ornek5.awk --
{
sub(/-r/, "R", $0);
print
}
```

```
}
-----
```

Scripti çalıştırırsak,

```
# ls -l | awk -f ../ornek5.awk
total 6
Rw-r--r-- 1 root wheel 767 Nov 24 09:20 .cshrc
Rw----- 1 root wheel 276 Nov 24 09:20 .history
Rw-r--r-- 1 root wheel 975 Nov 24 09:21 .shrc
Rw-r--r-- 1 root wheel 0 Nov 24 09:21 .vimrc
#
```

Yukarıda bahsettiğimiz gibi, sub fonksiyonu sadece ilk eşleşen kelimeyi değiştiriyor.

```
-- ornek6.awk --
{
gsub(/-r/, "R", $0);
print
}
-----
```

Yukarıdaki örneği uyguladığımızda,

```
# ls -l | awk -f ../ornek.awk
total 6
RwR-R-- 1 root wheel 767 Nov 24 09:20 .cshrc
Rw----- 1 root wheel 276 Nov 24 09:20 .history
RwR-R-- 1 root wheel 975 Nov 24 09:21 .shrc
RwR-R-- 1 root wheel 0 Nov 24 09:21 .vimrc
#
```

görüldüğü gibi eşleşen tüm kelimeler değiştirilmiş.

Bir de & işaretinin kullanımı hakkında bir örnek verelim:

```
-- ornek7.awk --
{
    for (i=1; i<=NF; i++) {
        sub(/[0-9]/, "#\&", $i)
    }
    print
}
-----
```

Örnek7'yi çalıştıralım;

```
# ls -l | awk -f ../ornek7.awk
total #6
-rw-r--r-- #1 root wheel #767 Nov #24 #09:20 .cshrc
-rw----- #1 root wheel #276 Nov #24 #09:20 .history
-rw-r--r-- #1 root wheel #975 Nov #24 #09:21 .shrc
-rw-r--r-- #1 root wheel #0 Nov #24 #09:21 .vimrc
#
```

Yaptığımız işlem,  $i$  değişkenini 1'den **NF** (*Number of Fields*) değerine kadar arttırarak değiştirerek her alanda tek tek sub fonksiyonu aracılığı ile rakamla başlayan ifadelerin önüne # karakteri ekliyoruz. Burada \& işareti, eşleşen kelimeyi temsil ediyor.

# Bölüm 5. Diziler ve Awk

Awk'da dizi anahtarları rakamların yanı sıra kelimeler de olabiliyor. (Örneğin `dizi["adi"]="Özkan"`)

Dizi işlemlerinin kullanıldığı, ifconfig'in çıktısını parse eden bir script üzerinde çalışalım:

```
-- ornek8.awk --
{
# Birinci alanın son karakteri ":" ise ve ikinci alanın başlangıcı "flags" ise
if ( ($1 ~ /\:$/) && ($2 ~ /^flags/) ) {
# yukarıdaki koşul doğruysa birinci alan arayüz ismidir
arayuz=$1
# arayuz isminin sonundaki ":" karakterini kaldıralım
sub(/\:/,"",arayuz)
}
# Bütün alanları sırayla tara
for (x=1; x<=NF; x++) {
# Eğer alanın değeri "inet" ise
if ($x=="inet") {
# bir sonraki alanda ip adresi var. alan sayacı olan x'i 1 arttır.
x++
# arayüz - ip adresi çiftini diziye at.
ipadresleri[arayuz]=$x
}
}
}
END {
for (arayuz in ipadresleri) {
print "Arayüz: "arayuz"\t IP Adresi: "ipadresleri[arayuz]
}
}
-----
```

Yukarıdaki scriptin çıktısı:

```
# ifconfig | awk -f ../ornek.awk
Arayüz: lo0      IP Adresi: 127.0.0.1
Arayüz: bge0     IP Adresi: 172.16.0.2
Arayüz: bge1     IP Adresi: 192.168.0.1
Arayüz: gif0     IP Adresi: 1.2.3.4
#

# ifconfig
bge0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
    options=1a<TXCSUM,VLAN_MTU,VLAN_HWTAGGING>
    inet 172.16.0.2 netmask 0xfffff00 broadcast 172.16.0.255
    ether 00:14:c2:60:83:9b
    media: Ethernet autoselect (1000baseTX <full-duplex>)
    status: active
bge1: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
    options=1a<TXCSUM,VLAN_MTU,VLAN_HWTAGGING>
    inet 192.168.0.1 netmask 0xfffff00 broadcast 192.168.0.255
    ether 00:14:c2:60:83:9a
```

```
media: Ethernet autoselect (100baseTX <full-duplex>)  
status: active  
lo0: flags=8049<UP,LOOPBACK,RUNNING,MULTICAST> mtu 16384  
    inet 127.0.0.1 netmask 0xff000000  
gif0: flags=8051<UP,POINTOPOINT,RUNNING,MULTICAST> mtu 1500  
    tunnel inet 1.2.3.4 --> 5.6.7.8
```