یک زبان برنامه نویسی تفسیری است که در سال ۱۹۷۷ ایجاد شده است. ساده ترین شکل استفاده از آن به شکل زیر است

awk ‘{ print $0 }’ ali.txt

که این دستور کل فایل متنی را برمیگرداند

awk ‘{ print $0 }’ ali.txt

این دستور فقط ستون اول را چاپ می کند

می توان یک فایل به عنوان ورودی برای command ها ایجاد کرد مثل زیر

BEGIN{

printf “this is begin\n”

}

{

Printf $1"\t"$2

}

END{

Printf “ this is the end”

}

و دستور awk آن مانند زیر است

awk -f com(name of file of commands) ali.txt(name of text file)

در این فایل دستورات ما می توانیم متغیر هم تعریف کنیم و از آن استفاده کنیم

BEGIN{

printf “this is begin\n”

}

{

Printf $1"\t"$2

}

END{

Printf “ this is the end”

}

ما می توانیم یک متغیر تعریف کنیم

BEGIN{

printf “this is begin\n”

count=0

}

{

Count++

}

END{

printf “ this is the end”

print count

}

فرق print و printf در این است که دستور print صرفا یک دستور در زبان awk است و برای پرینت کردن محتوی جلوی آن استفاده می شود در صورتی که printf دستور جامعتری است و مانند printf در زبان c می توان متغیر و یک سری پارامتر بگیرد

می توان یک متغیر را از ورودی دریافت کرد که این متغیر می تواند به وسیله خود کاربر تعریف شود و یا از متغیر های محیطی باشد.

BEGIN{

printf “this is begin\n”

count=0

}

{

Count++

}

END{

printf “ this is the end”

print count

printf “variable1 is : %s\n”, var1

}

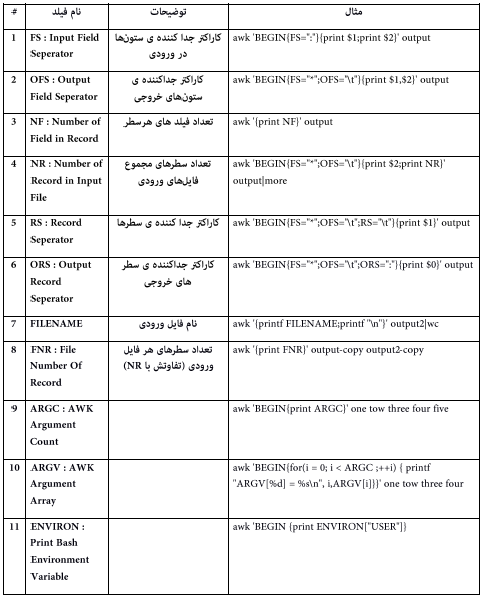
حالا با دستور زیر می توانیم variable را از بیرp[-ون مقدار دهیم

awk -v var1=new\_text -f com ali.txt

ویا حتی می توانیم از متغیر های محیطی استفاده کنیم

awk -v var1=$USER -f com ali.txt

در زبان awk ما یک سری متغیر های داخلی داریم



به صورت پیش فرض AWK جدا کننده فیلدها space و جداکننده رکوردها enter است. برای اینکه به awk بفهمانیم باید از ‘,’ به جای space به عنوان field separator استفاده کند می بایست مقدار متغیر FS = ‘,’ را اصلاح کنیم. بنابراین فایل دستورات به شکل زیر خواهد بود که حالا می تواند یک فایل از نوع CSV را به عنوان ورودی بگیرد.

BEGIN{

FS=”,”

}

{

print $1, $2, $3

}

END{

}

که به این ترتیب ما به awk یک فایل csv می دهیم و سه ستون اول آن را چاپ می کند.

awk -f com(command file) ali.csv

متغیر OFS به awk میگوید وقتی که میخواهد خروجی را چاپ کند از چه جداکننده ای بین فیلدها استفاده کند. ( مثلا OFS=”\*” در خروجی به جای space بین فیلدها ستاره اضافه می کند)

متغیر NF تعداد فیلدهای یک رکورد را اعلام میکند و $NF مقدار آخرین فیلد از رکورد را نمایش می دهد. بنابراین اگر در قسمت میانی باشد به ازای هر رکورد با NF می توانیم تعداد فیلدهای آن رکورد را داشته باشیم.

با متغیر NR می توانیم تعداد رکوردهای را داشته باشیم. بنابراین اگر از این متغیر در قسمت میانی دستور awk استفاده کنیم به هر رکورد که بالا می رود تعداد رکوردها تا آن خط را نمایش می دهد.

متغیر RS معرف جداکننده رکوردها می باشد که به صورت پیش فرض این متغیر برابر است با enter. اگر بخواهیم به awk بفهمانیم فایل ورودی برای جدا کردن رکورها مثلا از : استفاده کند میبایست بنویسیم RS=”:” که به این ترتیب اعلام می کنیم جداکننده رکوردها در فایل وروردی : است و اگر بخواهیم جداکننده در خروجی را عوض کنیم می بایست مقدار متغیر ORS را مقداردهی کنیم.

متغیر FILENAME (می بایست بدون $ بیاید) برای خواندن نام فایل استفاده می شود و تنها می توان از این متغیر در قسمت میانی استفاده کرد.

متغیر FNR نیز تعداد رکوردهای فایل را می شمرد و تفاوت آن با NR وقتی مشخص می شود که بیش از یک فایل به عنوان ورودی به awk داده شود که در این صورت FNR به ازای هر فایل از ابتدا شمردن را شروع می کند در صورتی که NR برای کل ورودی های awk رکوردها را در ادامه هم می شمرد.

متغیر ARGC تعداد آرگومان های ورودی دستور awk را نمایش می دهد و ARGV لیست آرگومان های ورودی را نمایش می دهد

BEGIN{

print ARGC

print ARGV[0]

print ARGV[1]

}

به وسیله ENVIRON می توان یک متغیر محیطی را چاپ کرد.

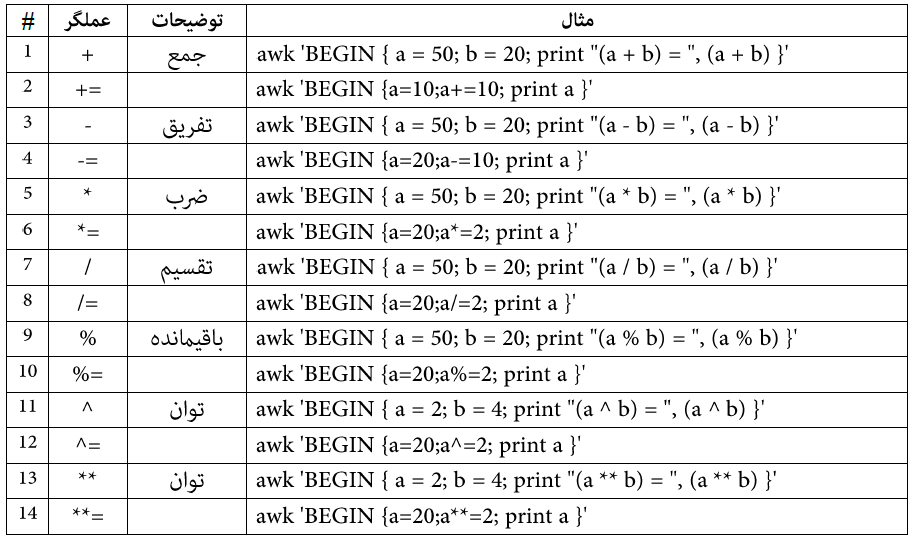
BEGIN{

print ENVIRON “HOME”

}

که در خروجی به ما مقدار متغیر محیطی HOME را نمایش می دهد.

# عملگرهای ریاضی در awk



می توانیم از عملیات ریاضی در داخل دستورات awk استفاده کنیم که این عملیات در جدول بالا ارائه شده اند. برای مثال به دستور زیر دقت کنید.

BEGIN{

printf ( “(%d + %d) = %d \n”, $1, $2, ($1+$2))

}

که به این ترتیب مقدار ستون اول در متغیر اول و مقدار ستون دوم در متغیر دوم قرار میگیرد و در نهایت نتیجه جمع در متغیر سوم نوشته شده و خروجی به شکل روبرو چاپ می شود.

(3 + 4) = 7

همچنین می توان از یک سری متغیر نیز در بدنه دستورات awk استفاده کرد. برای مثال داریم

BEGIN{

A=10

}

{

printf (“%d + 10 = %d \n”, a, a+=10)

}

در ضمن عملگر های ++ و -- نیز مانند آنچه در زبان C++ استفاده می شود قابل استفاده است.

در زبان awk می توان آرایه تعریف کرد و سپس محتویات آن را فراخوانی کرد.

BEGIN{

}

{

Num[0]= $1

Num[1]= $2

Num[3]= $3

Name[0]= “ali”

Name[1]= “farahani”

print Num[0], Num[1], Num[2]

print Name[0], Name[1]

}

این دستورات میانی به ازای هر خط در فایل ورودی awk یکبار اجرا می شود. می توان مقادیر مرتبط با اندیس های آرایه را از متغیرهای قابل فهم برای awk در نظر گرفت و یک آرایه دو بعدی ساخت. فرض کنید یک فایل با ورودی دو بعدی داریم مثل زیر.

[1,2]

[3,4]

BEGIN{

Col=0

}

{

Num[NR, col]=$1

Num[NR, ++col]=$2

Num[NR, col]=$3

Num[NR, ++col]=$4

}

END{

delete [NR, col] # این دستور مقدار محتوی این اندیس از آرایه را خالی می کند

}

می توان از دستورات if و else/if استفاده کرد

BEGIN{

Num=1

}

{

If (Num == 1)

print “num is 1”

else if (num ==2)

print “num is 2”

else

print “I don’t know’

}

از شروط زیر در ساختار if میتوان استفاده کرد



این زبان regex را پشتیبانی می کند می بایست دستورات حاوی regex را مابین دو // قرار داد

(نکته echo به صورت پیش فرض هرچه جلویش هست را چاپ می کند ولی اگر با option -e بیاید می تواند دستورات مثل \n را داخل متن متوجه بشود)

Echo -e “knife\nknow\nfun\nfin\nfan\nine” | awk ‘/^kn/’

این دستور هر چه با kn شروع شود را نمایش می دهد

Echo -e “knife\nknow\nfun\nfin\nfan\nine” | awk ‘/$in/’

این دستور هر چه با in تمام شود را نشان می دهد

echo -e “knife\nknow\nfun\nfin\nfan\nine” | awk ‘/.a./’

این دستور هر چه شامل یک a در میان کلمه اش باشد را نشان می دهد

echo -e “call\ntall\nball\” | awk ‘/[BC]all/’

در این دستور هر کدام از ورودی ها که در ابتدا B و C دارند و در انتهای آنها all است را نمایش می دهد در ضمن در براکت می توان از – به عنوان بازه نیز استفاده کرد [A-C] یعنی مواردی که از A تا C هستند را نمایش می دهد

حالا اگر ^ را در ابتدای یک براکت بگذاریم معنی آن است که کلماتی را انتخاب کند که با این دو مورد شروع نشود و در مثال زیر پاسخ برابر است با call

echo -e “call\ntall\nball\” | awk ‘/[^A-C]all/’

و کاراکتر \* به معنی صفر و یا بیشتر از هر نوع کاراکتری است

و () نیز برای گروه بندی استفاده می شود مثال

echo -e “apple pie\napple tart\napple cake” | awk ‘/apple (pie|cake)/’

که در خروجی apple pie و apple cake را نمایش می دهد.

میتوان از regex ها در داخل ساختارهای if استفاده کرد برای مثال

BEGIN{

}

{

if ( $1 ~ /f.n/) # character ~ means = when we want to match in regex

print “matched”

else

print “not matched”

}

در awk امکان خروجی دستور را در یک فایل دیگر ذخیره کرد

awk ‘BEGIN{ print “Hello World” > ./newfile}’

همچنین می توان از pipe line استفاده نمود مثلا می توان دستور زیر را نوشت

awk ‘BEGIN{ print “Hello World” | tr [a-z] [A-Z]}’

که به این ترتیب تمام حروف کوچک به بزرگ تبدیل می شوند.

می توانیم برای خروجی awk یک exit کد نیز در نظر بگیریم (در لینوکس هر دستوری که اجرا می شود یک exit کد به لینوکس بر می گرداند که نشاندهنده وضعیت تمام شدن دستور است که معمولا اگر این عدد 0 باشد به معنی درست اجرا شدن دستور آخر است این متغیر را می توان با echo $? مشاهده کرد)

awk ‘{ if (1==1) print “ok” ; exit 30}’ ali.txt

که در این حالت پس از اجرای درست این دستور مقدار 30 به عنوان exit code به لینوکس برگردانده می شود.

در awk می توان چند فایل را به عنوان ورودی داد و در داخل فایل دستورات چک کرد اگر یک شرطی درست بود دیگر بقیه موارد را درفایل اول چک نکند و به فایل دوم برود

BEGIN{

}

{

if ($1 == 6)

nextfile

}

END{

}

به این ترتیب اگر در فیلد اول یکی از رکوردهای فایل اول عدد 6 دیده شود دیگر این فایل بررسی نمی شود به فایل دوم منتقل می شود.

در awk می توان از دستورات حلقه دقیقا مشابه آنچه در زبان C++ وجود دارد استفاده نمود.

BEGIN{

i=0

while(i<10)

{

print i;

print i++;

}

}

BEGIN{

for(i=0;i<10;i+=3)

{

print i

}

}

همچنین دقیقا مانند زبان C++ می توان از دستورات continue و break و exit‌ استفاده کرد.

BEGIN{

sum = 0;

for (i = 0; i < 10; ++i) {

sum += i;

printf ("i = %d ",i)

if(sum > 30)

{

print "Break",sum

break;

}

if(sum ==15)

{

print "Continue",sum

continue

}

if(sum==45)

{

print "Exit",sum

exit 20

}

else

print "Sum =", sum

}

یک سری از دستورات ریاضی به صورت پیش فرض در awk تعریف شده است مثلا دستورات مثلثاتی مانند sin و cos (البته باید ورودی آن را بر اساس رادیان داد مثل pi/2 برای ۹۰ درجه)

دستورات زیر در این زبان به صورت پیش فرض تعریف شده است

* sin
* cos
* int # که خروجی عدد اعشاری را صحیح می کند
* sqrt # که جذر بر میگرداند
* rand # که یک عدد تصادفی تولید می کند البته باید ابتدا از تابع srand استفاده کرد (مانند زبان C++)

همچنین یک سری از دستورات جایگزینی برای string ها در این زبان تعریف شده است

* sub که این دستور ۳ ورودی می گیرد

sub (“world”, “iran”, str)

که کلمه اول را در متن فایل str را به کلمه دوم تبدیل میکند البته تنها اولین بار که این دستور را ببیند تبدیل می کند و مابقی موارد تفاوت ایجاد نمی شود. اگر به جای sub از دستور gsub استفاده شود در تمام متن به جای تمام مواردی که world را مشاهده کند کلمه iran را جایگزین می کند

نکته : وقتی این str به عنوان ورودی به این دستورات داده می شود (اشاره گر آن به تابع پاس داده می شود بنابراین) محتوی خود str تغییر می کند.

دستور substr یک string میگیرد و از عدد اول تا عدد آخر را چاپ میکند

new\_sub = substr(str, 1,10)

#3-SubStr

print "Substitude String : "

str="My Name Is Farahani"

subs=substr(str,1,14)

printf ("Char 1 to 10 is : %s",subs)

دستور match که برای پیدا کردن مقدار یک Str در داخل یک str دیگر استفاده می شود که اولین نقطه ای که این مقدار دیده می شود را بیان می کند

Result = match(str, str1)

#4-Match

print "Match String Func : "

str= "One Two Three"

subs = "Three"

result = match(str,subs)

printf ("String %s Found at %d Location",subs,result)

می توان از دستور split استفاده کرد که به این ترتیب این دستور ۳ آرگومان می گیرد که اولی فایل مبدا و دومی آرایه و سومی جداکننده که با وسیله آن فایل اول را جدا کرده و در آرایه می نویسد

#5-split

print "Split Func : "

str = "one,two,three,four"

split(str,arr,",")

print "Array Contain Following Values : "

for (i in arr){

print arr[i]

}

و از دستور tolower می توان استفاده کرد تا حروف را به کوچک تبدیل کرد ( بالعکس از toupper)

#6-toLower

print "To Lower : "

str = "HELLO WORLD!"

printf ("Lowercase String is : %s\n\n",tolower(str))

می توان از length‌ برای پیدا کردن طول string‌ استفاده کرد

#8-length

print "Length Func : "

str = "hello world!"

printf ("String Length : %d\n\n",length(str))

می توان از تابع sort برای مرتب کردن استفاده کرد که دو نوع assort و asorti دارد که اولی بر اساس محتوی هر خانه از آرایه عملیات مرتب سازی را انجام میدهد و دومی بر اساس ایندکس هر خانه (زیرا در این زبان ایندکس می توان حروف و یا حتی کلمات باشد مانند آنچه در dict در زبان پایتون داریم)

#9-asort

print "Asort Func : "

sarr[0] = "Three"

sarr[1] = "One"

sarr[2] = "Two"

print "Array elements before sorting : "

for (i in sarr){

print sarr[i]

}

asort(sarr)

print "Array elements after sorting :"

for (i in sarr){

print sarr[i]

}

#10-asorti

print "AsortI Func : "

iarr["one"] = "Three"

iarr["two"] = "One"

iarr["three"] = "Two"

print "Array elements before sorting : "

for (i in iarr){

print iarr[i]

}

asorti(iarr)

print "Array elements after sorting :"

for (i in iarr){

print iarr[i]

}

با استفاده از دستور strtonum می توان یک داده string را داد و آن را به مقدار عددی آن تبدیل کرد. می توان در این مدل اعداد را از مبنای ه 8 تایی یا 16 تایی نیز تبدیل کرد.

#11-strtonum

print "Convert String To Num : "

print "Decimal num (Decimal)= " strtonum("123")

print "Decimal num (Octal)= " strtonum("0123")

print "Decimal num (Hexadecimal)= " strtonum("0x123")

در این زبان عملیات منطقی مبتنی بر بیت نیز قابل اجرا می باشد.

BEGIN{

num1=2

num2=5

#010

#101

#000 = 0

#111 = 7

printf ("Num1 = %d, Num2 = %d\n-----\n",num1,num2)

printf ("%d AND %d = %d\n",num1,num2,and(num1,num2))

printf ("%d OR %d = %d\n",num1,num2,or(num1,num2))

#010

#100 = 4

#010

#1000 = 8

printf ("lShift(%d) by 1 = %d\n",num1,lshift(num1,3))

#010

#001 = 1

printf ("rShift(%d) by 1 = %d\n",num1,rshift(num1,1))

printf ("\n-----\n")

}

{

}

END{

}

در زبان awk تابعی برای برگرداندن زمان سیستم و اجرای دستورات سیستم نیز وجود دارد.

BEGIN{

print strftime("Time = %m/%d/%Y %H:%M:%S", systime())

res1 = system("echo $HOME");

res2 = system("date");

res3 = system("ls -l");

print "Return Value 1: " res1 # return 0

print "Return Value 2: " res2 # return 0

print "Return Value 3: " res3 # return 0

}

# تعریف توابع دلخواه

در زبان Awk می توان توابع دلخواه کاربر را به وسیله کلمه کلیدی function نیز تعریف کرد. به این ترتیب که ابتدا (قبل از BEGIN این توابع تعریف می شوند و سپس در قسمت های بدنه awk (یعنی BEGIN یا قسمت میانی یا END) می توان از این توابع استفاده کرد.

function find\_min(num1,num2){

if(num1<num2)

return num1

else

return num2

}

function find\_max(num1,num2){

if(num1>num2)

return num1

else return num2

}

function main(num1,num2){

#find Minimum Number

result=find\_min(num1,num2)

printf ("Minimum = %d\n",result)

#find Maximum Number

result=find\_max(num1,num2)

printf ("Maximum = %d\n",result)

}

BEGIN{

printf("Num1=10,Num2=20\n")

main(10,20)

exit 10

}

{

}

END{