Sed et awk

1. Introduction

Les classiques

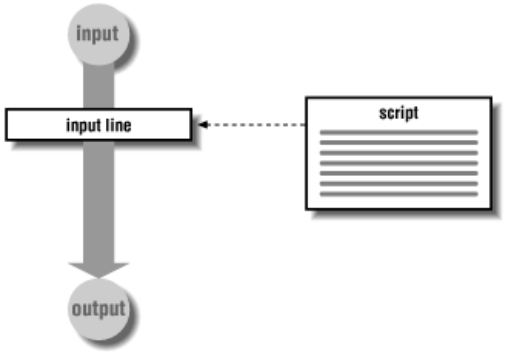
Sed est un éditeur de flux comme son nom l'indique : Stream Editor.

Awk c'est Aho, Weinberger et Kernighan du nom de ses développeurs. Awk est un langage de traitement de flux.

Sed et awk ont beaucoup de choses en commun car il tire tous les 2 leur origine de l'éditeur de ligne ed.

Une des options commune à sed et awk est l'option –f permettant de spécifier un fichier de script.

```
sed –f scriptfile inputfile
```



Utilisation de sed

La substitution

L'utilisation la plus classique de sed est la substitution.

```
sed 's/ancienne chaine/nouvelle chaine/' inputfile
```

On peut définir plusieurs opérations :

```
sed 's/ancienne chaine/nouvelle chaine/ ; s/ ' inputfile
```

Restriction

Restreindre à une ligne particulière

```
sed '3 s/[0-9][0-9]*//' <file >new
```

Restreindre à toutes les lignes commençant par # :

```
sed '/^#/ s/[0-9][0-9]*//'
```

Restreindre à un intervalle de ligne :

```
sed '1,100 s/A/a/'
```

Utiliser un script

```
sed –f sedscript <old > new
```

sedscript :

```
# sed comment - This script changes lower case vowels to upper case
s/a/A/g
s/e/E/g
s/i/I/g
s/o/O/g
s/u/U/g
```

Suppression:

Suppression des lignes 11 jusqu'à la fin:

```
sed '11,$ d' <file
```

Suppression de toute les lignes commençant par # :

```
sed '/^#/ d' < file
```

Utilisation de awk

Utilisation simple

Affichage d'un champ d'une ligne

```
awk '{print $1}' input_file
```

Affichage de ligne contenant une chaine spécifique

```
awk '/MA/' input_file
```

Combinaison des 2

```
awk '/MA/ { print $1 }' list
```

Spécification du délimiteur

```
awk –F, '/MA/ { print $1 }' list
```

```
awk 'BEGIN {FS=,} /MA/ { print $1 $2}'
```

Afficher les champs sur une même ligne

```
awk –F, '/MA/ { print $1,$2,$3 }' list
```

```
$ echo a b c d | awk 'BEGIN { one = 1; two = 2 }
> { print $(one + two) }'
c
```

Afficher les champs sur une ligne différente

```
awk –F, '/MA/ { print $1 ; print $2 ; print $3 }' list
```

Utilisation avancée

Hello World

```
echo 'this line is ignored' > test
awk '{print "Hello, world" }' test
```

```
echo "Hello World" > test
awk '{print}' test
```

```
awk 'BEGIN {print "Hello world"}'
```

Pattern matching

```
echo '/^$/ {print "This is a blank line."} > awkscr
awk –f awkscr test
```

```
cat > awkscr
# test for integer, string or empty line.
/[0-9]+/ { print "That is an integer" }
/[A-Za-z]+/ { print "This is a string" }
/^$/ { print "This is a blank line." }
```

Opérations

```
# Count blank lines.
/^$/ {
print x += 1
}
```

```
# Count blank lines.
/^$/ {
++x
}
END {
print x
}
```

Calcul de moyennes

```
john 85 92 78 94 88
andrea 89 90 75 90 86
jasper 84 88 80 92 84
```

```
# average five grades
{ total = $2 + $3 + $4 + $5 + $6
avg = total / 5
print $1, avg }
```

Conditions

```
{ total = $2 + $3 + $4 + $5 + $6
avg = total / 5
if (avg >= 90) grade = "A"
else if (avg >= 80) grade = "B"
else if (avg >= 70) grade = "C"
else if (avg >= 60) grade = "D"
else grade = "F"
print $1, grade }
```

Boucles

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

```
BEGIN {
do {
++x
print x
} while ( x <= 4 )
}
```

```
for ( i = NF; i >= 1; i-- )
print $i
```

Calcul de factoriel

```
awk '# factorial: return factorial of user-supplied number
BEGIN {
# prompt user; use printf, not print, to avoid the
newline
printf("Enter number: ")
}
# check that user enters a number
$1 ~ /^[0-9]+$/ {
# assign value of $1 to number & fact
number = $1
if (number == 0)
fact = 1
else
fact = number
# loop to multiply fact*x until x = 1
for (x = number - 1; x > 1; x--)
fact *= x
printf("The factorial of %d is %g\n", number, fact)
# exit -- saves user from typing CRTL-D.
exit
}
# if not a number, prompt again.
{ printf("\nInvalid entry. Enter a number: ")
}'
```

Les tableaux

```
{ total = $2 + $3 + $4 + $5 + $6
avg = total / 5
student_avg[NR] = avg}
END {
for ( x = 1; x <= NR; x++ )
class_avg_total += student_avg[x]
class_average = class_avg_total / NR
for ( x = 1; x <= NR; x++ )
if (student_avg[x] >= class_average)
++above_average
else
++below_average
print "Class Average: ", class_average
print "At or Above Average: ", above_average
print "Below Average: ", below_average
}
```

Autre exemple:

```
# grades.awk -- average student grades and determine
# letter grade as well as class averages.
# $1 = student name; $2 - $NF = test scores.
# set output field separator to tab.
BEGIN { OFS = "\t" }
# action applied to all input lines
{
# add up grades
total = 0
for (i = 2; i <= NF; ++i)
total += $i
# calculate average
avg = total / (NF - 1)
# assign student's average to element of array
student_avg[NR] = avg
# determine letter grade
if (avg >= 90) grade = "A"
else if (avg >= 80) grade = "B"
else if (avg >= 70) grade = "C"
else if (avg >= 60) grade = "D"
else grade = "F"
# increment counter for letter grade array
++class_grade[grade]
# print student name, average and letter grade
print $1, avg, grade
}
# print out class statistics
END {
# calculate class average
for (x = 1; x <= NR; x++)
class_avg_total += student_avg[x]
class_average = class_avg_total / NR
# determine how many above/below average
for (x = 1; x <= NR; x++)
if (student_avg[x] >= class_average)
++above_average
else
++below_average
# print results
print ""
print "Class Average: ", class_average
print "At or Above Average: ", above_average
print "Below Average: ", below_average
# print number of students per letter grade
for (letter_grade in class_grade)
print letter_grade ":", class_grade
[letter_grade] | "sort"
}
```