awk

- Objet : awk
- Niveau requis :
  débutant, avisé
- Commentaires : *Cet utilitaire a été créé dans le but de remplacer les commandes grep et sed.*
- Débutant, à savoir : Utiliser GNU/Linux en ligne de commande, tout commence là !.
- Suivi :
  - Création par smolski 18/10/2012)
  - Testé par **smolski** le 28/10/2013)
  - Testé par **MicP** le 28/10/2013)
- Commentaires sur le forum : Lien vers le forum concernant ce tuto[1)]

Introduction

awk est un langage de programmation datant de 1977, date de son apparition dans le monde Unix. Il tire son nom des trois programmeurs qui l'ont développé : Alfred V. Aho, Peter J. Weinberger et Brian W. Kernighan.
Cet utilitaire a été créé dans le but de remplacer les commandes grep et sed. Sa grande souplesse lui a permis de connaître un succès immédiat. Et de nouvelles versions sont apparues au fil du temps : nawk et gawk aujourd'hui.

Aujourd'hui encore, cet utilitaire est toujours utilisé du fait de sa ressemblance avec le langage C, de sa souplesse et de sa présence sur la majorité des systèmes d'exploitation Unix. Il est encore utilisé en administration système et dans les scripts Shell en tant que commande.

Exercices

Créez[2)] le fichier `file.txt` contenant la liste suivante :

file.txt

```
Nom             Genre           Age
-----------------------------------------
CAMILLE         M               7
CHLOE           F               12
CLARA           F               11
CLEMENT         M               7
EMMA            F               6
THEO            M               8
```

Extraire des colonnes

Extraire des données d'un fichier, par exemple les 2 premières colonnes :

```
awk '{ print $1, $2 }' file.txt
```

retour de la commande

```
Nom Genre
---------------------------------------
CAMILLE M
CHLOE F
CLARA F
CLEMENT M
EMMA F
THEO M
```

- $1 correspond à la première colonne, $2 la seconde, $3 la troisième...
- $0 correspond à la ligne entière

Dans le format de la sortie, les tabulations sont remplacées par un espace qui est le séparateur de sortie par défaut.

Par défaut, les espaces et tabulations contigües sont considérées comme un séparateur unique. Il s'agit de la seule exception.

Filtres et regexp

ATTENTION ! Vérifiez toujours l'écriture de chaque script donné ici avant de l'appliquer !
Lien utile : **Les Regexp.** *Et c'est pas la peine de faire Ouch..! Tuto inévitable à ce niveau.*

Précédemment, des colonnes ont été filtrées, mais awk est aussi principalement utilisé pour filtrer des lignes grâce aux syntaxes des expressions régulières.

Retrouver les lignes qui contiennent CAMILLE :

```
awk '/CAMILLE/ { print $1, $3, $2 }' file.txt
```

retour de la commande

```
CAMILLE 7 M
```

Nota :
L'ordre des colonnes a été modifié pour l'exemple.

Un autre filtre plus complexe, rechercher les lignes qui commencent par C et qui contiennent la lettre A ou la lettre O :

```
awk '/^C.*[AO]/ { print $1, $3, $2 }' file.txt
```

retour de la commande

```
CAMILLE 7 M
CHLOE 12 F
CLARA 11 F
```

awk est également très utile et puissant pour gérer des filtres sur des paragraphes.
Pour récupérer les lignes de CL à E, tapez :

```
awk '/^CL/,/^E/ { print $0 }' file.txt
```

retour de la commande

```
CLARA F 11
CLEMENT M 7
EMMA F 6
```

Variables awk

awk fournit des variables utiles qui peuvent être utilisées, affichées, calculées ou assignées.

Variable NR NF

- NR : nombre d'enregistrements (numéro de ligne).
- NF : nombre de champs (nombre de colonnes).

```
awk '{ print NR, NF, $0 }' file.txt
```

retour de la commande

```
1 3 Nom         Genre           Age
2 1 ----------------------------------------
3 3 CAMILLE             M               7
4 3 CHLOE               F               12
5 3 CLARA               F               11
6 3 CLEMENT             M               7
7 3 EMMA                F               6
```

```
8 3 THEO                    M                   8
```

Variable FS OFS

- FS : Séparateur de champ (par défaut : espace/tabulation).
- OFS : Séparateur de champ en sortie (par défaut : espace).

```
awk '/CAMILLE/ { OFS="," ; print $2,$1 }' file.txt
```

retour de la commande

```
M,CAMILLE
```

À noter le caractère “;” pour séparer les instructions dans la même ligne et la façon dont on assigne une valeur à une variable (OFS=“,”).

Scripts awk

awk a été utilisé précédemment en mode lignes de commande. Lorsque le programme awk devient complexe, ce dernier peut être stocké dans un fichier `prog.awk` comme ici :

prog.awk

```
/^CL/,/^E/ {
     print NR, $0
}
```

Puis interprété grâce à l'option -f :

```
awk -f prog.awk file.txt
```

retour de la commande

```
5 CLARA F 11
6 CLEMENT M 7
7 EMMA F 6
```

Pré et Post opérations

awk offre des sections pré-traitement (BEGIN) et post-traitement (END) lors de l'analyse d'un fichier.

La structure du script awk est :

prog.awk

```
/^CL/,/^E/
BEGIN {
        action
}

/filter/,/filter/ { action }

{ action}

END {
        action
}
```

Donne :

```
awk -f prog.awk file.txt
```

retour de la commande

```
CLARA           F               11
CLEMENT         M               7
EMMA            F               6
```

Les blocs BEGIN et END ne sont pas obligatoires. Il peut y avoir un bloc BEGIN sans bloc END, un bloc END sans bloc BEGIN, ou aucun de ces 2 blocs.

Des scripts bien plus complexes peuvent alors être écrits. Par exemple, extraire 2 colonnes en remplaçant les tabulations par des “;” et afficher le nombre de lignes à la fin :

prog.awk

```
BEGIN {
        FS=" "
        OFS=";"
}
{
        print $1, $3
}
END {
        printf "\nThe file has %d lines\n", NR
}
```

```
awk -f prog.awk file.txt
```

retour de la commande

```
Nom;Age
-------------------------------------;
CAMILLE;7
CHLOE;12
CLARA;11
CLEMENT;7
EMMA;6
THEO;8

The file has 8 lines
```

ATTENTION ! Vérifiez toujours l'écriture de chaque script donné ici avant de l'appliquer ! *Bis.* 

Fonctions

Le parseur awk offre beaucoup de fonctions internes très utiles pour traiter les données.
Consulter les manuels de l'utilitaire awk pour la liste complète des fonctions internes, en voici une liste partielle :

toupper tolower

Convertir du texte en majuscules ou en minuscules avec les fonctions toupper et tolower

```
awk '/THEO/ { print $1, tolower($1) }' file.txt
```

retour de la commande

```
THEO theo
```

int

Convertir une valeur en entier avec la fonction **int** :

```
awk '/CHLOE/ { print $3, int($3/5)}' file.txt
```

retour de la commande

```
12 2
```

printf

La fonction printf avec awk fonctionne comme la fonction printf en C afin de formater la sortie :

```
awk 'NR > 2 { printf "%10s %02d %-10s\n", $1,$3, $1}' file.txt
```

retour de la commande

```
   CAMILLE 07 CAMILLE
     CHLOE 12 CHLOE
     CLARA 11 CLARA
   CLEMENT 07 CLEMENT
      EMMA 06 EMMA
      THEO 08 THEO
```

length

Afficher la taille d'une chaîne de caractères avec la fonction length :

```
awk '/CLEM/ { print $1, length($1) }' file.txt
```

retour de la commande

```
CLEMENT 7
```

match

Retourne la position d'une chaîne de caractères remplissant les critères d'une expression régulière avec la fonction match :

```
awk 'NR >2 { print $1, match($1,"A")}' file.txt
```

retour de la commande

```
CAMILLE 2
CHLOE 0
CLARA 3
CLEMENT 0
EMMA 4
THEO 0
```

gsub

Remplacer des chaînes de caractères avec la fonction gsub :

```
awk 'NR >2 { gsub("A","_",$1) ; print $1 }' file.txt
```

retour de la commande

```
C_MILLE
CHLOE
CL_R_
CLEMENT
EMM_
THEO
```

substr

Extraire une portion de texte avec la fonction substr :

```
awk '{ print $1, substr($1,2,3) }' file.txt
```

retour de la commande

```
Nom om
---------------------------------------- ---
CAMILLE AMI
CHLOE HLO
CLARA LAR
CLEMENT LEM
EMMA MMA
THEO HEO
```

Fonctions utilisateur

La possibilité de créer des fonctions utilisateur est une des fonctionnalités les plus importantes de l'utilitaire awk. Les fonctions sont définies avec le mot clé **function**.

prog.awk

```
function gentag(nom,age) {
        tmp=tolower(substr(nom,1,3))
        return tmp "_" age
}
```

```
BEGIN {
        FS=" "
        OFS=";"
}

{
        print $1, $3, gentag($1,$3)
}

END {
print NR , "lines"
}
```

```
awk -f prog.awk file.txt
```

retour de la commande

```
Nom;Age;nom_Age
----------------------------------------;;---_
CAMILLE;7;cam_7
CHLOE;12;chl_12
CLARA;11;cla_11
CLEMENT;7;cle_7
EMMA;6;emm_6
THEO;8;the_8
8;lines
```

Programmation

Le parseur awk offre toutes les structures de programmation : conditions, boucles, itérations.

Condition

Les enfants sont ils en primaire ou au collège avec if() {} else {} ?

prog.awk

```
BEGIN {
        OFS=","
}
NR <=2 { next }
{
        if ( $3 < 11 ) {
                ecole="primaire"
        } else {
```

```
                ecole="college"
        }

        print $1, ecole
}
```

```
awk -f prog.awk file.txt
```

retour de la commande

```
CAMILLE,primaire
CHLOE,college
CLARA,college
CLEMENT,primaire
EMMA,primaire
THEO,primaire
```

Remarquer la façon dont l'entête est écartée : NR ⇐2 { next }

Boucles

Remplacer l'âge de l'enfant par un nombre de points avec while() {}.

prog.awk

```
NR <=2 { next }
{
        min=1
        printf "%-10s", $1
        while  ( min <= $3 ) {
                printf "."
                min++
        }
        printf "\n"
}
```

```
awk -f prog.awk file.txt
```

retour de la commande

```
CAMILLE   .......
CHLOE     ............
CLARA     ..........
CLEMENT   .......
```

```
EMMA      ......
THEO      ........
```

Itérations

Remplacer l'âge de l'enfant par un nombre de points avec for (i= ; i< ; i++) { }.

prog.awk

```
NR <=2 { next }
{
        printf "%-10s", $1
        for ( min=1 ; min <= $3; min++ ) {
                printf "."
        }
        printf "\n"
}
```

```
awk -f prog.awk file.txt
```

retour de la commande

```
CAMILLE   .......
CHLOE     ............
CLARA     ...........
CLEMENT   .......
EMMA      ......
THEO      ........
```

Tableaux (Arrays)

Pour terminer cette brève présentation : les tableaux avec awk, particulièrement pratiques pour calculer des agrégats.
La structure d'un tableau avec awk est très simple :

tab[indice] = value

Calculer la moyenne d'âge des enfants par sexe :

prog.awk

```
{
        if ( NR <= 2 ) { next } # skip first 2 lines
        tab_age[$2]+=$3
```

```
        tab_cpt[$2]++
}
END {
        for ( genre in tab_age ) {
        print genre, " : ", "Moy :",
int(tab_age[genre]/tab_cpt[genre]), "ans", "nb :", tab_cpt[genre]
        }
}
```

```
awk -f prog.awk file.txt
```

retour de la commande

```
F : Moy : 9 ans nb : 3
M : Moy : 7 ans nb : 3
```

Remarquer comment les 2 tableaux sont remplis et traités à la fin.

Lien

Un lieu formidable là :

- Outil Linux - **SQLPAC** SQL Pour Administrateurs & Concepteurs

1)

N'hésitez pas à y faire part de vos remarques, succès, améliorations ou échecs !

2)

Vous pouvez faire cela avec votre éditeur de texte favori, voire avec cat ou simplement en téléchargeant la liste.