La pratique d'awk par l'exemple: Introduction rapide

Résumé:

C'est une introduction au bon vieux programme AWK d'Unix. Elle montre comment écrire de petites séquences type AWK pour automatiser les tâches du travail quotidien.

Au départ, l'idée d'écrire ce texte m'est venue après avoir lu deux autres articles publiés dans LinuxFocus et écris par Guido Socher. Le premier, concerne find et les commandes associées, m'a montré que je n'étais pas le seul à utiliser la ligne de commande. Souvent,de belles interfaces graphiques ne vous disent pas comment les choses se passent réellement (c'est de cette manière que windows est arrivé il y a des années). L'autre article concerne les expressions régulières. Bien que les expressions régulières ne soient que légèrement abordées dans le présent article, il est nécessaire de les connaître pour tirer le maximum d'awk et d'autres commandes comme sed et grep.

A la question: "la commande awk est elle réellement utile?" nous répondons sans ambiguité oui! Awk peut être utile pour un utilisateur lambda afin de traiter des fichiers texte, les reformater etc... Pour un administateur système, awk est vraiment un utilitaire très important. Jetez donc un coup d'oeil à `/var/yp/Makefile` ou aux scripts d'initialisation. Awk est utilisé partout.

Introduction à awk

Mes premiers pas avec awk sont sufisement vieux pour avoir été oubliés. J'avais un collègue qui devais travailler avec de très grosse sorties issues d'un petit Cray. La page du manuel pour `awk` sur le cray était vraiment réduite au minimum, mais il décida quand même qu'awk ressemblait beaucoup à ce qu'il souhaitait même s'il ne savait pas encore l'utiliser.
Beaucoup plus tard, dans une situation banale de la vie courante, un autre collègue utilisa awk pour extraire la première colonne d'un tableau:

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

Simple n'est ce pas? Cette simple tâche ne nécessite pas de programmation complexe en C ou de n'importe quel autre langage interprété ou compilé. Une simple ligne d'awk suffit.

Une fois que nous savons extraire une colonne, nous pouvons faire des choses comme renommer des fichiers:

```
ls -1 pattern | awk '{print "mv "$1" "$1".new"}' | sh
```

... et il y en a d'autres. L'utilisation de `sed` ou de `grep` avec les exemples précédents, nous donne des outils encore plus puissants.

1. Modification d'une partie du nom
   ```
   ls -1 *old* | awk '{print "mv "$1" "$1}' | sed s/old/new/2 | sh
   ```
   (cela ne marchera pas dans certain cas comme: file_old_and_old)
2. supprimer seulement les fichiers (cela peut être fait avec rm seul, mais pas avec un alias comme 'rm -r')
   ```
   ls -l * | grep -v drwx | awk '{print "rm "$9}' | sh
   ```
   (Cela peut ne pas marcher avec des nom étranges ou des permissions)
   Attention si vous essayez ceci dans votre répertoire personnel, il y a destruction de fichiers!
3. supprimer seulement les répertoires
   ```
   ls -l | grep '^d' | awk '{print "rm -r "$9}' | sh
   ```
   (Je pense que cela fonctionne dans tous les cas et peut être amélioré en utilisant:
   ```
   ls -p | grep /$ | awk '{print "rm -r "$1}'
   ```
)
   Attention si vous essayez ceci dans votre répertoire personnel, il y a destruction de fichiers!

Comme vous pouvez le voir, awk est d'un grand secours pour répéter de nombreuses fois une opération... et en plus de cela, il est bien plus amusant d'écrire un programme awk que de répeter 20 fois la même chose manuellement.

Bien que nous utilisions le terme `commande-awk`, awk n'est pas le genre de chose qui est habituellement appelé une commande. `awk` est un langage de programmation, avec une syntaxe proche du C à beaucoup d'égards. C'est un langage interprété et l'interpréteur `awk` traduit des instructions.

Voici la syntaxe de la commande:

```
# gawk --help
Usage: gawk [POSIX or GNU style options] -f progfile [--] file ...
        gawk [POSIX or GNU style options] [--] 'program' file ...
POSIX options:          GNU long options:
        -f progfile             --file=progfile
        -F fs                   --field-separator=fs
```

```
        -v var=val              --assign=var=val
        -m[fr] val
        -W compat               --compat
        -W copyleft             --copyleft
        -W copyright            --copyright
        -W help                 --help
        -W lint                 --lint
        -W lint-old             --lint-old
        -W posix                --posix
        -W re-interval          --re-interval
        -W source=program-text  --source=program-text
        -W traditional          --traditional
        -W usage                --usage
        -W version              --version

Report bugs to bug-gnu-utils@prep.ai.mit.edu,
with a Cc: to arnold@gnu.ai.mit.edu
```

Au lieu d'encadrer la commande avec des guillemets simples ('), on peut les écrire dans un fichier et l'appeler avec l'option `-f`, ainsi qu'avec des variables définies en ligne de commande avec `-v var=`val, ce qui rends le programme plus versatile.

Awk est, en gros, un langage de traitement de tableau. Autrement dit, dédié à des informations qui peuvent être groupées en champs et enregistrements. L'avantage ici réside dans l'extrême flexibilité de ces deux définitions.

Awk est puissant. Il est conçu pour travailler avec des enregistrements arrangés en lignes; mais ceci peut être modifié. Afin de présenter plus en détails certains de ces aspects, nous allons etudier quelques exemples représentatifs tirés de cas réels.

- Imprimer des beaux tableaux
  Vous avez peut être eu a imprimer des tables ASCII de provenances diverses. Par exemple, vous deviez imprimer les noms d'hôtes, les adresses ethernet et les IP sous forme de liste. Quand de telles tables sont très grosses, elles sont difficilement lisibles et le besoin d'un outils comme LaTeX ou, au moins d'un meilleur format se fait sentir. Si la table est simple, (et/ou nous connaissons bien `awk`), ce n'est pas trop difficile:

```
BEGIN {                                 { printf $1" &
  printf "LaTeX preample"               "
  printf                                  printf $2" &
"\\begin{tabular}{|c|c|...|c|}"         "
  }                                       .
                                          .
                                          .
                                          printf $n"
                                        \\\\ "          END {
                                          printf          print
                                        "\\hline"       "\\end{document}"
                                          }               }
```

Ce n'est certainement pas un programme *générique* mais nous débutons

seulement...
(Les double backslashes (\) sont nécessaires parce que c'est le caractère d'échappemment de l'interpréteur)

- Tronçonner les fichiers de sortie
  SIMBAD est une base de données d'objets astronomique qui fournit, entre autres, les positions des étoiles dans le plan du ciel. Par le passé, j'ai eu à y effectuer des recherches pour dessiner des graphiques autour d'objets. L'interface permettait de sauver les résultats dans un fichier texte. J'ai eu deux approches: créer un fichier pour chaque objet, ou lui fournir toute la liste résultant dans un gros fichier de résultats. Comme j'ai décidé d'opter pour la seconde approche, j'ai utilisé awk pour découper le gros fichier de résultats. Evidemment j'ai eu besoin de profiter de certaines caractéristiques de la sortie.

1. Chaque requète produit une ligne d'entête comme suit:
   `====> name : nlines <====`
   Le premier entête nous permet de repérer le début d'un nouvel objet et le quatrième combien d'entrées contient l'objet (bien que cette donnée ne soit pas strictement nécessaire)
2. Le caractère utilisé dans la liste pour séparer les colonnes était '|'. Ceci nécessita deux lignes de code supplémentaires pour filtrer la liste et ne conserver que les champs qui m'interessaient.

```
( $1 == "====>" ) {
  NomObj = $2
  TotObj = $4
  if ( TotObj > 0 )
{
    FS = "|"
    for ( cont=0 ;
cont<TotObj ;
cont++ ) {
        getline
        print $2 $4
$5 $3  >> NomObj
        }
    FS = " "
    }
  }
```

NOTA: En fait, le nom de l'objet n'étatit pas donné et ce fut légèrement plus compliqué; mais ceci est supposé être un exemple didactique.

- Jouons avec le buffer du courrier

```
BEGIN {
  BEGIN_MSG  =
"From"
  BEGIN_BDY  =
"Precedence:"
  MAIN_KEY   =
"Subject:"
  VALIDATION =
"[MONTH REPORT]"

  HEAD = "NO"; BODY
= "NO"; PRINT="NO"
  OUT_FILE =
"Month_Reports"
  }

  {
```

Si nous sommes administrateur d'une liste de courrier et que de temps en temps, des messages spéciaux y sont envoyés (par exemple des rapports mensuels) avec un format spécifique (le sujet contient '[MONTH REPORT] mois , dept'). Nous pouvons décider soudainement à la fin de l'année de regrouper ces messages à part des autres.
Ceci peut être fait en traitant le buffer de courrier avec le programme awk sur la gauche.

Pour que chaque rapport soit écrit individuellement dans un fichier, nous aurons besoin de trois lignes de code supplémentaires.

NOTA: Cet exemple suppose que le tampon de courrier est structuré comme je l'attends. En fait, je ne connais pas le format réel, mais ce programme fonctionne avec mon installation (ici aussi, dans certains cas étranges, il pourrait ne pas marcher).

```
  if ( $1 ==
BEGIN_MSG ) {
    HEAD = "YES";
BODY = "NO";
PRINT="NO"
    }

  if ( $1 ==
MAIN_KEY ) {
    if ( $2 ==
VALIDATION ) {
      PRINT = "YES"
      $1 = ""; $2 =
""
      print
"\n\n"$0"\n" >
OUT_FILE
      }
    }

  if ( $1 ==
BEGIN_BDY ) {
    getline
    if ( $0 == "" )
{
      HEAD = "NO";
BODY = "YES"
    } else {
      HEAD = "NO";
BODY = "NO";
PRINT="NO"
      }
    }

  if ( BODY ==
"YES" && PRINT ==
"YES" ) {
    print $0 >>
OUT_FILE
    }
  }
```

Des programmes comme ceux-ci ne prennent que 5 mn à élaborer et 5 de plus à écrire (ou bien plus de 20 mn sans réfléchir en utilisant la méthode plus drôle des essais et erreurs succesifs).

J'ai utilisé awk pour beaucoup d'autres tâches (génération automatique d'informations four des bases de données simples) et j'en sais suffisemment sur le sujet pour être certain que l'on peut en faire beaucoup de choses.
Laisser juste votre imagination faire le reste.

Un problème

Un des problèmes vient du fait qu'awk à besoin d'informations parfaitement tabulées,

(et une solution)

J'ai finalement réalisé que la colonne que je cherchait à trier était la dernière et que awk

sans trous. Awk ne marche pas, par exemple, avec des colonnes de largeurs fixes. Ce n'est pas génant si nous créons nous même les données d'entrée: on choisit quelque chose d'inhabituel pour séparer les colonnes, et plus tard on s'en sert avec `FS` et c'est tout!!! Si l'on dispose déjà des données cela peut être un peu plus compliqué parce que le séparateur de champs (`FS`) peut aussi être un espace, par conséquent certain noms composés peuvent poser un problème. Ainsi, un tableau comme celui-là:

```
1234  HD 13324  22:40:54 ....
1235  HD12223   22:43:12 ....
```

est difficile à manipuler avec awk. Des données comme celles là sont parfois obligatoires. De plus, cela se retrouve assez souvent car les données ne sont jamais homogènes. Si l'on n'a qu'une colonne avec de telles carctéristiques, on peut résoudre le problème (si quelqu'un sais comment traiter plus d'une colonne de manière générique, merci de m'avertir!).
Une fois, j'ai eu à traiter une table de cette espèce. La seconde colonne était un nom et comprenait un nombre variable d'espaces. Comme cela arrive souvent, j'avais à le trier sur la troisième colonne. J'ai essayé plusieurs fois avec `sort +/-n.m` qui a rencontré les mêmes problèmes d'espaces intégrés.

sait combien il y a de champs dans l'enregistrement en cours. Cela était suffisant pour pouvoir accèder à la dernière colonne (parfois `$9`, et parfois `$11`, mais toujours `NF`). A la fin de la journée, le résultat souhaité était obtenu:

```
{
  printf $NF
  $NF = ""
  printf " "$0"\n"
}
```

Ceci fournit un résultat égal à l'entrée mais avec la dernière colonne en première position. Bien sûr, cette méthode est applicable facilement depuis le troisième champ depuis la fin, ou depuis un champ de contrôle qui contient toujours la même valeur.
Avec le dernier champ en première position, on peut trier (`sort`) les données sans problèmes.
Il suffit d'utiliser ses idées et son imagination.

AWK en profondeur

Travailler sur des lignes ciblées

Jusqu'ici, tous les exemples cités traitent toutes les lignes du fichier d'entrée. Mais, comme spécifié dans la page du manuel, il est possible de ne traiter que certaines de ces lignes. Il suffit juste de faire précéder le groupe de commandes avec la condition que doit remplir la ligne. Cette condition peut aller d'une simple expression régulière pour vérifier le contenu d'un champ à la possibilité de grouper les conditions avec des opérateurs logiques adaptés.

Awk en tant que langage de programmation

Comme tous les autres langages de programmation, `awk` propose toutes les structures de contrôle nécessaires ainsi qu'un jeux d'opérateurs et de fonctions prédéfinies pour manipuler les nombres et les chaînes. La syntaxe est proche du C.

Et bien sûr, il est possible d'inclure des fonctions définies par l'utilisateur avec le mot clef **function**, en écrivant les commandes associées comme si l'on traitait une ligne du fichier d'entrée. En plus des variables scalaires habituelles, awk peut aussi manipuler des tableaux de taille variable.

Inclure des bibliothèques

Comme dans tous langages de programmation, il existe des fonctions très fréquemment utilisées et qu'il est incomfortable de couper-coller depuis d'autres lignes de code source. C'est la raison pour laquelle les bibliothèques existent. Avec la version GNU d'`awk`, il est possible de les inclure avec le programme `awk`. Ceci fait par contre partie des sujets qui dépassent les objectifs de cet article.

Conclusions

Awk n'est certainement pas aussi puissant que d'autres outils conçus dans des buts similaires, mais il a le grand avantage d'avoir la possibilité d'écrire de petits programmes adaptés à nos besoins, en très peu de temps.

AWK est très bien adapté pour les raisons qui ont conduit à sa construction: Lire des lignes et agir en fonction de leur contenu.

Des fichiers comme `/etc/password` s'avèrent idéaux pour être traités et reformatté par AWK. Ce dernier est inestimable pour de telles tâches.

Bien sûr,AWK n'est pas seul. Perl est un compétiteur très valable mais qui n'empêche pas de connaître quelques trucs AWK.

Informations complémentaires

Ces commandes de base ne sont pas forcément bien documentées, vous devriez toutefois trouver votre bonheur en cherchant un peu.

- La syntaxe n'est pas identique d'un *nix à l'autre, mais il existe une méthode pour connaître celui de notre système:
  `man awk`

- O'Reilly a publié un livre: *Sed & Awk (Nutshell handbook)* par Dale Dougherty.
- En recherchant chez Amazon, on trouve d'autres titres comme *Effective Awk Programming: A User's Guide*, orientés vers gawk, et une demi douzaine d'autres titres.

D'habitude, tous les livres unix mentionnent cette commande mais seuls quelques uns donnent des détails utils. Le mieux que nous puissions faire est de parcourir tous les livres qui nous passent entre les mains car on ne sait jamais d'avance où l'information utile peut être trouvée.