Le nom de la commande AWK provient de la première lettre du nom de ses trois créateurs : Alfred Aho, Peter Weinberger et Brian Kernighan. AWK est presque un langage à lui seul, c'est un outil puissant permettant de faire des opérations bien plus complexes que GREP et SED ne peuvent le faire. On peut faire des opérations d'arithmétique, de logique, de filtrage, de substitution, d'édition de tableaux... D'ailleurs, nous utiliserons principalement AWK comme un éditeur de tableau. (sachant que la base de donnée Lexique3.txt est un tableau).

Le principe de fonctionnement de la commande AWK est le même que pour GREP et SED , dans le sens ou ces commandes peuvent soit recevoir et traiter un flux de données issu d'une commande préalablement effectuée (cat fichier.txt | awk), ou soit éditer directement un fichier (awk fichier.txt).

I/ Éditer un tableau avec AWK

On peut facilement lire, écrire ou modifier un tableau avec AWK. Une colonne d'un tableau est représentée par le caractère dollar ($) suivi de son numéro de référence. Par exemple, la première colonne sera accessible par $1, la deuxième par $2, la troisième par $3, etc...
Il faudra aussi préciser le caractère de séparation de champs présent entre chaque colonne avec -F. Le caractère de séparation de champs pourra aussi être précisé par FS lorsqu'on le fait dans une instruction. Il en est de même pour le caractère de séparation de ligne, habituellement n, qui sera défini par RS, mais nous verrons ça plus tard.

Exemple :

awk -F"t" '{print $1,$4}' Lexique3.txt

Dans le fichier Lexique3.txt (téléchargeable sur lexique.org), le caractère de séparation employé est tabulation (t). L'action {print $1,$4} va envoyer à la sortie de la commande AWK la première et la quatrième colonne ($1,$4) qui correspondent respectivement au mot et à sa catégorie grammaticale.

Il est cependant nécessaire de redéfinir le caractère de séparation qui à la sortie de l'éditeur awk est par défaut un espace. Nous devrons donc ajouter par exemple après l'action print une tabulation entre chaque colonne à l'aide de t :

awk -F"t" '{print $1,"t"$4}' Lexique3.txt

Nous retiendrons de ces deux exemples la manipulation des colonnes d'un tableau avec le caractère dollar suivi de son numéro ($1,$2,$3,etc...) et, le renseignement des caractères de séparation de champs pour la lecture d'un tableau (-F"t") et avec l'ajout du caractère à la sortie de la commande awk (print $1,"t"$4).

Notes :

Pour des raisons de confort de lecture des résultat, vous pouvez rediriger le résultat de chacune des commandes présentées ici dans l'éditeur LESS via un pipe (|) :

awk -F"t" '{print $1"t",$4}' Lexique3.txt | less

Pour arrêter le défilement d'un flux ou pour sortir de l'éditeur LESS, faites : CTRL Z

II/ Les Motifs avec AWK

Les motifs acceptés par AWK peuvent être une expression régulière, une relation, un test de logique ou une combinaison de plusieurs expressions régulières.

AWK comme un filtre GREP avec une expression régulière

commande :

awk '/voiture/ {print}' Lexique3.txt

Cette commande va envoyer sur la sortie standard toutes les lignes dans lesquelles se trouve l'expression régulière « voiture ». Pour avoir un résultat un peu plus confortable, on peut demander à ce que la commande awk ne retourne à sa sortie que la première colonne de la base Lexique3.txt :

awk -F"t" '/voiture/ {print $1}' Lexique3.txt

résultat :

voituraient

voiturait
voiture
voiture
voiture-balai
voiture-bar
voiture-lit
voiture-restaurant
voiturer
voitureront
voitures
voitures
voitures-balais
voiturette
voiturettes
voitur

Remarques :

Regardez attentivement la liste du résultat et comparez-la à l'expression régulière demandée ci-dessus.
Nous demandions à l'aide d'une expression régulière la chaîne de caractères « voiture » et nous avons eu dans notre résultat la chaîne de caractères « voituraient » qui ne correspond pas à l'expression exacte « voiture ». Pour comprendre pourquoi nous avons eu ce résultat si surprenant, il est nécessaire de connaître un peu l'organisation de la base Lexique3 et le fonctionnement de tous filtres Unix qui se respectent :

- Premièrement, dans la base Lexique3 se trouve une colonne intitulée lemme (la colonne numéro 3 accessible via awk '{print $3}').

- Deuxièmement, les filtres Unix, qu'il s'agisse de GREP ou de AWK, traitent un fichier ligne par ligne sur toute la ligne. Autrement dit, les filtres Unix cherchent l'expression régulière sur toute la ligne d'un fichier. Cela à pour conséquence que si l'expression régulière demandée est présente quelque part sur la ligne, comme par exemple dans la colonne lemme, la ligne complète sera filtrée et envoyée à la sortie. Voyons ce qui se passe si nous demandons à AWK de nous donner à sa sortie la colonne 3 correspondant au lemme de l'expression régulière recherchée :

awk -F"t" '/voiture/ {print $1"t",$3}' Lexique3.txt

voituraient voiturer
voiturait voiturer
voiture voiture
voiture voiturer
voiture-balai voiture-balai
voiture-bar voiture-bar
voiture-lit voiture-lit
voiture-restaurant voiture-restaurant
voiturer voiturer
voitureront voiturer
voitures voiture
voitures voiturer
voitures-balais voiture-balai
voiturette voiturette
voiturettes voiturette
voituré voiturer

On pourra dire ce que l'on veut du lemme « voiturer », en attendant il explique la présence de l'expression « voituraient » dont le lemme est « voiturer » qui contient la sous chaîne de caractères « voiture » dans notre résultat. (voyez ci-après comment pallier ce problème)

Les tests de relations de champs avec AWK

Notre problème précédant me permet d'introduire facilement les tests de relations avec AWK, dont le but est de vérifier si un champs particulier (une colonne) contient une expression régulière. Nous allons, avec les tests de relations, pouvoir par exemple travailler uniquement dans la première colonne. Cela signifie que si nous cherchons l'expression régulière « voiture » seulement dans la première colonne, nous n'aurons que les mots contenant la chaîne « voiture » et rien d'autre :

awk -F"t" '$1~/voiture/ {print $1}' Lexique3.txt

Résultat :

voiture
voiture
voiture-balai
voiture-bar
voiture-lit
voiture-restaurant
voiturer
voitureront
voitures
voitures
voitures-balais
voiturette
voiturettes

Explications :

La commande ci-dessus considère que le caractère de séparation est une tabulation (-F"t"), puis elle sélectionne la colonne 1 ($1~), ensuite elle cherche l'expression « voiture » dans la colonne sélectionnée (/voiture/) et enfin elle affiche la première colonnes des lignes sélectionnées (print $1).
Dans notre exemple, si nous souhaitions rechercher l'expression exacte « voiture » sans retenir celles qui la contiennent, nous devrions alors la préfixer du caractère symbolisant le début de ligne (^) et la suffixer avec le caractère symbolisant la fin de ligne ($) (on utilise le caractère de fin de ligne ($) car nous avons préalablement sélectionné la colonne ($1~) s'affichant désormais sur une ligne) :

awk -F"t" '$1~/^voiture$/ {print $1}' Lexique3.txt

résultat :

voiture
voiture

Les combinaisons logiques et les opérateurs booléens

Le ET logique : &&

Le ET logique avec AWK symbolisé par un double esperluette (&&) est très utile car il permet de vérifier par exemple deux relations de champs.
Vous avez très probablement remarqué que dans la langue française, il existe un certain

nombre de mots partageant le même orthographe alors qu'ils appartiennent à différentes catégories grammaticales. Je sais que vous savez qu'il s'agit des homographes, mais je préfère m'adresser à vous comme un programmeur. Au passage, je peux vous dire que l'application systématique d'une définition linguistique n'est pas toujours si évidente que ça !
Pour en revenir au homographes, par exemple le mot « plus » écrit de cette manière peut être soit un adverbe ou soit le participe passé du verbe « plaire ».

Le ET logique nous donne la possibilité de choisir l'expression régulière ayant l'orthographe « plus » ET appartenant à la catégorie grammaticale souhaitée (adverbe ou verbe) :

awk '$1~/plus/ && $4~/ADV/ {print $1,$3,$4}' Lexique3.txt

résultat :

plus plus ADV

awk '$1~/plus/ && $4~/VER/ {print $1,$3,$4}' Lexique3.txt

résultat :

plus plaire VER
plus plus VER
plusse plaire VER

Dans cet exemple, nous avons travaillé sur la première colonne de Lexique3.txt correspondant à l'orthographe du mot ET sur la quatrième colonne correspondant à sa catégorie grammaticale. Je me suis permis d'afficher (avec print) entre la colonne $1 et $4, la colonne $3 (correspondant au lemme du mot) afin de rendre le résultat un peu plus compréhensible.

Le OU logique : ||

Symbolisé par deux pipes (||) le OU logique est généralement un peu moins pertinent que le ET logique :

awk '$4~/ADJ/ || $4~/ADV/ {print $1,$3,$4}' Lexique3.txt | less

Cette commande retournera tous les adjectifs ainsi que tous les adverbes de la base de donnée Lexique3.txt.

Le NON : !

Symbolisé par un point d'exclamation, le NON appelé aussi « contraire de » ou « différent de » est bien plus subtile qu'il n'y paraît. En effet cet opérateur nous permettrait par exemple de récupérer tous les mots sauf ceux qui contiennent la lettre e (comme la si bien fait Georges Perec) :

awk '!($1~/e/) {print $1,$3,$4}' Lexique3.txt | less

Attention, pensez à utiliser des parenthèses avec le NON, et remarquez aussi que la commande ci-dessus ne tient pas compte ni des E majuscules ni des E accentués, c'était juste pour l'exemple.

Pour finir, vous pouvez aisément assembler des combinaisons d'opérateurs logiques et définir des priorités d'exécutions à l'aide des parenthèses :

awk '!($1~/e/) && !($1~/a/) {print $1}' Lexique3.txt | less

awk '($4~/ADJ/ || $4~/ADV/) && $1~/plus/ {print $1,$3,$4}' Lexique3.txt | less

III/ Les Actions avec AWK

Nous avons vu précédemment l'usage des motifs avec la commande AWK. Voyons maintenant ce que nous pouvons faire de ces motifs en nous servant d'actions. Il existe de nombreuses actions et depuis le début de cette introduction et j'en ai utilisé une que vous reconnaitrez :

Fonctions traitants de chaînes de caractères :

Les fonctions traitant de chaînes de caractères sont absolument vitales pour le fonctionnement d'un script AWK d'autant plus s'il est orienté dans une manipulation à des fins linguistiques.

L'action print :

L'action print est employée pour afficher le résultat d'une commande AWK sur la sortie standard à savoir l'écran de votre terminal.

awk -Ft '{print $1,$2,$3}' Lexique3.txt

Cette commande va afficher la première, la deuxième et la troisième colonne de toutes les lignes du fichier Lexique3.txt.

L'action printf :

L'action printf est une autre manière d'afficher le résultat d'une commande qui sera en général peu utile pour nous... Je préfère aborder l'utilité de cette action plus tard.

L'action length :

Cette action calcule le nombre de caractères constituant une chaîne, une ligne ou une colonne :

awk -F"t" '{print $1,length($1)}' Lexique3.txt | less

Ce script calcule le nombre de caractères constituant chaque mot présent dans la première colonne du fichier Lexique3.txt. Puis, il écrit le mot suivit du nombre de caractères le constituant.

Résultat :

a l'instar 10
a posteriori 12
abaca 5
abaissa 7

L'action gsub :

Cette action nous donne l'occasion de substituer une chaîne de caractère par une autre exactement comme le fait l'éditeur SED :

echo "bonjour" | awk '{gsub("bonjour","hello"); print}'

La commande ci-dessus remplace la chaîne de caractères « bonjour » (initié par echo) par « hello ».

L'action index :

L'action index retourne la position la plus à gauche d'une sous chaîne dans une chaîne de caractères. Dans l'exemple qui suit, nous recherchons la valeur numérique de la position la plus à gauche de la lettre « l » dans le mot « ville » :

echo "ville" | awk '{print index($1,"l")}'

résultat :

3

Remarques :

Dans cet exemple, « $1 » représente la première colonne du résultat de la commande echo (qui se fait toujours sur une colonne, mais il fallait tout de même le préciser).
Rappelons-nous que lorsque nous cherchons un mot dans un index (un dictionnaire), nous commençons par lire la première lettre la plus à gauche, puis la deuxième, puis la troisième, etc...

L'action substr :

L'action substr retourne à partir de n caractères en partant de la gauche une sous chaîne de m caractères : substr (chaîne,n,m) :

echo "refroidissement" | awk '{print substr($1,3,5)}'

Résultat :

froid

Fonctions traitants de nombres :

Voici la liste des fonctions mathématiques disponible avec AWK :

cos (cosinus), sin (sinus), exp (exponentiel), log (logarithme), sqrt (racine carré), rand (randomisation 0 1), srand (ré-initialisation de rand), int (valeur entière).

Exemple :

echo "25" | awk '{print cos($1)}'

résultat :

0,991203

Retour sur les motifs : BEGIN et END :

Ces deux sélections particulières ou « patterns » donnent la possibilité d'exécuter des instructions AVANT et APRÈS un programme AWK. Le sélecteur BEGIN est plus souvent employé pour signaler un traitement à venir ou pour initialiser les variables d'un programme :

awk 'BEGIN {print "calcul de la racine carré de 25 :"} {print sqrt(25)}'
awk 'BEGIN {X=25} {print "calcul de la racine carré de " X} {print sqrt(X)}'

IV/ Les variables :

Les commandes AWK étant interprétées - et comme le plus souvent dans ce cas de figure - il n'est pas nécessaire de déclarer les variables avant de les initialiser. On peut directement initialiser la variable. D'autre part l'appel d'une variable dans une commande AWK se fait sans

le caractère $ (contrairement au Shell). Si on souhaite initialiser une variable avec une valeur non nul, on utilisera l'instruction BEGIN.

awk 'BEGIN {var="retourner"} {print "nombre de lettres dans" var} {print length(var)}'

Variables prédéfinies :

Variable Description
ARGC Nombre d'arguments de la ligne de commande
ARGIND Index du tableau ARGV du fichier courant
ARGV Tableau des arguments de la ligne de commande
CONVFMT Format de conversion pour les nombres
ENVIRON Tableau contenant les valeurs de l'environnement courant
ERRNO Contient une chaine décrivant une erreur
FIELIWIDTHS Variable expérimentale à éviter d'utiliser
FILENAME Nom du fichier d'entrée
FNR Numéro d'enregistrement dans le fichier courant
FS Contrôle le séparateur des champs d'entrée
IGNORECASE Contrôle les expressions régulières et les opérations sur les chaînes de caractères
NF Nombre de champs dans l'enregistrement courant
NR Nombre d'enregistrements lus
OFMT Format de sortie des nombres
OFS Séparateur des champs de sortie
ORS Séparateur des enregistrements de sortie
RLENGTH Longueur de la chaîne sélectionnée par le critère " n"
RS Contrôle le séparateur des enregistrements d'entrée " n"
RSTART Début de la chaîne sélectionnée par le critère
SUBSEP Séparateur d'indiçage

V/ Conclusion :

Dans cette petite introduction à l'éditeur AWK, je ne vous ai présenté qu'une infime partie de ce que l'on peut faire avec cet outil. J'ai agis de la sorte pour des raisons pédagogiques. Vous devriez cependant êtres capables d'une part de comprendre des scripts écrits en AWK de niveau intermédiaire et d'autre part d'écrire des petits scripts grâce à cette introduction. Je vous conseille de faire appel à plusieurs sources d'informations et de vous entrainer à rédiger des

scripts même s'ils n'ont que peut d'intérêts au début. C'est vrai qu'il existe de nombreuses solutions pour exploiter des données, mais avec AWK vous aurez un certain sentiment d'autonomie et de liberté.

Pou finir, vous habituer à la syntaxe d'AWK est une très bonne préparation à la programmation en langage C.

Bonne programmation !

SCHMITT Vivien http://www.lingunix.org – http://www.tsgeri.net