Fonctionnement général

awk lit sur l'entrée (fichier ou saisie clavier) ligne par ligne, puis sélectionne les lignes à traiter à l'aide d'expressions rationnelles. Chaque ligne sélectionnée est découpée en champs selon le séparateur d'entrée désigné par le symbole FS (caractère espace ou tabulation). Puis les différents champs sont mis à disposition dans des variables : $1 (1er champ), $2 (2ème champ), $3 (3ème champ), …, $NF (dernier champ).

Options et paramètres

-F *fs* *ou* **--field-sepearator *fs***	Utilise *fs* comme séparateur de champs.
-v *var*=*val* *ou* **--assign *var*=*val***	Assigne la valeur *val* à *var*.
-f *program-file* *ou* **--file *program-file***	Charge et exécute *program-file*.
-mf *NNN* *ou* **-mr *NNN***	*mr* pour *maximum number of records* (nbr max d'enregistrements) et *mf* pour *maximum number of fields* (nbr max de champs).
-W *compat* *ou* **-W *traditional*** *ou* **--compat--traditional**	Permet une compatibilité entre les différentes variantes d'AWK (awk, nawk, gawk).
-W dump-variables[=fichier] *ou* **--dump-variables[=fichier]**	Affiche un récapitulatif trié des variables globales, leurs types et leurs valeurs.
-W help *ou* **-W usage** *ou* **--help** *ou* **--usage**	Affiche l'aide.
-W version *ou* **--version**	Affiche la version.
--	Marque la fin des options.

Entrées/sorties

close(*fichier* [, *méthode*])	Ferme un fichier, un *pipe* ou un traitement déporté. Le paramètre optionnel *méthode* n'est à utiliser que lors de la fermeture d'un *pipe* bidirectionnel vers un traitement déporté (valeurs possibles : "to" ou "from").
getline	Charge la dernière saisie dans $0 ; NF, NR et FNR sont également initialisés. Retourne 0 une fois atteinte la fin du fichier (EOF) ou 1 en cas d'erreur. Si erreur il y a, ERRNO contient une description du problème.
getline <*fichier*	Renseigne $0 avec le contenu de l'enregistrement (*record*) suivant ; initialise NF.
getline *var*	Renseigne *var* avec le contenu de la saisie (*input record*) suivante ; initialise NF ainsi que FNR.
getline *var* <*fichier*	Renseigne *var* avec le contenu de l'enregistrement (*record*) suivant issu de *fichier*.
commande **| getline [*var*]**	Exécute la *commande* et envoie la sortie soit vers $0, soit vers *var*.
commande **|& getline [*var*]**	Exécute *commande* en qualité de traitement déporté.
next	Interrompt le traitement de l'enregistrement (*record*) courant, puis passe au suivant.
nextfile	Interrompt le traitement du fichier en cours, puis passe au suivant. FILENAME et ARGIND sont mis à jour, FNR repasse à 1.

Entrées/sorties (cont)

print	Affiche l'enregistrement (*record*) en cours. La chaîne se termine par le contenu de la variable OFS.
print *expressions*	Affiche les éléments passés en paramètre, tous séparés par la valeur définie dans OFS. La chaîne de sortie se termine par la valeur contenue dans ORS.
print *expressions* > *fichier*	Insère dans *fichier* les éléments passés en paramètre, tous séparés par la valeur définie dans OFS. La chaîne de sortie se termine par la valeur contenue dans ORS.
printf *format*, *expressions*	Affichage formaté, à la manière du *printf* du C.
printf *format*, *expressions* > *fichier*	Même chose qu'au-dessus, mais dans un fichier.
system(*ligne de commande*)	La *commande* est exécutée et son code de sortie récupéré.
fflush([*fichier*])	Purge les caches associés avec le *fichier*. Si ce dernier n'existe pas, c'est *stdout* qui se trouve purgé. Si le paramètre *fichier* est à *null*, tous les caches de fichiers et de *pipes* sont alors purgés.
print ... >> *fichier*	Ajout de données à la fin d'un *fichier*.
print ... | *commande*	Envoie la sortie d'une *commande* dans un *pipe*.
print ... |& *commande*	Envoie des données à un traitement déporté.

Actions pré-traitement et post-traitement

```
BEGIN { Actions}
{ACTION} # Actions qui seront exécutées sur chaque ligne
END { Actions }
# les lignes de commentaires commencent par un dièse.
```

Les actions suivant la balise BEGIN sont exécutées avant que le traitement des données fournies en entrée ne commence, tandis que les actions derrière END sont exécutées une fois que toutes les lignes ont été traitées.

Fonctions relatives au temps

systime()	Retourne le nombre de secondes écoulées depuis l'Epoch (1970-01-01 00:00:00 UTC).
mktime(*datespec*)	Transforme *datespec* (de la forme YYYY MM DD HH MM SS[DST]) en nombre de secondes écoulées depuis l'Epoch (1970-01-01 00:00:00 UTC).
strftime([*format* [, timestamp]])	Formate *timestamp* sur la base de ce qui est fourni dans *format*. *timestamp* est un nombre de secondes écoulées depuis l'Epoch tel que retourné par *systime()*. Si *timestamp* n'est pas renseigné, l'heure courante est utilisée.

Variables

FS	Séparateur de champs en entrée.
OFS	Séparateur de champs en sortie.
RS	Séparateur de lignes en entrée.
ORS	Séparateur de lignes en sortie.
NR	Nombre de lignes.
NF	Nombre de champs.
FILENAME	Nom du fichier passé en paramètre.
FNR	Nombre de lignes dans le fichier courant.
$0	Toute la ligne.
$1, $2...$n	Champs de 1 à *n*.
ARGC	Nombre de paramètres sur la ligne de commande.
ARGV	Tableau contenant les paramètres passés sur la ligne de commandes. Indexé de 0 à ARGC - 1.
ARGIND	Indique quel fichier listé dans ARGV est en cours de traitement.

Notions-clef

- *awk* voit le texte sous la forme de lignes et de champs,
- *awk* supporte l'usage de conditions, de boucles et de variables,
- *awk* peut traiter des chaînes de caractères et des chiffres,
- *awk* peut générer des rapports formatés.

Opérateurs

&& || !	Opérateurs logiques (ET, OU, NON).
< <= == != >= > ~ !~	Opérateurs de comparaison.

Mathématiques

atan2(*y*, *x*)	Retourne l'arc tangente de y/x en radians.
cos(*x*)	Retourne le cosinus de *x*, en radians.
exp(*z*)	Retourne l'exponentielle de *z*.
int(*f*)	Retourne la part entière de *f*.
log(*e*)	Fonction logarithmique.
rand()	Retourne un nombre réel aléatoire N compris entre 0 et 1 où 0 <= N < 1.
sin(*x*)	Retourne la sinusoïdale de *x*, en radians.
sqrt(*x*)	Retourne la racine carrée de *x*.
srand([*expression*])	Génère un nombre aléatoire sur la base d'*expression*. Si *expression* n'est pas renseigné, la date est utilisée.

Traitements sur les chaînes

asort(*s* [, *d*])	Retourne le nombre d'éléments présents dans le tableau *s*. Le contenu de *s* est trié, puis réindexé en partant de 1. Si le tableau destination optionnel *d* est spécifié, alors *s* est dupliqué dans *d*, puis le traitement s'effectue sur *d*, laissant *s* inchangé.

Traitements sur les chaînes (cont)	
asorti(*s* [, *d*])	Même chose que *asort()* à ceci près que ce sont les indices du tableau et non les valeurs qu'il contient qui sont utilisées pour le tri.
gensub(*r*, *s*, *h* [, *t*])	Cherche les occurrences de la regex *r* dans *t*. Si *h* est une chaîne commençant par g ou G, alors toutes les occurrences de *r* sont remplacées par *s*. Sinon, *h* est necessairement un chiffre indiquant quelle occurrence de *r* remplacer. Si *t* n'est pas fourni, $0 est alors utilisé. Cette fonction, au contraire de *sub()* et *gsub()*, retourne la chaîne modifiée en résultat, laissant la chaîne d'origine inchangée.
gsub(*r*, *s* [, *t*])	Remplace toutes les occurrences de la regex *r* au sein de la chaîne *t* par *s*, puis retourne le nombre de changements opérés. Si *t* n'est pas renseigné, $0 est utilisé.
index(*s*, *t*)	Retourne la position de *t* dans *s* ou 0 si *t* n'est pas trouvé.
length([*s*])	Retourne la taille de la chaîne *s*.
match(*s*, *r* [, *a*])	Retourne la position de la regex *r* au sein de *s*. RSTART et RLENGTH sont initialisées.
split(*s*, *a* [, *r*])	Découpe la chaîne *s* sur la base de la regex *r* et stocke les sections résultantes dans le tableau *a*. Le nombre de champs (*fields*) est retourné. Si *r* n'est pas fourni, FS est utilisé. Notez que *a* est vidé avant le début du traitement.
sprintf(*format*, *expressions*)	Affichage formaté, à la manière de *printf* en C.
strtonum(*chaîne*)	Transforme une chaîne en nombre. Si la chaîne commence par 0, elle est traitée comme une valeur octale. Si elle commence par 0x ou 0X, elle est traitée comme une valeur hexadécimale.
sub(*r*, *s* [, *t*])	Remplace la 1ère occurrence de la regex *r* au sein de la chaîne *t* par *s*, puis retourne le nombre de changements opérés. Si *t* n'est pas renseigné, $0 est utilisé.
substr(*s*, *i* [, *n*])	Collecte le contenu de *s* à partir du point *i*. Si *n* (balise de fin) n'est pas fourni, alors on récupère le reste de *s*.
tolower(*chaîne*)	Transforme les majuscules en minuscules.
toupper(*chaîne*)	Transforme les minuscules en majuscules.

Manipulation d'éléments binaires	
and(*v1*, *v2*)	ET logique sur *v1* et *v2*.
compl(*vx*)	Retourne le complément bit-à-bit de *vx*.
lshift(*valeur*, *compteur*)	Décale *valeur* de *compteur* bits vers la gauche.
or(*v1*, *v2*)	OU logique sur *v1* et *v2*.
rshift(*valeur*, *compteur*)	Décale *valeur* de *compteur* bits vers la droite.
xor(*v1*, *v2*)	OU EXCLUSIF sur *v1* et *v2*.

By **TME520** (TME520)
cheatography.com/tme520/
sysadmin.tme520.net

Published 19th April, 2015.
Last updated 19th April, 2015.