Awk

https://connect.ed-diamond.com/gnu-linux-magazine/glmf-273/awk-compare-aux-outils-unix-et-python

2023-awk-dio.pdf

awk001.pdf

Awk comparé aux outils Unix et Python Magazine Marque GNU/Linux Magazine n° Numéro 273 Mois de parution janvier 2025 Auteurs Par Blondon Stéphane

Résumé

Vous connaissez probablement déjà les outils classiques installés sur toutes les distributions GNU/Linux, et Python, langage populaire aujourd'hui. Découvrez Awk, un (vieux) langage qui pourrait vous servir lorsque les premiers sont trop limités et qu'un script Python vous semble surdimensionné.

Un script shell permet de réaliser des traitements basiques sur des lignes de texte grâce à un ensemble d'outils Unix (cut, grep, sort, wc, tr…). Cependant, lorsque les traitements deviennent plus complexes, un script shell avoue ses limites : faire des calculs est laborieux, le code est généralement peu structuré, etc. Dans ce cas, un script dans un langage plus évolué (comme Python) sera plus adapté. Cet article présente une solution intermédiaire : Awk, un langage créé par Alfred Aho, Peter Weinberger et Brian Kernighan. La version initiale date de 1977, révisée dans les années 80. Bien que toujours disponible, son heure de gloire semble passée. Il est spécialisé dans le traitement de lignes et leur transformation, ce qui le rapproche du comportement des enchaînements de filtres dans un terminal.

L'article présente Awk, en le comparant aux outils Unix classiques et à du code Python.

1. Disponibilité

En général, plusieurs interpréteurs Awk sont disponibles dans une distribution GNU/Linux (souvent gawk et mawk) et il est bien possible qu'il soit déjà installé. Tapez juste awk dans un terminal pour vérifier.

Par exemple, si mawk est installé, le début de la sortie sera :

ex001.txt

```
$ awk
Usage: mawk [Options] [Program] [file ...]

Program:
    The -f option value is the name of a file containing program text.
```

```
[...]
```

Dans le cas contraire, vous pouvez en installer un depuis les paquets de votre distribution. Ils sont nommés mawk ou gawk sous Debian et Arch Linux (mawk est dans les dépôts AUR). D'autres interpréteurs plus récents existent (nawk, goawk).

2. Utilisation

Awk a été conçu pour l'écriture de one-liners, mais il est aussi possible d'écrire un programme Awk pour l'exécuter.

L'usage en tant que one-liner est :

```
$ awk ‘le_programme’ les_fichiers_à_traiter
```

L'usage avec un programme Awk écrit dans un fichier est :

```
$ awk -f programme.awk les_fichiers_à_traiter
```

Dans le cas où le programme est écrit sur la ligne de commande, il faut l'entourer de guillemets simples, car le programme va utiliser le caractère dollar (par exemple, la variable $0). Si ce sont des guillemets doubles qui sont utilisés, le programme ne fonctionnera pas correctement, car le shell va tenter d'interpréter les variables préfixées par le dollar comme des variables du shell avant d'exécuter le programme.

Ceci est un comportement classique du shell et n’est donc pas une spécificité liée à Awk.

Pour la suite de l’article, nous allons utiliser un fichier nommé distribs.txt contenant les données suivantes :

```
Raspbian linux 2012
Archlinux linux 2002
Debian linux 1993
FreeBSD bsd 1993
pfSense bsd 2004
```

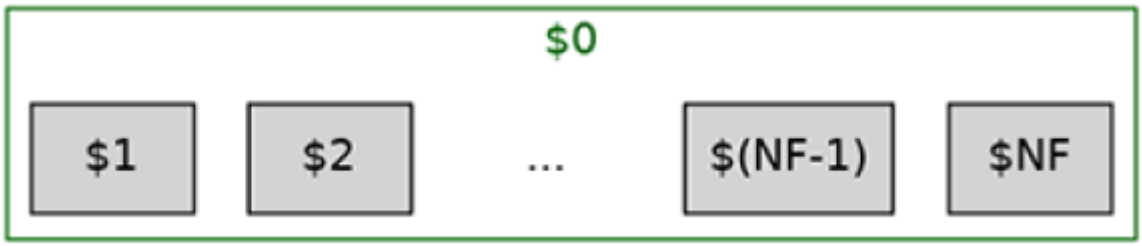
2.1 Imiter cat

Par exemple, pour faire l'équivalent de cat distribs.txt :

```
$ awk '{print $0}' distribs.txt
```

Comme le montre la figure 1, $0 représente l'enregistrement total (qu'on peut assimiler ici à la ligne). $1, le contenu du premier champ, $2 le contenu du deuxième champ, etc. Le contenu du dernier champ est $NF.NF signifie Number of Fields. On peut donc obtenir astucieusement le contenu de

l'avant-dernier champ avec $(NF-1).



awk compare fig 01-s Fig. 1 : Représentation des données accessibles d'un enregistrement.

La plupart des langages de programmation démarrent les listes à partir de 0 (comme C, Java, Lisp, etc.) ; Python fait aussi partie de cette catégorie. Quelques langages, comme Awk, commencent à partir de 1. $0 représentant l'enregistrement complet, le premier élément ne peut pas lui aussi commencer à 0. Parmi les autres langages commençant aussi à 1, R ou Lua sont probablement les plus connus. Julia permet de choisir n'importe quel entier, mais utilise 1 comme index par défaut.

2.2 Imiter cut

Afficher le premier et le troisième champ avec cut avec les paramètres longs :

```
$ cut --delimiter " " --fields 1,3 distribs.txt
```

Avec les paramètres courts :

```
$ cut -d " " -f 1,3 distribs.txt
```

Avec Awk :

```
$ awk '{print $1, $3}' distribs.txt
```

Alors que cut ne fait que filtrer des champs en gardant le même ordre, Awk permet de changer facilement l'ordre des champs ou d'insérer du texte dans la ligne qui sera affichée :

```
$ awk '{print $3, "est l'année de création de", $1}' distribs.txt
```

Doc awk

Présentation Présentation et syntaxe awk est une commande très puissante, c'est un langage de programmation a elle tout seule qui permet une recherche de chaînes et l'exécution d'actions sur les lignes sélectionnées. Elle est utile pour récuperer de l'information, générer des rapports, transformer des données entre autres.

Une grande partie de la syntaxe a été empruntée au langage c, d'ailleurs awk sont les abréviations de ces 3 créateurs dont k pour Kernighan, un des inventeurs du c.

La syntaxe de awk est la suivante:

awk [-F] [-v var=valeur] 'programme' fichier

ou

awk [-F] [-v var=valeur] -f fichier-config fichier

L'argument -F doit être suivi du séparateur de champ (-F: pour un “:” comme séparateur de champ).

L'argument -f suivi du nom du fichier de configuration de awk.

L'argument -v définit une variable (var dans l'exemple) qui sera utilisée par la suite dans le programme.

Un programme awk possède la structure suivante: critère de sélection d'une chaîne {action}, quand il n'y a pas de critère c'est que l'action s'applique à toutes les lignes du fichier.

Exemple:

awk -F“:” '{print $NF}' /etc/passwd

Il n'y a pas de critères, donc l'action s'applique à toutes les lignes du fichier /etc/passwd. L'action consiste à afficher le nombre de champ du fichier. NF est une variable prédéfinie d'awk, elle est égale au nombre de champs dans une ligne.

Généralement on utilisera awk en utilisant un script.

#!/bin/sh awk [-F] [-v var=valeur] 'programme' $1

Vous appelerez votre script mon-script.awk, lui donnerez des droits en exécution (755 par exemple), et l'appelerez ainsi:

mon-script.awk fichier-a-traiter

Dans la suite du cours, on utilisera awk en sous entendant que celui-ci est à insérer dans un script.

Le quote ' se trouve sur un clavier azerty standard avec le 4 et éventuellement l'accolade gauche.

ATTENTION: ils existent plusieurs “variétés” de awk, il se pourrait que certaines fonctions ou variables systèmes qui vous sont présentées dans ce cours n'existent pas sur votre UNIX. Faites en sorte si vos scripts awk doivent fonctionner sur des plates-formes différentes d'utiliser gawk sous licence GNU qui est totalement POSIX.

J'ai constaté des grosses différences de comportement entre le awk natif qu'on soit sous HP-UX, Solaris et sous LINUX, de même quand on insère la commande dans un script, on fait appel à un shell, suivant son type (bash shell, csh, ksh, ...), vous pouvez avoir quelques surprises. Enregistrements et champs awk scinde les données d'entrée en enregistrements et les enregistrements en champ. Un enregistrement est une chaîne d'entrée délimitée par un retour chariot, un champ est une chaîne délimitée par un espace dans un enregistrement.

Par exemple si le fichier à traiter est /etc/passwd, le caractère de séparation étant “:”, un enregistrement est une ligne du fichier, et un champ correspond au chaîne de caractère séparé par un “:” (login:mot de passe crypté:UID:GID:commentaires:home directory:shell).

Dans un enregistrement les champs sont référencés par $1, $2, ..., $NF (dernier champ). Par exemple pour /etc/passwd $1 correspond au login, $2 au mot de passe crypté, $3 à l'UID, et $NF (ou $7) au shell.

L'enregistrement complet (une ligne d'un fichier) est référencé par $0.

Par exemple, si l'on veut voir les champs login et home directory de /etc/passwd, on tapera:

awk -F“:” '{print $1,$6}' /etc/passwd

[retour haut de la page] Critères de sélection Présentation Un critère peut être une expression régulière, une expression ayant une valeur chaîne de caractères, une expression arithmétique, une combinaison des expressions précédentes.

Le critère est inséré entre les chaînes BEGIN et END, avec la syntaxe suivante:

awk -F“:” 'BEGIN{instructions} critères END{instructions}' fichier

BEGIN peut être suivi d'instruction comme une ligne de commentaire ou pour définir le séparateur. Exemple BEGIN { print“Vérification d'un fichier”; FS=“:”}. Le texte à afficher peut être un résumé de l'action de awk. De même pour END on peut avoir END{print “travail terminé”} qui indiquera que la commande a achevé son travail. Le END n'est pas obligatoire, de même que le BEGIN. Les expressions régulières La syntaxe est la suivante:

/expression régulière/ {instructions} $0 /expression régulière/ {instructions}

les instructions sont exécutées pour chaque ligne contenant une chaîne satisfaisant à l'expression régulière.

expression /expression régulière/{instructions}

les instructions sont exécutées pour chaque ligne où la valeur chaîne de l'expression contient une chaîne satisfaisant à l'expression régulière.

expression !/expression régulière/ {instructions}

les instructions sont exécutées pour chaque ligne où la valeur chaîne de l'expression ne contient pas une chaîne satisfaisant à l'expression régulière.

Soit le fichier adresse suivant (nom, numéro de téléphone domicile, numéro de portable, numéro quelconque):

gwenael | 0298452223 | 0638431234 | 50 marcel | 0466442312 | 0638453211 | 31 judith | 0154674487 | 0645227937 | 23

L'exemple suivant vérifie que dans le fichier le numéro de téléphone domicile (champ 2) et le numéro de portable (champ 3) sont bien des nombres.

awk 'BEGIN { print “On vérifie les numéros de téléphone; FS=“|”}

```
 $2 ! /^[0-9][0-9]*$/ { print "Erreur sur le numéro de téléphone domicile,
ligne n°"NR": \  n"$0}
 $3 ! /^[0-9][0-9]*$/ { print "Erreur sur le numéro de téléphone du
portable, ligne n°"NR": \  n"$0}
```

END { print “Vérification terminé”} ' adresse

BEGIN est suivi d'une instruction d'affichage qui résume la fonction de la commande, et de la définition du séparateur de champ. L'expression $2 se référe au deuxième champ d'une ligne (enregistrement) de adresse soit le numéro de téléphone domicile, on recherche ceux qui ne contiennent pas de chiffre (négation de contient des chiffres), en cas de succés on affichera un message d'erreur, le numéro de ligne courante, un retour à la ligne, puis le contenu entier de la ligne. L'expression $3 se référe au troisième champ d'une ligne (enregistrement) de adresse soit le numéro du portable, on recherche ceux qui ne contiennent pas de chiffre (négation de contient des chiffres), en cas de succés on affichera un message d'erreur, le numéro de ligne courante, un retour à la ligne, puis le contenu entier de la ligne. END est suivi d'une instruction d'affichage indiquant la fin du travail. Les expressions relationnelles Un critère peut contenir des opérateurs de comparaison (- <, ⇐,==,!=,>=,>). Exemple avec le fichier adresse suivant:

awk 'BEGIN { print “On cherche lignes dont le numéro (champ 4) est supérieur à 30”; FS=“|”} $4 > 30 { print “Numéro supérieur à 30 à la ligne n°”NR”: \ n“$0} END { print “Vérification terminé”} ' adresse

Combinaison de critères Un critère peut être constitué par une combinaison booléenne avec les opérateurs ou (||), et (&&) et non (!). Exemple:

awk 'BEGIN { print “On cherche la ligne avec judith ou avec un numéro inférieur à 30”; FS=“|”} $1 = = “judith” || $4 < 30 { print “Personne “$1” numéro “$4” ligne n°”NR”: \ n“$0} END { print “Vérification terminé”} ' adresse

Plage d'enregistrement délimitées par des critères La syntaxe est la suivante critère1,critère2 {instructions}. Les instructions sont exécutées pour toute les lignes entre la ligne répondant au critère1 et celle au critère2. L'action est exécutée pour les lignes comprises entre la ligne 2 et 6.

awk 'BEGIN NR==2;NR==6 { print “ligne n°”NR”:\ n“$0} END ' adresse

[retour haut de la page] Les actions Présentation Les actions permettent de transformer ou de manipuler les données, elles contiennent une ou plusieurs instructions. Les actions peuvent être de différents types: fonctions prédéfinies, fonctions de contrôle, fonctions d'affectation, fonctions d'affichage. Fonctions prédéfinies traitant des numériques atan2(y,x) arctangente de x/y en radian (entre -pi et pi)

cos(x) cosinus (radian)

exp(x) exponentielle à la puissance x

int(x) partie entière

log(x) logarithme naturel

rand(x) nombre aléatoire (entre 0 et 1)

sin(x) sinus (radian)

sqr(t) racine carrée

srand(x) définition d'une valeur de départ pour générer un nombre aléatoire Fonctions prédéfinies traitant de chaînes de caractères Pour avoir la liste des fonctions prédéfinies sur votre plate-forme vous devez faire un man awk, voici la liste des fonctions les plus courantes sur un système UNIX.

gsub(expression-régulière,nouvelle-chaine,chaine-de-caractères) dans chaine-de-caractères tous les caractères décrits par l'expression régulière sont remplacés par nouvelle-chaine. gsub et équivalent à gensub.

gsub(/a/,"ai",oi") Remplace la chaine oi par ai

index(chaine-de-caractères,caractère-à-rechercher) donne la première occurence du caractère-à-rechercher dans la chaine chaine-de-caractères

n=index("patate","ta") n=3 length(chaine-de-caractères) renvoie la longueur de la chaine-de-caractères

n=length("patate") n=6

match(chaine-de-caractères,expression-régulière) renvoie l'indice de la position de la chaîne chaine-de-caractères, repositionne RSTART et RLENGTH

n=match("PO1235D",/[0-9][0-9]/) n=3, RSTART=3 et RLENGTH=4

printf(format,valeur) permet d'envoyer des affichages (sorties) formatées, la syntaxe est identique de la même fonction en C

printf("La variable i est égale à %7,2f",i) sortie du chiffre i avec 7 caractères (éventuellement caractères vides devant) et 2 chiffres après la virgule.

printf("La ligne est %s",$0) > "fichier.int" Redirection de la sortie vers un fichier avec >, on peut utiliser aussi la redirection ». Veillez à ne pas oublier les "" autour du nom du fichier.

split(chaine-de-caractères,tableau,séparateur) scinde la chaîne chaine-de-caractères dans un tableau, le séparateur de champ est le troisième argument

n=split("zorro est arrivé",tab," ") tab[1]="zorro", tab[2]="est", tab[3]="arrivé", n=3 correspond au nombre d'éléments dans le tableau

sprintf(format,valeur) printf permet d'afficher à l'écran alors que sprintf renvoie la sortie vers une chaîne de caractères.

machaine=sprintf("J'ai %d patates",i) machaine="J'ai 3 patates" (si i=3)

substr(chaine-de-caractères,pos,long) Extrait une chaine de longueur long dans la chaîne chaine-de-caractères à partir de la position pos et l'affecte à une chaîne.

machaine=substr("Zorro est arrivé",5,3) machaine="o e"

sub(expression-régulière,nouvelle-chaine,chaine-de-caractères) idem que gsub sauf que seul la première occurence est remplacée (gsub=globale sub)

system(chaine-de-caractères) permet de lancer des commandes d'autres programmes

commande=sprintf(“ls | grep toto”) Exécution de la commande UNIX “ls |grep toto”

system(commande)

tolower(chaine-de-caracteres) retourne la chaîne de caractères convertie en minuscule

toupper(chaine-de-caracteres) retourne la chaîne de caractères convetie en majuscule Fonctions définies par l'utilisateur Vous pouvez définir une fonction utilisateur de telle sorte qu'elle puisse être considérée comme une fonction prédéfinie. La syntaxe est la suivant:

fonction mafonction(liste des paramètres) {

```
instructions
return valeur
```

}

[retour haut de la page] Les variables et opérations sur les variables Présentation On trouve les variables système et les variables utilisateurs. Les variables systèmes non modifiables donnent des informations sur le déroulement du programme. Les variables utilisateurs sont définies par l'utilisateur. Les variables utilisateur Le nom des variables est formé de lettres, de chiffres (sauf le premier caractère de la variable), d'underscore. Ce n'est pas nécessaire d'initialiser une variable, par défaut, si c'est un numérique, elle est égale à 0, si c'est une chaîne, elle est égale à une chaîne vide. Une variable peut contenir du texte, puis un chiffre, en fonction de son utilisation awk va déterminer son type (numérique ou chaîne). Les variables prédéfinies (système) Les variables prédéfinies sont les suivantes (en italique les valeurs par défaut):

ARGC nombre d'arguments de la ligne de commande néant

ARGIND index du tableau ARGV du fichier courant

ARGV tableau des arguments de la ligne de commande néant

CONVFMT format de conversion pour les nombres “%.6g”

ENVIRON tableau contenant les valeurs de l'environnement courant

ERRNO contient une chaîne décrivant une erreur “”

FIELIWIDTHS variable expérimentale à ne pas utiliser

FILENAME nom du fichier d'entrée néant

FNR numéro d'enregistrement dans le fichier courant néant

FS contrôle le séparateur des champs d'entrée “ ”

IGNORECASE contrôle les expressions régulières et les opérations sur les chaînes de caractères 0

NF nombre de champs dans l'enregistrement courant néant

NR nombre d'enregistrements lus jusqu'alors néant

OFMT format de sortie des nombres (nombre après la virgule) “%.6g”

OFS séparateur des champs de sortie “ ”

ORS séparateur des enregistrements de sortie

RLENGTH néant 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 néant

SUBSEP séparateur d'indiçage “\ 034” Opérations sur les variables On peut manipuler les variables et leur faire subir certaines opérations. On trouve différents types d'opérateurs, les opérateurs arithmétiques classiques (+, -, *, /, %(modulo, reste de la division), (puissance)), les opérateurs d'affectation (=, +=, -=, *=, /=, %=,  . Exemples:

var=30 affectation du chiffre 30 à var

var=“toto” affectation de la chaîne toto à var

var=“toto ” “est grand” concaténation des chaînes

“toto ” et “est grand”, résultat dans var

var=var-valeur var-=valeur expressions équivalentes

var=var+valeur var+=valeur expressions équivalentes

var=var*valeur var*=valeur expressions équivalentes

var=var/valeur var/=valeur expressions équivalentes

var=var%valeur var%=valeur expressions équivalentes

var=var+1 var++ expressions équivalentes

var=var-1 var– expressions équivalentes Les variables de champ Comme on l'a déjà vu auparavant les champs d'un enregistrement (ligne) sont désignés par $1, $2,...$NF(dernier champ d'une ligne). L'enregistrement complet (ou ligne) est désigné par $0. Une variable champ est a les mêmes propriétés que les autres variables. Le signe $ peut être suivi par une variable, exemple:

i=3 print $(i+1)

Provoque l'affichage du champ 4

Lorsque $0 est modifié, automatiquement les variables de champs $1,..$NF sont redéfinies.

Quand l'une des variables de champ est modifée, $0 est modifié. ATTENTION le séparateur ne sera pas celui définit par FS mais celui définit par OFS (output field separator). Exemple:

awk 'BEGIN{print“ # Tous les utilisateurs du groupe users(GID 22)basculeront dans le groupe boulot(GID 24)

```
        ";FS=":";OFS=":"}
```

```
  $4 != 22 {print $0} # Si le groupe n'est pas users on fait rien
  $4 ==22 {$4=24;print $0} # Si le groupe est 22, on lui réaffecte 24
```

END {print”C'est fini“}' /etc/passwd > passwd.essai

[retour haut de la page] Les structures de contrôle Présentation Parmi les structures de contrôle, on distingue les décisions (if, else), les boucles (while, for, do-while, loop) et les sauts (next, exit, continue, break). Les décisions (if, else) La syntaxe est la suivante:

if (condition) si la condition est satisfaite (vraie)

```
instruction1 on exécute l'instruction
```

else sinon

```
instruction2 on exécute l'instruction 2
```

Si vous avez une suite d'instructions à exécuter, vous devez les regrouper entre deux accolades. Exemple:

awk ' BEGIN{print”test de l'absence de mot de passe“;FS=”:“} NF==7 { #pour toutes les lignes contenant 7 champss

```
if ($2=="") # si le deuxième champ est vide (correspond au mot de passe
crypté))
{ print $1 " n'a pas de mot de passe"} # on affiche le nom de l'utilisateur
($1=login) qui n'a pas de mot de passe
else sinon
{ print $1 " a un mot de passe"} # on affiche le nom de l'utilisateur
possédant un mot de passe
```

} END{print”C'est fini“) ' /etc/passwd

Les boucles (while, for, do-while) while, for et do-while sont issus du langage de programmation C. La syntaxe de while est la suivante:

while (condition) tant que la condition est satisfaite (vraie) instruction on exécute l'instruction

Exemple:

awk ' BEGIN{print”affichage de tous les champs de passwd“;FS=”:“} { i=11 # initialisation du compteur à 1 (on commence par le champ 1) while(i=<NF) # tant qu'on n'est pas en fin de ligne

```
{ print $ii # on affiche le champ
i+++ # incrémentation du compteur pour passer d'un champ au suivant
}
```

} END{print”C'est fini“) ' /etc/passwd

La syntaxe de for est la suivante:

for (instruction de départ; condition; instruction d'incrémentation) On part d'une instruction de départ,

on incrémente instruction on exécute l'instruction jusqu'à que la condition soit satisfaite

Exemple:

awk ' BEGIN{print” affichage de tous les champs de passwd“;FS=”:“} { for (i=1;i⇒<NF;i++) # initialisation du compteur à 1, on incrémente le compteur jusqu'à ce qu'on atteigne NF (fin de la ligne)

```
{ print $i } # on affiche le champ tant que la condition n'est pas satisfaite
```

} END{print”C'est fini“) ' /etc/passwd

Avec for on peut travailler avec des tableaux. Soit le tableau suivant: tab[1]=“patate”, tab[2]=“courgette”,tab[3]=“poivron”. Pour afficher le contenu de chacun des éléments du tableau on écrira:

for (index in tab) { print “legume :” tab[index] }

La syntaxe de do-while est la suivante:

do {instructions}} on exécute les instructions while (condition) jusqu'à que la condition soit satisfaite

La différence avec while, est qu'on est sûr que l'instruction est exécutée au moins une fois. Sauts controlés (break, continue, next, exit) break permet de sortir d'une boucle, la syntaxe est la suivante:

for (;;;) boucle infinie {instructions on exécute les instructionss if (condition) break si la condition est satisfaite on sort de la boucle instructions}

Alors que break permet de sortir d'une boucle, continue force un nouveau passage dans une boucle.

next permet d'interrompre le traitrement sur la ligne courante et de passer à la ligne suivante (enregistrement suivant).

exit permet d'abandonner la commande awk, les instructions suivant END sont exécutées (s'il y en a).

Les tableaux

Présentation

Un tableau est une variable se composant d'un certains nombres d'autres variables (chaînes de caractères, numériques,...), rangées en mémoire les unes à la suite des autres. Le tableau est dit unidimensionnelle quand la variable élément de tableau n'est pas elle même un tableau. Dans le cas de tableaux imbriqués on parle de tableau unidimensionnels.

Les termes matrice, vecteur ou table sont équivalents à tableau. Les tableaux unidimensionnels Vous pouvez définir un tableau unidimensionnel avec la syntaxe suivante:tab[index]=variable, l'index est un numérique (mais pas obligatoirement, voir les tableaux associatifs), la variable peut être soit un numérique, soit une chaîne de caractère. Il n'est pas nécessaire de déclarer un tableau, la valeur initiale des éléments est une chaîne vide ou zéro. Exemple de définition d'un tableau avec une boucle for.

var=1 for (i=1;i⇐NF;i++)

```
{ mon-tab[i]=var++}
```

On dispose de la fonction delete pour supprimer un tableau (delete tab). Pour supprimer un élément de tableau on tapera delete tab[index]. Les tableaux associatifs Un tableau associatif est un tableau unidimensionnel, à ceci près que les index sont des chaînes de caractères. Exemple:

age[“olivier”]=27 age[“veronique”]=25 age[“benjamin”]=5 age[“veronique”]=3 for (nom in age) { print nom ” a “ age[nom] “ans” }

On a un tableau age avec une chaîne de caractères prénom comme index, on lui affecte comme éléments de tableau un numérique (age de la personne mentionnée dans le prénom). Dans la boucle for la variable nom est remplie successivement des chaînes de caractères de l'index (olivier, veronique, ...).

Les valeurs de l'index ne sont pas toujours triées. Les tableaux multidimensionnels awk n'est pas prévu pour gérer les tableaux multidimensionnels (tableaux imbriqués, ou à plusieurs index), néanmoins on peut simuler un tableau à deux dimensions de la manière suivante. On utilise pour cela la variable prédéfinie SUBSEP qui, rappelons le, contient le séparateur d'indiçage. Le principe repose sur la création de deux indices (i, j) qu'on va concaténer avec SUBSEP (i:j).

SUBSEP=”:“ i=“A”,j=“B” tab[i,j]=“Coucou”

L'élément de tableau “Coucou” est donc indexé par la chaîne “A:B”.