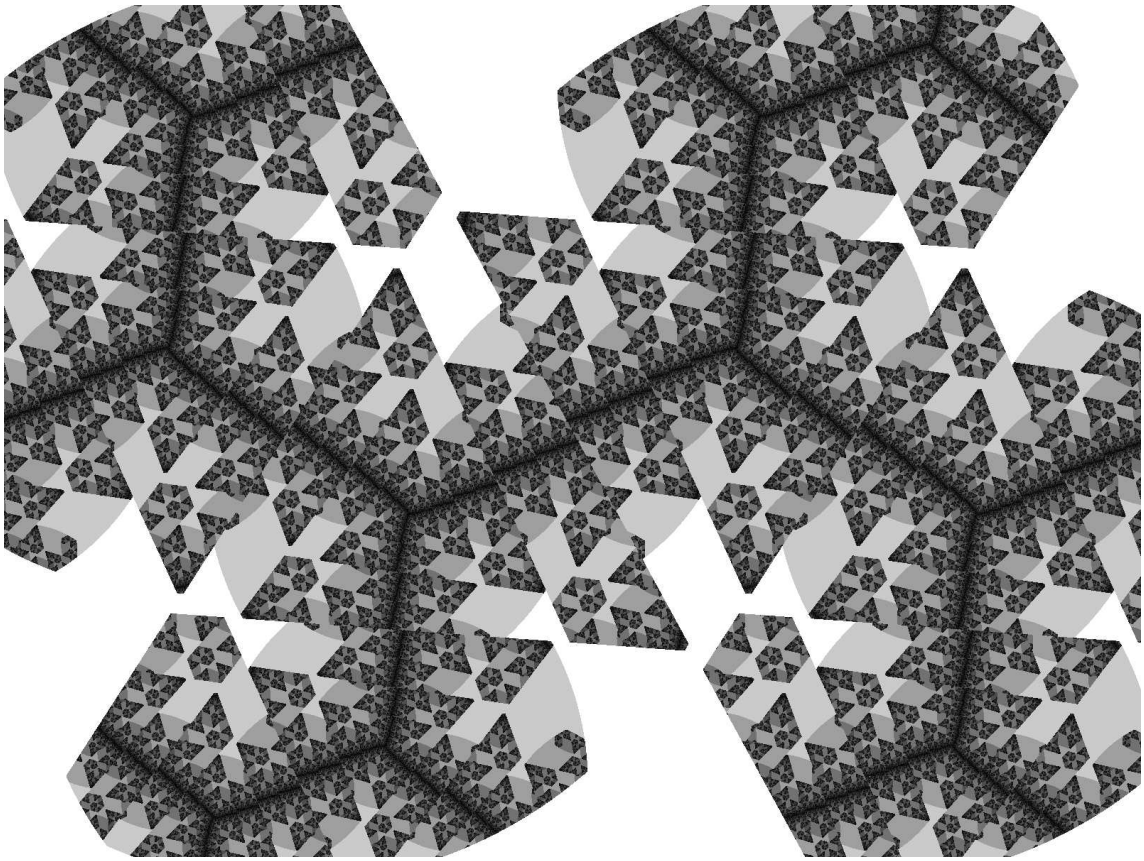
Unix

Petit complément de référence



Laurent Mazet mazet@softndesign.org

Table des matières

Chapitre 1

Avant propos

Ce manuel est un complément au « petit manuel de référence ». Vous y trouverez un rappel sur les expressions régulières ainsi que les descriptions du fonctionnement des commande *sed* et *awk*.

L'élaboration de ce petit fascicule a été très fortement inspirée des excellents documents publiés sur la toile (http :///www.shellunix.com) par M^{me} Isabelle Vollant.

Chapitre 2

Expressions régulières

Les expressions régulières caractérisent uniquement des chaînes de caractères et pas des noms de fichiers. Elles sont utilisées avec les commandes *ed*, *vi*, *ex*, *sed* et *awk*. Pour les exemples, nous utiliserons la commande de substitution de *vi* ou *sed* :

s/RE/chaîne de remplacement/ (ou RE caractérise une expression régulière)

Lorsqu'on recherche une chaîne de caractères à l'aide d'une expression régulière, la chaîne renvoyée est la chaîne la plus grande correspondant avec l'expression.

Pour banaliser un caractère, il faut utiliser « \ ».

2.1 « . »

Caractérise n'importe quel caractère.

Rappel : pour pouvoir caractériser le « . » , il faut le banaliser : « \. »

Exemple :

Chaîne de départ	asdeuy...dur
Commande	s/.../—/
Résultat	—euy...dur

2.2 « [] »

Un des caractères entre les crochets.

Si le premier caractère entre crochet est « ^ », alors cela caractérise un caractère qui ne correspond pas avec les caractères entre crochets.

[abc]	a, b ou c
[a-z]	Une lettre minuscule
[0-57]	0, 1, 2, 3, 4, 5 ou 7
[a-d5-8X-Z]	a, b, c, d, 5, 6, 7, 8, X, Y ou Z
[0-5-]	0, 1, 2, 3, 4, 5 ou -
[^0-9]	Pas un chiffre
[^a-zA-Z]	Pas une lettre
[012^]	0, 1, 2 ou ^

2.3 « ^ », « $ »

Descriptions :

– ^ : caractérise le début de ligne (attention ne pas confondre ^ et [^]).
– $: caractérise la fin de ligne.

Exemples :

/^abc/	Ligne commençant par abc
/abc$/	Ligne finissant par abc
/^$/	Ligne vide

2.4 « * »

Attention, l'utilisation de « * » est un peu particulière dans les expressions régulières :

Elle signifie de 0 à n fois le caractère la précédant.

a* 0 ou n fois *a*
aa* Au moins un *a*
.* N'importe quelle chaîne de caractères (y compris la chaîne vide)

Exemples :

Chaîne de départ	aabbabbaab		
Commande	s/[ab]*/x/	s/a.*b/y/	s/a.*bb/z/
Résultat	x	y	zaab

Autre exemple : /^[0-9][0-9]*$/ ligne qui ne contient que des chiffres.

2.5 « \(\) »

Pour isoler des sous chaînes. On peut réutiliser les sous chaînes ainsi trouvées grâce à \1, \2...

Exemples :

Chaîne de départ	ejkf fed 158e fd
Commande	s/.*\([0-9][0-9]*\).*/ result = \1/
Résultat	result = 158

Chapitre 3

Sed

« sed » est un éditeur non interactif. Cette commande permet d'appliquer un certain nombre de commandes sur un fichier puis d'en afficher le résultat (sans modification du fichier de départ) sur la sortie standard.

3.1 Syntaxe générale

Syntaxe : sed [-n] [-e commande] [-f fichier de commandes] [fichier]

Description des options :

-n	Ecrit seulement les lignes spécifiées (par l'option /p) sur la sortie standard
-e	Permet de spécifier les commandes à appliquer sur le fichier. Cette option est utile lorsque vous appliquez plusieurs commandes. Afin d'éviter que le shell interprète certains caractères, il faut mieux encadrer la commande avec des ' ou des " .
-f	Les commandes sont lues à partir d'un fichier.

Principe de fonctionnement : Pour chaque ligne, on applique la commande (si cela est possible) puis on affiche sur la sortie standard la ligne modifiée ou non.

La syntaxe générale des commandes est de la forme *caractérisation_des_adresses commandes* avec caractérisation_des_adresses de la forme :

	Toutes les lignes
num	La ligne *num* (la dernière ligne est référencée par $)
num1,num2	Les lignes entre les lignes *num1* et *num2*
RE	Les lignes correspondant à l'expression régulière *RE*
RE1,RE2	Les lignes entre la première ligne correspondant à l'expression régulière *RE1* et la première ligne correspondant à l'expression régulière *RE2*

3.2 La commande de substitution « s »

Syntaxe : ad1,ad2s/RE/remplacement/flags

Description : Remplace les expressions régulières *RE* par la chaîne de remplacement entre les lignes *ad1* à *ad2*

Descriptions des flags :

flags	
g	Global, c'est à dire toutes les occurrences de la chaîne RE (par défaut seule la première occurrence est remplacée)
p	Imprime la ligne (utile avec l'option -n)
w fichier	Ecrit la ligne dans le fichier spécifié en plus de la sortie standard.

Exemple :
- `sed "s/[Cc]omputer/COMPUTER/g" fichier`
  Remplacement de toutes les occurrences de « computer » et « Computer » dans le fichier.
- `sed -e "s/\([0-9][0-9]*\)/**\1**/" fichier`
  Encadrement du premier nombre de la ligne avec des **.

3.3 La négation « ! »

Syntaxe : ad1,ad2 !fonction argument

Description : La fonction est appliquée à toutes les lignes qui ne correspondent pas à la caractérisation.

3.4 La commande de suppression « d »

Description : Efface les lignes (au niveau de la sortie, le fichier d'origine n'est pas modifié).

Exemples :
- `sed "1,10d" fichier`
  La sortie correspond au fichier à partir de la onzième ligne.
- `sed "/^From/!d" fichier`
  Effacement de tout sauf les lignes commençant par *From* , et affichage des lignes commençant par *From*.

Remarque : Il est parfois plus facile de caractériser, la négation de ce que l'on veut (voir exemple précédent). Par exemple plutôt de récupérer ce que l'on veux, on efface ce qui ne nous intéresse pas

3.5 Les commandes d'insertions « a » et « i »

Descriptions :

- a\
  teste
  Ecrit le texte après la ligne.
- i\
  teste
  Ecrit le texte avant la ligne.

Exemple :

- Prenons le fichiers de commandes suivant :

```
1i\
\
----------------------------------------\
LOGIN USER\
----------------------------------------
s/:/!/
s/:/-/
s/:/-/
s/:/!/
/!-/d
```

```
/!\*-/d
/!!/d
s/!.*!/ /
s/,,,,//
s/:.*//
```

– Appliquons la commande : `sed -f fich_commandes /etc/passwd`
– Nous obtiendrons :

```
-------------------------------------
LOGIN                          USER
-------------------------------------
root   Operator
jd     Jean Dupond
vm     Vincent Martin
```

3.6 Les autres commandes : « q », « = » et « w »

Descriptions :

q	Quitte.
=	Ecrit les numéros de ligne.
w fichier	Ecrit dans un fichier

Pour mieux comprendre, prenons plusieurs exemples.

Exemple 1 :

– Prenons le fichier de commandes suivant :

```
Un,
deux.
Trois,
quatre.
```

– Commande :

```
sed -e "q" fichier
```

– Résultat :

```
Un,
```

Exemple 2 :

– Commande :

```
sed -e "/\./=/" -e "/[A-Z]/w capitale" fichier
```

– Résultat à l'écran :

```
Un,
2
deux.
Trois,
4
quatre.
```

– Résultat dans le fichier *capitale* :

```
Un,
Trois,
```

Chapitre 4

Awk

Cette commande permet d'appliquer un certain nombre d'actions sur un fichier. La syntaxe est inspirée du C.

4.1 Syntaxe générale

Syntaxe : awk [-Fs] [-v variable] [-f fichier de commandes] 'program' fichier

Description des options :

-F	Spécifie les séparateurs de champs
-v	Définie une variable utilisée à l'intérieur du programme.
-f	Les commandes sont lues à partir d'un fichier.

Principe de fonctionnement :

- Un *enregistrement* est une chaîne de caractères séparée par un retour chariot.
- Un *champ* est une chaîne de caractères séparée par un ou plusieurs espaces (ou par le caractère spécifié par l'option -F).
- La variable *NF* contient le nombre de champ de l'enregistrement courant.
- On accède à chaque champs par la variable *$1, $2, ... $NF*. *$0* correspond à l'enregistrement complet.
- Le programme est une suite d'actions de la forme : motif { action } .

Exemples :

- `awk -F ":" '{ $2 = "" ; print $0 }' /etc/passwd`
  Imprime chaque ligne du fichier /etc/passwd après avoir effacé le deuxième champ.
- `awk 'END {print NR}' fichier`
  Imprime le nombre total de lignes du fichiers.
- `awk '{print $NF}' fichier`
  Imprime le dernier champs de chaque ligne .
- `who | awk '{print $1,$5}'`
  Imprime le login et le temps de connexion.
- `awk 'length($0)>75 {print}' fichier`
  Imprime les lignes de plus de 75 caractères. (*print* équivaut à print $0).

4.2 Les variables pré-définies

Variable	Signification	Valeur par défaut
ARGC	Nombre d'arguments de la ligne de commande	-
ARGV	Tableau des arguments de la ligne de commande	-
FILENAME	Nom du fichier sur lequel on applique les commandes	-
FNR	Nombre d'enregistrements du fichier	-
FS	Séparateur de champs en entrée	" \t\n"
NF	Nombre de champs de l'enregistrement courant	-
NR	Nombre d'enregistrements déjà lus	-
OFMT	Format de sortie des nombres	"%.6g"
OFS	Séparateur de champs pour la sortie	" "
ORS	Séparateur d'enregistrement pour la sortie	"\n"
RLENGTH	Longueur de la chaîne trouvée	-
RS	Séparateur d'enregistrement en entrée	"\n"
RSTART	Début de la chaîne trouvée	-
SUBSEP	Séparateur de sous script	"\034"

4.3 Syntaxe du motif

Si le motif existe, cela détermine si l'action doit être appliquée à la ligne ou non.

Le motif peut être :

- Un expression régulière :
  - /expression régulière/
  - $0 ~ /expression régulière/
  - expression ~ /expression régulière/
  - expression !~ /expression régulière/
- Une expression *BEGIN* ou *END*.
- Une expression de comparaison : <, <=, ==, !=, >= et >.
- Une combinaison des trois (à l'aide des opérateurs booléens || ou, && et, ! négation).
- Une caractérisation des lignes.
- motif1,motif2 : Chaque ligne entre la première ligne correspondant au *motif1* et la première ligne correspondant au *motif2*.

Exemple 1 :

```
awk 'BEGIN { print "Verification des UID et GID dans le fichier /etc/passwd";
             FS=":"}
      $3 !~ /^[0-9][0-9]*$/ {print "UID  erreur ligne "NR" :\n"$0 }
      $4 !~ /^[0-9][0-9]*$/ {print "GID  erreur ligne "NR" :\n"$0 }
      END   { print "Fin" }
' /etc/passwd
```

Résultat :

```
Verification des UID et GID dans le fichier /etc/passwd
UID erreur ligne 14 :
clown:*:aaa:b:utilisateur en erreur:/home/clown:/bin:sh
GID erreur ligne 14 :
clown:*:aaa:b:utilisateur en erreur:/home/clown:/bin/sh
Fin
```

Exemple 2 :

```
awk 'BEGIN { print "Verification du fichier /etc/passwd pour ...";
              print "- les utilisateurs avec UID = 0 " ;
              print "- les utilisateurs avec UID >= 60000" ;
              FS=":"}
      $3 == 0 { print "UID 0 ligne "NR" :\n"$0 }
      $3 >= 60000  { print "UID >= 60000 ligne "NR" :\n"$0 }
      END   { print "Fin" }
' /etc/passwd
```

Résultat :

```
Verification du fichier /etc/passwd pour ...
- les utilisateurs avec UID = 0
- les utilisateurs avec UID >= 60000
UID 0 ligne 5 :
root:*:0:b:administrateur:/:/bin/sh
UID >= 60000 ligne 14 :
clown:*:61000:b:utilisateur en erreur:/home/clown:/bin/sh
Fin
```

Exemple 3 :

```
awk 'BEGIN { print "Verification du fichier /etc/group";
              print "le groupe 20 s'appelle t-il bien users ? " ;
              FS=":"}
      $1 == "users" && $3 ==20 { print "groupe "$1" a le GID "$3" !" }
      END   { print "Fin" }
' /etc/group
```

Résultat :

```
Verification du fichier /etc/group
le groupe 20 s'appelle t-il bien users ?
groupe users a le GID 20 !
Fin
```

Exemple 4 :

```
awk 'NR == 5 , NR == 10 {print NR" : " $0 }' fichier
```

Imprime de la ligne 5 à la ligne 10, chaque ligne précédée par son numéro

4.4 Syntaxe de l'action

Une action transforme ou manipule des données. Par défaut *print*.

Type des actions :

– Fonctions pré-définies, numériques ou sur les chaînes de caractères.
– Contrôle de flots.
– Affectation.
– Impression.

4.4.1 Fonctions numériques

Nom des fonctions	Signification
atan2(y,x)	Arc-tangente de $\frac{x}{y}$ en radians dans l'intervalle $]-\pi, \pi]$.
cos(x)	Cosinus (en radians).
exp(x)	Exponentielle e à la puissance *x*.
int(x)	Valeur entière.
log(x)	Logarithme naturel.
rand()	Nombre aléatoire entre 0 et 1.
sin(x)	Sinus (en radians).
sqrt(x)	Racine carrée.
srand(x)	Réinitialiser le générateur de nombre aléatoire.

Les opérations arithmétiques

On +, -, * et /.

4.4.2 Les fonctions sur les chaînes de caractères

Dans le tableau suivant :

- *s* et *t* représentent des chaînes de caractères.
- *r* une expression régulière.
- *i* et *n* des entiers.

Nom des fonctions	Signification.
gsub(r,s,t)	Sur la chaîne *t*, remplace toutes les occurrences de *r* par *s*.
index(s,t)	Retourne la position la plus à gauche de la chaîne *t* dans la chaîne *s*.
length(s)	Retourne la longueur de la chaîne *s*.
match(s,r)	Retourne l'index ou *s* correspond à *r* et positionne *RSTART* et *RLENTH*
split(s,a,fs)	Eclate *s* dans le tableau *a* sur *fs*, retourne le nombre de champs.
sprintf(fmt,liste expr.)	Retourne la liste des expressions formatées suivant *fmt*.
sub(r,s,t)	Comme *gsub*, mais remplace uniquement la première occurrence.
substr(s,i,n)	Retourne la sous chaîne de *s* commençant en *i* et de taille *n*.

4.4.3 Les variables et les expressions

Les opérations et affectations arithmétiques

- Les opérateurs arithmétiques sont les opérations usuelles : + - * / % (reste division entière) et ^ (puissance). Tous les calculs sont effectués en virgule flottante.
- La syntaxe de l'affectation : *var* = *expression*. Vous pouvez aussi utiliser les opérateurs +=, -=, *=, /=, %= et ^= ($x+=y$ équivaut à $x=x+y$).

Les variables de champs

Rappel : Les champs de la ligne courant sont : *$1*, *$2*, ... *$NF*.

La ligne entière est *$0*.

Ces variables ont les mêmes propriétés que les autres variables. Elles peuvent être réaffectées. Quand *$0* est modifiées, les variables *$1*, *$2* ... sont aussi modifiées ainsi que *NF*. Inversement si une des variables *$i* est

modifiées, *$0* est mise à jour. Les champs peuvent être spécifiés par des expressions, comme *$(NF-1)* pour l'avant dernier champs.

Exemple :

```
awk 'BEGIN { FS=":" ;
             OFS=":"}
     $NF != "/bin/ksh" { print $0 }
     $3 == "/bin/ksh" && NF == 7  { $7 = "/bin/posix/sh" ;
                                    print $0 } '
     /etc/passwd > /etc/passwd.new
```

Résultat : Création d'un nouveau fichier de mot de passe /etc/passwd.new en remplaçant le shell /bin/ksh par /bin/posix/sh.

Concaténation de chaînes de caractères

Il n'y a pas d'opérateur de concaténation, il faut simplement lister les chaînes à concaténer.

Exemple 1 :

```
awk '{ print NR " : " $0 }' fichier
```

Résultat : Numérotation des lignes du fichier .

Exemple 2 :

```
awk 'BEGIN { FS=":" ;
             OFS=":" ;
             print " Run Level 2 : Liste des actions "}
     $2 ~ /2/  { print "Keyword <<"$3">>, \n Tache <<"$4">>" }
     $2 == ""  { print "Keyword <<"$3">>, \n Tache <<"$4">>" }
' /etc/inittab > /etc/passwd.new
```

Résultat : Affichage des actions exécutées lors du passage à l'état 2.

4.4.4 Les structures de contrôle

while

Syntaxe : while (*condition*) *action*

for

Syntaxe : for (i = *petit* ; i <= *grand* ; i++) *action*

Fonctionnement proche de celui de la structure for du C.

ou

Syntaxe : for (*var* in *tableau*) *action*

Fonctionnement décrit plus loin.

break

Sortie de boucle pour while et for.

continue

Commence une nouvelle itération d'une boucle while ou for.

if else

Syntaxe : if (*expression*) *action* else *action*

commentaire et action vide

Le commentaire est précédé par « # ». tout ce qui est entre « # » et la fin de la ligne est ignoré par awk. Une action vide est représentée par « ; ».

next

Passe à l'enregistrement suivant et reprend le script awk à son début.

exit

Ignore le reste de l'entrée et exécute les actions définie par END.

4.4.5 Affichage

Syntaxes : print *exp*, *exp* ou print (*exp* , *exp*)

print equivaut à print $0.

printf *format* , *exp*, *exp* ou printf (*format*, *exp*, *exp*) permet de préciser un *format*. Un format est une chaîne de caractères et des constructeurs commençant par %.

Description des formats :

Spécifieur	Signification
d	Nombre décimal.
s	Chaîne de caractères.
-	Expression justifiée à gauche.
largeur	Largeur d'affichage.
.précision	Longueur maximale d'une chaîne de caractères ou du nombre de décimales.

Remarques :

– La sortie d'un print ou d'un printf peut être redirigée dans un fichier ou sur un pipe.
– Les noms de fichiers doivent être entre guillemets sinon ils sont considérés comme des variables.

Exemples :

– `awk ' { print NR " :" , $0 > "fich.numerote" } ' fichier`
  Le fichier *fich.numerote* contient le fichier *fichier* avec les lignes numérotées.
– `awk ' { printf "%3d : %s " , NR , $0 > "fich.numerote" } ' fichier`
  Le fichier *fich.numerote* contient le fichier *fichier* avec les lignes numérotées sur 3 caractères.

4.4.6 Les tableaux

On peut utiliser des tableaux de chaînes de caractères et de nombres à une dimension. Il n'est pas nécessaire de les déclarer. La valeur par défaut est "" ou 0. Les indices sont des chaînes de caractères.

Exemple 1 :

```
awk 'BEGIN { print "M{\'e}morisation de votre fichier " FILENAME }
                {memfile [NR] = $0 }
          END   { for ( i = NR ; i >= 1 ; i-- ) {
                    print i ":" memfile[i]
                    }
                  print "Fin"
                } ' fichier
```

Résultat : Affiche le fichier en commençant par la dernière ligne.

Exemple 2 :

```
awk ' NF > 0 {
                    for (i=1;i<=NF;i++) {
                        if ( $i ~ /^[0-9a-zA-Z][0-9a-zA-Z]*$/ ) {
                            index[$i] = index[$i] ";" NR "," i " " ;
                            index["total"]++ ;
                            }
                        }
                     }
            END     { x="total" ;
                      printf("%s mots detect{\'e}s = %d\n",x,index[x]);
                      } ' fichier
```

Résultat : Construction d'un index de références croisées.

For et les tableaux

Comme les indices des tableaux sont des chaînes de caractères, on ne peut pas déterminer la taille d'un tableau. On doit donc utiliser la construction : for (*var* in *tableau*) *action*.

Exemple :

```
awk ' NF > 0 {
                    for (i=1;i<=NF;i++) {
                        if ( $i ~ /^[0-9a-zA-Z][0-9a-zA-Z]*$/ ) {
                            index[$i] = index[$i] ";" NR "," i " " ;
                            index["total"]++ ;
                            }
                        }
                     }
            END     {
                    for ( x in index ) {
                     if ( x != "total" )
                       printf("%-20s\t%s\n",x,index[x]) | "sort -f "
                      }
                      x="total";
                      printf("%s mots detect{\'e}s = %d\n",x,index[x]);
                      } ' fichier
```

Résultat : Construction d'un index de références croisées.

Simulations des tableaux multidimensions

On ne peut pas utiliser des tableaux multidimentionnels. On peut les simuler en concatenant des composants des sous-chaînes avec le séparateur SUBSEP. Exemple :

```
awk 'BEGIN { print "M{\'e}morisation de votre fichier " FILENAME
                  SUBSEP=":"
                }
                {  for ( i=1 ; i <=NF ; i++ ) {
                   memfields[ NR , i ] = $i
                   }
                }
          END   { for ( i in memfields ) {
                   print i ":" memfields[i] | "sort -n -t: "
                   }
                 print "Fin"
                } ' fichier
```

Résultat : Affiche le fichier en commençant par la dernière ligne.

Annexe A

Références et licence

La dernière version est disposible à l'adresse http ://www.softndesign.org/ ?page=manuels/unix-3.php. Des versions Postscript et PDF y sont aussi disponibles.