

# Programmation système - Shell et Commandes UNIX

Gestion des filtres avancés

Tuyêt Trâm DANG NGOC

<dntt@u-cergy.fr>

Université de Cergy-Pontoise

- 1 Expressions rationnelles
- 2 tr
- 3 sed
- 4 awk
  - Présentation et syntaxe
  - Fonctions
  - Variables
  - Structures de contrôles
  - Tableaux
- 5 Exemples
- 6 Crédits

# Expressions rationnelles (regular expressions - regexp)

- `'.'` : un joker, qui représente un caractère unique quelconque (sauf caractères de contrôle et fin de ligne).
- `'[ ]'` : des classes de caractères entre crochets `[ ]`.
- `'^'` : au début d'une classe de caractères entre crochets, signifie qu'on considère le complément de cette classe (l'ensemble des caractères qui ne sont pas dans la classe).
- `'^'` et `'$'` : représentent respectivement un début et une fin de ligne.
- `'*'` répétition de 0 à  $n$  fois le caractère précédent (attention, ne pas confondre avec le `'*'` utilisé par le shell pour lister les fichiers)
- `'\'` permet de considérer le caractère suivant comme un caractère normal.

Utilisé dans `grep`, `sed`, `awk`, `ruby`, `javacc`, `lex`, `php`, `shells`, `perl`, `sql`, etc.

# Exemple regexp

- « `.ac` » représente les chaînes de 3 caractères qui se terminent par « `ac` »
- « `[a-z]` » correspond à n'importe quelle lettre minuscule (non-accentuée)
- « `[^a-z]` » correspond à n'importe quel caractère qui n'est pas une lettre minuscule non-accentuée
- « `[st]ac` » représente entre autres « `sac` » et « `tac` »
- « `[^f]ac` » représente les mots de trois lettres qui se terminent par « `ac` » et ne commencent pas par « `f` »
- « `^[st]ac` » représente les mots « `sac` » et « `tac` » en début de ligne
- « `[st]ac$` » représente les mots « `sac` » et « `tac` » en fin de ligne
- « `^trax$` » représente le mot « `trax` » seul sur une ligne
- « `ba*` » représente le mot « `b` », « `ba` », « `baa` », ... « `baaaaaaaaaaaaaaaaa` », ...
- « `.*` » représente n'importe quel mot
- « `\*` » représente le caractère « `*` »

- 1 Expressions rationnelles
- 2 tr
- 3 sed
- 4 awk
  - Présentation et syntaxe
  - Fonctions
  - Variables
  - Structures de contrôles
  - Tableaux
- 5 Exemples
- 6 Crédits

## Conversion de chaînes de caractères : tr

## tr

- c Tous les caractères qui ne sont pas spécifiés dans la première chaîne sont convertis selon les caractères de la seconde.
- d Efface le caractère spécifié.
- s Si le caractère spécifié se répète plusieurs fois de suite, il est réduit à une seule unité.

```
$ cat fichier
j'ai un chapeau de paille
$ cat fichier | tr "[A-Z]" "[a-z]"
J'AI UN CHAPEAU DE PAILLE
$ tr "abc" "AB" << EOF
> j'ai un chapeau de paille
> EOF
j'Ai un BhApeAu de pAille
$ tr -d "ac" < fichier
j'i un hpeu de pille
```

## Conversion de chaînes de caractères : tr

```
$ ls -l
total 36
drwx----- 3 dntt      users  4096 mar 15 14:35 gconfd-dntt
drwx----- 2 ntravers users  4096 mar  8 14:23 gconfd-ntra
drwx----- 2 root      root   4096 mar  8 15:45 gconfd-root
$ ls -l | tr -s " "
total 36
drwx----- 3 dntt users 4096 mar 15 14:35 gconfd-dntt
drwx----- 2 ntravers users 4096 mar 8 14:23 gconfd-ntraver
drwx----- 2 root root 4096 mar 8 15:45 gconfd-root
```

- 1 Expressions rationnelles
- 2 tr
- 3 sed**
- 4 awk
  - Présentation et syntaxe
  - Fonctions
  - Variables
  - Structures de contrôles
  - Tableaux
- 5 Exemples
- 6 Crédits



# Éditeur de flux : sed

## sed

Éditeur de flux

-e commande

commande d'édition semblable à celles de l'éditeur **vi**

- La commande de substitution **s** :  
`ad1,ad2s/RE/remplacement/flags` où si `flags` vaut `g` : global, c'est à dire toutes les occurrences de la chaîne `RE` (par défaut seule la première occurrence est remplacée).  
 négation : précéder la commande de `!`
- La commande de suppression **d**
- Les commande d'insertions **a,i**
  - `a texte` : écrit le texte après la ligne
  - `i texte` : écrit le texte avant la ligne

# sed : exemples

```
$ cat toto
```

```
ga
```

```
bu
```

```
zo
```

```
meuh
```

```
$ sed -e "a xxx" toto
```

```
ga
```

```
xxx
```

```
bu
```

```
xxx
```

```
zo
```

```
xxx
```

```
meuh
```

```
xxx
```

```
$ sed -e "i xxx" toto
```

```
xxx
```

```
ga
```

```
xxx
```

```
bu
```

```
xxx
```

```
zo
```

```
xxx
```

```
meuh
```

## sed : exemples plus complexes

Remplace les mots Chat et chat par CHAT sur toute la ligne dans tout le fichier.

```
sed "s/[Cc]hat/CHAT/g" fichier
```

Encadre le premier nombre trouvé sur la ligne avec des \*\*

```
sed -e "s/\([0-9][0-9]*\)/**\1**/" fichier
```

Remplace tous les ':' du fichier par une espace.

```
sed -e "s:/ /g" fichier
```

Affiche le contenu du fichier à partir de la 11ème ligne.

```
sed "1,10d" fichier
```

On efface tout sauf les lignes commençant par From , donc on imprime les lignes commençant par From.

```
sed "/^From/!d" fichier
```

- 1 Expressions rationnelles
- 2 tr
- 3 sed
- 4 awk**
  - Présentation et syntaxe
  - Fonctions
  - Variables
  - Structures de contrôles
  - Tableaux
- 5 Exemples
- 6 Crédits

# Éditeur de flux évolué : **awk**

## **awk** 'programme' fichier

Langage de recherche de motif et manipulation de texte

- F Spécifie les séparateurs de champs
- v Définit une variable utilisée à l'intérieur du programme.
- f Les commandes sont lu à partir d'un fichier.
- Un **enregistrement** est une chaîne d'entrée délimitée par un retour chariot
- Un **champ** est une chaîne délimitée par un séparateur de champ (FS) dans un enregistrement. Le FS peut être défini dans le bloc BEGIN. Par défaut, c'est une espace.

Suite d'actions :

```
motif { action }
```

champs d'un enregistrement (ligne) sont désignés par  
\$1, \$2, ... \$NF

# Utilisation générale de awk

```
awk 'BEGIN {
    instructions exécutées au début du programme,
    une seule fois seulement.
}
critère1 {
    instructions
}
critère2 {
    instructions
}
...
END {
    instructions exécutées à la fin du programme,
    une seule fois seulement.
}'
```

BEGIN et END ne sont pas obligatoires.

# awk : exemples simples

Affiche toutes les lignes du fichiers

```
cat fichier | awk '{ print $0 }'
```

Affiche le second champ de chaque ligne du fichier

```
cat fichier | awk '{ print $2 }'
```

Affiche toutes les lignes comprenant /\* et \*/

```
cat un_fichier.c | awk '/\/*/, /*\*/ { print $0 }'
```

Affiche toutes les lignes dont le premier caractère non-blanc n'est pas un #

```
cat un_fichier | awk '! (/^ *#/ || /^$/) { print $0 }'
```

# Critères

- `/expression régulière/`
- `$0 /expression régulière/`
- `$0 !/expression régulière/`
- une expression de comparaison : `<`, `<=`, `==` , `!=`, `>=`, `>`
- une combinaison (opérateurs booléens `||` `&&` `!`)
- `motif1,motif2` : chaque ligne entre la première ligne correspondant au `motif1` et la première ligne correspondant au `motif2`



# Type des actions

- fonctions prédéfinies, numérique ou chaîne de caractères
- contrôle de flux
- affectation
- impression
  
- # commentaire
- ; instruction vide

# Affichage

<code>print exp, exp</code> <code>print (exp, exp)</code>	ou	affiche les expressions
<code>print</code>		equivaut à <code>print \$0</code>
<code>printf format, exp, exp</code> ou <code>printf (format, exp, exp)</code>		identique à <code>print</code> mais avec format ( <code>printf</code> en C)

Un format est une chaîne de caractères et des constructeurs commençant par % specifieur signification

<code>d</code>	nombre decimal
<code>s</code>	chaîne de caractères
specifieur	signification
-	expression justifiée à gauche
largeur	largeur d'affichage
.precision	longueur maximale d'une chaîne de caractères
ou nombre de decimales	

# Exemple de manipulation sur les champs

```
toto | 0298452223 | 0638431234 | 50
titi | 0466442312 | 0638453211 | 31
tutu | 0154674487 | 0645227937 | 23
```

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

```
cat adresse | awk '
BEGIN { print "On vérifie les numéros de téléphone; FS="|" }

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

$3 ! /^[0-9][0-9]*$/ {
print "Erreur sur le numéro de téléphone du portable"
print " ligne n`NR": \ n"$0}
END { print "Vérification terminé" } '
```

# Fonctions prédéfinies (numériques)

Nom des fonctions	signification
<code>atan2(y,x)</code>	arc tangente de $x/y$ en radians dans l'intervall $-\pi$ $\pi$
<code>cos(x)</code>	cosinus (en radians)
<code>exp(x)</code>	exponentielle $e$ à la puissance $x$
<code>int(x)</code>	valeur entière
<code>log(x)</code>	logarithme naturel
<code>rand()</code>	nombre aléatoire entre 0 et 1
<code>sin(x)</code>	sinus (en radians)
<code>sqrt(x)</code>	racine carrée
<code>srand(x)</code>	reinitialiser le générateur de nombre aléatoire
...	...

## Fonctions prédéfinies (chaines de caractères)

Nom des fonctions	signification
<code>gsub(r,s,t)</code>	sur la chaîne t, remplace toutes les occurrences de r par s
<code>index(s,t)</code>	retourne la position la plus à gauche de la chaîne t dans la chaîne s
<code>length(s)</code>	retourne la longueur de la chaîne s
<code>match(s,r)</code>	retourne l'index où s correspond à r et positionne RSTART et RLENGTH
<code>split(s,a,fs)</code>	split s dans le tableau a sur fs, retourne le nombre de champs
<code>sprintf(fmt,liste expressions)</code>	retourne la liste des expressions formatée suivant fmt
<code>sub(r,s,t)</code>	comme gsub, mais remplace uniquement la première occurrence
<code>substr(s,i,n)</code>	retourne la sous chaîne de s commençant en i et de taille n

# Fonctions définies par l'utilisateur

```
fonction mafonction(liste des paramètres)
{
    instructions
    return valeur
}
```

# Variables définies par l'utilisateur

Pas de déclaration de variable, on utilise une variable quand on en a besoin. Le typage est déterminé automatiquement par awk (chaîne ou numérique).

```
valeur=5  
var=30  
var=var-valeur  
var="toto"  
var="toto " "va a la plage"
```

# Variables prédéfinies

Variable	Signification	Val. par déf.
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	separateur de champs en entrée	" "
NF	nombre de champs de l'enregistrement courant	-
NR	nombre d'enregistrements déjà lu	-
OFMT	format de sortie des nombres	"%.6g"
OFS	separateur de champs pour la sortie	" "
ORS	separateur d'enregistrement pour la sortie	"\n"
RLENGTH	longueur de la chaîne trouvée	-
RS	separateur d'enregistrement en entrée	"\n"
RSTART	debut de la chaîne trouvée	-
SUBSEP	separateur de subscript	"\034"

Les champs de la ligne courante sont : \$1, \$2, ..., \$NF. La ligne entière est \$0



# Tests : if, else

if,else

```
if (condition) instruction1 else instruction2
```

Soit le fichier /etc/passwd :

```
login:motdepasse:uid:gid:geco:repertoiremaison:shell
```

```
cat /etc/passwd | awk ' 
```

```
BEGIN { print "test de l absence de mot de passe"; FS=":"}  
NF==7
```

```
{ #pour toutes les lignes contenant 7 champs  
  if ($2=="") # si le deuxieme champ est vide  
  { print $1 " n a pas de mot de passe"}  
  else  
  { print $1 " a un mot de passe"}  
}
```

```
END { print ("C est fini")}
```

```
,
```

# Boucle : while

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

```
while
```

```
while (condition) instruction
```

Exemple :

```
cat /etc/passwd | awk '
BEGIN {print "affichage de tous les champs de passwd";FS=":"}
{ i=1
while (i<NF) # tant qu on n est pas en fin de ligne
  { print $i # on affiche le champ
  i++      # incrementation du compteur pour passer
           # au champ suivant
  }
}
END {print ("C'est fini")}'
```

# Boucle : do, while

On exécute les instructions jusqu'à que la condition soit satisfaite

do, while

do instructions while (condition)

# Boucle : for

for

for (instruction de départ ; condition ; instruction d'incréméntation)

```
cat /etc/passwd | 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 a ce
                    # qu on atteigne NF (fin de la ligne)
    { print $i }      # on affiche le champ
}
END {print"C'est fini"}
```

# Sauts : break, continue, next, exit

- Break : sortie de boucle
- Continue : commence une nouvelle itération de la boucle.
- Next : passe à l'enregistrement suivant. On reprend le script awk à son début
- Exit : ignore le reste de l'entrée et exécute les actions définies par END

# Tableaux

Un tableau est une variable se composant d'un certains nombres d'autres variables (chaînes de caractères, numériques,...)

```
tab[index]=valeur
```

L'index est soit un numérique soit une chaîne de caractère

```
tableau[1]="truc"
```

```
age["toto"]=27
```

```
age["titi"]=42
```

```
for (index in tab) {  
  print "legume :" tab[index]  
}
```

- 1 Expressions rationnelles
- 2 tr
- 3 sed
- 4 awk
  - Présentation et syntaxe
  - Fonctions
  - Variables
  - Structures de contrôles
  - Tableaux
- 5 Exemples
- 6 Crédits

```
cat /etc/passwd | awk '
# Tous les utilisateurs du groupe users(GID 22)basculeront
# groupe boulot(GID 24) ";
BEGIN {print"  FS=":"; OFS=":"}

$4 != 22 {print $0}      # Si le groupe n'est pas 22
                        # on ne fait rien

$4 ==22 {$4=24;print $0} # Si le groupe est 22, on lui
                        # reassigne 24

END {print"C'est fini"}
' > passwd.essai
```



- 1 Expressions rationnelles
- 2 tr
- 3 sed
- 4 awk
  - Présentation et syntaxe
  - Fonctions
  - Variables
  - Structures de contrôles
  - Tableaux
- 5 Exemples
- 6 **Crédits**

# Crédits

Ces transparents ont été réalisés en s'inspirant des pages web suivantes :

- <http://www.lmd.ens.fr/Ressources-Info/Unix-Doc/html/cours-unix-13.html>
- [http://www.xgarreau.org/aide/admin/boutils/awk\\_lea.php](http://www.xgarreau.org/aide/admin/boutils/awk_lea.php)
- <http://www.shellunix.com/awk.html>