Chapitre 6

LES FILTRES

Objectif:

Connaître et manipuler les différentes techniques de recherche, d'organisation et d'accès aux données se trouvant dans le système de fichier (fichiers et répertoires).

Prérequis:

① Les chapitres précédents de ce cours d'UNIX.

Plan

Chapitre 6

LES FILTRES

I/ LA COMMANDE GREP: (RECHERCHE DANS LE TEXTE)

La commande **grep** permet la recherche, dans un ou plusieurs fichiers, des lignes qui correspondent à une séquence donnée.

Sa **syntaxe** est:

```
$grep [-option]  séquence  [fichier(s)]
```

Parmi les options de grep:

-v : Affichage des lignes qui ne contiennent pas la séquence.

-c : Affichage du nombre d'occurrence.

-l : Affichage des noms de fichiers contenant la séquence.

-n : Chaque ligne affichée est numérotée.

-y : Confondre les minuscules et les majuscules.

Exemples:

1- recherche du mot "amitié" dans le fichier humanite

```
$grep amitié humanite
$grep -v amitié humanite
$grep -n amitié humanite
```

2- savoir si l'utilisateur user2 est connecté:

```
$who|grep user2
```

La commande grep supporte les métacaractères:

Caractères spéciaux	Signification
[.....]	identification d'une plage pour une série de caractères.
[^.....]	négation de la plage indiquée.
.	n'importe quel caractère y compris l'espace
*	caractère de répétition, agit sur le caractère précèdent.
^	symbolise un début de ligne dans un critère de recherche.
$	matérialise la fin de le ligne
\{.....\}	caractère de répétition. Entre les accolades vous pouvez définir librement le nombre de survenances possibles pour le caractère précèdent.

Exemples:

1- Trouver et afficher toutes les lignes du fichier timbre commençant par une majuscule.

```
$grep '^[A-Z].*' timbre
ou  $grep '^[A-Z]\{1,\}'  timbre
```

2- afficher les processus de tous les utilisateurs userx

```
$ ps -f|grep user.*
```

3- afficher à partir du fichier /etc/passwd les utilisateurs qui commencent par les lettres 'a' jusqu'à 'e':

```
$grep '^[a-e].*'  /etc/passwd
```

Exercice : Expliquez la commande suivante:

```
$grep '^[a-e].*[:]$'  /etc/.passwd
```

La commande grep n'est qu'un membre d'une famille qui comprend deux autres commandes dérivées: **egrep** et **fgrep**.

❄ Le programme **fgrep** recherche plusieurs séquences à la fois.

Sa s**yntaxe** est:

```
fgrep  -option  expr_reg  fichier(s)
ou  fgrep  -option  -f  fich_expr  fichier(s)
```

❄ Le programme **egrep** interprète des expression régulières, utilisant les parenthèses et l'opérateur logique "ou". Sa syntaxe est la même que fgrep.

Exemples:

1- recherche de deux mots dans le fichier "virus":

```
$fgrep  'programmeur
>programme'  virus
```

2- Affichage des lignes de /etc/passwd contenant les utilisateurs userx et quelques utilisateurs IIX :

```
$egrep  'user*|II[1-5]' /etc/passwd
```

3- affichage d'un mot et quelques dérivés:

```
$egrep 'programm(es|e|ation)' virus
```

II/ LA COMMANDE SORT: (TRI ET FUSION)

Le programme **sort** à pour rôle de trier un ou plusieurs fichiers ligne par ligne. Le tri se fait sur la base de clés de tri qui peuvent êtres un ou plusieurs champs de la ligne ou une partie d'un champ.

La **syntaxe** est:

```
sort -option +pos1 -pos2 fichier_in -o fichier_out
```

parmi les options de sort:

-r : tri en ordre inverse

-n : tri en ordre numérique

-d : seuls sont considérés l'alphabet, les chiffres et le "blanc"

-u : suppression de tous les lignes doubles, sauf un.

Pos1 et pos2 indiquent le début et la fin de la clé de tri et peuvent prendre le valeurs m ou m.n:

+ m.n est interprétée comme le $(n+1)^{ème}$ caractère du $(m+1)^{ème}$ champ

-m.n est interprétée comme le $n^{ème}$ caractère du $m^{ème}$ champs.

Exemples:

1- tri des utilisateurs courants:

```
$who|sort
```

2- tri en ordre croissant: du fichier "fich1":

```
$sort -r fich1
```

3- tri et fusion de deux fichiers "fich1" et "fich2" sur le troisième champ. Le résultat est sauvegardé dans fich3.

```
$Sort +2 fich1 fich2 -o fich3
```

4- tri des utilisateurs selon l'heure de connexion:

```
$who |sort +4
```

Le tri peut se faire selon plusieurs clés de tri:
un clé primaire (déclaré en premier lieu) et des clés secondaires (déclarés après le clé primaire.

Exemple:

tri des processus en cours: le premier champ est le clé primaire et le deuxième est le clé secondaire.

```
$ps -f|sort +0 +1
```

III/ LA COMMANDE FIND: (RECHERCHE DE FICHIERS)

La commande **find** sert à parcourir l'arborescence des répertoires à la recherche des fichiers. Pour ce faire, il va falloir fixer des critères de recherche.

La commande find parcourt les répertoires et leurs sous-repertoires de manière récursive, à la recherche de fichiers.

Trois indication sont nécessaires a l'exécution de la commande find.

1- répertoire de recherche (spécification)

2- les critères de recherche à mettre en oeuvre

3- comportement si un fichier répond aux critères.

La **syntaxe** de la commande find est la suivante:

find répertoires [critères _de_séléction] [option_de_commande]

Si aucune indication n'est faite, la commande find recherche tous les fichiers. Il est possible de mentionner des options de recherche pour limiter la recherche. Si cette commande à trouvée un fichier correspondant au critère, il est traité par les options de commande. On peut utiliser -print, -exec et -ok.

Les options de choix utilisables pour la recherche sont:

OPTION	SIGNIFICATION
-name fichier	recherche par nom de fichier
-type type	recherche par type de fichier
-user nom	recherche par propriétaire
-group nom	recherche par groupe
-size nombre	recherche par taille
-atime jour	recherche par date du dernier accès
-mtime jour	recherche par date de dernière modification
-ctime jour	recherche par date de création
-perm droit	recherche par droit d'accès
-lien nombre	recherche par nombre de lien

Pour toutes les options en dehors de -name, -type, -user, -group et -perm, les nom d'options sont complétés par des valeurs. Ces chiffres peuvent aussi êtres précédés des signes "+" ou "-".

"+" signifie "supérieur à".

"-" signifie "inférieur à".

Exemple:

pour trouver tous les fichiers de répertoire /work ayant plus de quatre liens:

```
$find /work -lien +4 -print
```

L'option -exec introduit une commande qui sera exécuté si un fichier correspondant au critère de sélection est trouvé:

Attention ! ! ✓ La commande placée derrière -exec doit se terminer par "/;".

✓ si on veut accéder, dans la commande placée derrière -exec, au fichier qui vient d'être trouvé, on doit utiliser l'abréviation {}.

Exemple:

Si on veut exécuter ls -l pour chaque fichier trouvé:

```
$find . -user user3 -exec ls -l {}\;
```

Pour l'option de commande -ok, on applique la même syntaxe que pour -exec. Mais dans cas, il sera demandé si on souhaite exécuter cette commande. Si la réponse est "y", la commande find exécute la commande qui suit l'option -ok.

Exemple:

```
$find . -ok rm{]\;
<rm.....>?n
```

Si plusieurs options sont spécifiés, elles sont censées êtres liées par un lien logique/

! pour la négation logique

-a pour le ET logique

-o pour le OU logique

Exemples:

1- Afficher tous les fichiers qui sont soit des répertoires soit dont le nom termine par "ier", à partir du répertoire actif:

```
$find . \(-type d -o -name "*ier"\) - print
```

2- Afficher tous les répertoires placés sous le répertoire actif et commençant par une minuscule:

```
$find . -name "[a-z]*" -type d -print
```

ou

```
$find . \(-name "[a-z]*" -a -type d\) -print
```

3- Afficher tous les fichiers ayant une taille supérieur à 400 blocs (200 ko)

```
$find . -type f -size +400 -print
```

4- Afficher tous les fichiers appartenant a user2. La recherche est effectuée dans le répertoire /dev et /usr/user2.

```
$Find /dev /usr/user2 -user user2 -exec ls -l {]\;
```

IV/ LA COMMANDE awk: (BALAYAGE DANS LE TEXTE)

L'utilitaire **awk** lit ligne par ligne dans un fichier spécifié ou par le canal d'entrée standard. Chaque ligne est subdivisée en champs.

La **syntaxe** de base est:

```
        awk 'programme' fichiers
ou      awk -f  progfile     fichiers
```

Si le programme est lu à partir d'un fichier, le programme est un ensemble de conditions et d'actions écrites de la façon suivante:

```
condition1      {action1}
condition2      {action2}
        .       .
        .       .
        .       .
conditionN      {actionN}
```

Le module awk traite les fichiers en entrée article par article. Chaque ligne est comparée au différentes conditions du programme. Si la séquence est vraie, alors l'action est exécutée.

IV-1/ Les variables prédéfinis de awk

Le programme awk divise automatiquement chaque ligne en champs qui sont normalement délimités par des "blancs" ou <TAB>. Ces champs auront pour noms $1,$2,.....,$NF. La constante $0 se réfère à l'article entier. Les différentes variables de awk sont:

VARIABLE	SIGNIFICATION
FILENAME	nom du fichier en entrée courant
FS	séparateur de champs des articles en entrée.
NF	nombre de champs des articles en entrée
NR	nombre des articles en entrée
RS	séparateur d'articles en entrée
OFS	séparateur de champs des articles en sortie
ORS	séparateur d'articles en sortie

IV-2/Les opérateurs et fonctions pré-définies de awk

Les conditions du programme awk peuvent êtres des expressions régulières, comme c'est le cas de grep et egrep, ou des conditions composées avec l'utilisation d'opérateurs

divers, de fonctions prédéfinies et de structures conditionnelles semblables à ceux du langage C (if, while, do, for, break, continue...). Par exemple, la structure suivante:

```
$awk '/expression régulière / {print}' fichiers
```

permet de parcourir les fichiers citées et d'afficher chaque ligne qui comprend l'expression mentionnée. Si l'expression est omise, toutes les lignes seront affichées. Si l'action est omise, c'est l'action print qui sera prise par défaut. Les opérations utilisées sont les même que celles utilisé dans le langage C.

Les fonctions pré-définies de awk sont multiples. Les plus utilisées sont:

FONCTION	DESCRIPTION
index(seq1,seq2)	donne la position de se2 dans seq1
length(seq)	donne la longueur en caractères de seq
split(seq,a,fs)	divise la séquence seq en n éléments de tableau a[1], a[2],.....,a[n] et où fs est le séparateur de champs (FS par défaut). La valeur de n est donnée en sortie.
substr(seq,m,n)	donne les n caractères de la sous-séquence de seq qui commence à la position m.
getline	lecture de la ligne suivante.

Exemples:

1- Subdivision du résultat de who en plusieurs champs et affichage des noms triés, date et nombre des champs:

```
$who|awk '{print $1,$NF,NF}' |sort
```

2- Affichage des lignes du fichier fich1 qui contiennent un nombre impair de mots:

```
$awk 'NF%2!=0 {print $0}' fich1
```

3- Affichage des lignes de fich1 qui dépassent les 75 colonnes:

```
$awk 'length>75  {print}' fich1
```

4- Affichage des caractères indésirables pour les lignes qui dépassent les 75 colonnes:

```
$cat fichprg
length > 75  {print  "résidu  ligne ", NR,, substr($0,75,NR)}
^d
$awk  -f fichprg   fich1
```

IV-3/ Les modèles de awk

awk à une multitude de possibilités permettant de former des modèles de sélection de ligne. On distingue quatre formes possibles:

- ✓ critère vide
- ✓ critère fixe
- ✓ comparaison
- ✓ critère de recherche

a- critère vide: utilisé à chaque fois qu'on souhaite appliquer une action à l'ensemble des lignes. Il suffit d'omettre le critère devant l'accolade ouverte de l'action.

b- critère fixe: Il y a deux critère **BEGIN** et **END**.

Les actions liées à ces critères sont exécutés avant la lecture de la première ligne ou après la lecture de la dernière ligne du fichier.

Exemple:

```
$awk  '
        BEGIN   {
                print  "début de traitement"
                  }
        { print    $1,$3  }
        END    {
                print  "fin de traitement"
                }
                ' fich1
```

c- comparaison: Un critère peut contenir une expression, donc des opérations de comparaison. (les mêmes opérateurs que le langage C.)

Exemple:

```
$awk ' $2>=5    {print $1,$2}' fich1
```

d- critère de recherche: Pour la recherche, awk utilise des métacaractères. Ces métacaractères sont les mêmes que celles de grep, mais placés entre backslash.

Exemple:

```
$awk  ' /[0-9]/1[0-9]/   {print } '  fich1
```

IV-4/ Les structures de contrôle de awk

awk dispose d'une série de structures de contrôle concernant:

✓ les décisions

✓ les boucles

✓ les sauts contrôlés.

a- les décisions:(le **if**)

même syntaxe que pour le langage C.

Exemple:

```
$awk '
        . . . . . .
        NF==7 {
                if($2=="")
                        {
                        print $1 "message"
                        }
                }
        . . . . . .
        ' fichier
```

b- les boucles: trois boucles sont possibles: ***while***, ***do-while*** *et* ***for***

même syntaxe que pour le langage C.

Exemples:

✓ utilisation de while:

```
$awk '
        {
        i=NF
        while (i>0)
                {
                print $i
                i=i-1           (ou i--)
                }
        } ' fichier
```

✓ utilisation de for

```
$awk '
        {
        for(i=NF;i>0;i--)
          {
              print $i
          }
        } ' fichier
```

c- les sauts contrôlés:

✓ Pour le contrôle des boucles: on utilise les instruction break et continue (comme pour le langage C.

✓ Pour le contrôle du programme : on utilise

next :Demande à awk d'interrompre le traitement de la ligne active et de lire immédiatement la ligne suivante.

exit :Termine le programme.

Attention !!! Dans tous les cas, les actions de la critère fixe END seront exécutés et le programme ne sera abandonné qu'après.

V/ CONCLUSION