Système d'exploitation
Les expressions régulières

IUT G.T.R

1ère Année

Sylvain MERCHEZ

PLAN DU COURS

Les expressions régulières : Introduction

Une expression régulière (en anglais Regular Expression) sert à identifier une chaîne de caractères répondant à un certain critère (par exemple une chaîne contenant des lettres minuscules uniquement).

=> L'avantage d'une expression régulière est qu'avec une seule commande on peut réaliser un grand nombre de tâches qui seraient fastidieuses à faire avec des commandes UNIX classiques

Les commandes ed, vi, ex, sed, awk, expr et grep utilisent les expressions régulières.

Les méta-caractères

.	un caractère quelconque
^	début de ligne
$	fin de ligne
\<	début d'un mot
\>	fin d'un mot
\	considère littéralement le caractère suivant
-	spécifie un intervalle : A-Z
x*	zéro ou plus d'occurrences du caractère x
x+	une ou plus occurrences du caractère x
x?	une occurrence unique du caractère x
[...]	plage de caractères permis
[^...]	plage de caractères interdits
\{n\}	pour définir le nombre de répétitions n du caractère placé devant
\(...\)	désigne une sous chaîne
\entier	désigne le numéro de sous chaîne

Les expressions régulières : les métacaratères (exemples 1)

[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 ^
^abc	ligne commençant par abc
abc$	ligne finissant par abc
^$	ligne vide
a*	0 ou n fois a
aa*	au moins un a
/^[0-9][0-9]*$/	ligne qui ne contient que des chiffres

Les expressions régulières : les métacaratères (exemples 2)

[a-zA-Z0-9]	caractère alphanumérique
[-^[]	ensemble de 3 caractères : -, ^,[
^[a-z]*$	les lignes qui contiennent uniquement des lettres minuscules
^[a-z]\{3\}[^a-z]\{3\}	les lignes qui commencent par 3 lettres minuscules et finissent par 3 caractères non lettre minuscule
^\(.*\)\1\1$	désigne les lignes qui sont obtenues par double concaténation d'un mot à lui même (mots triples)

PLAN DU COURS

- Les expressions régulières
  - Introduction
  - Les méta-caractères
- Les commandes administrateur
  - find
  - grep
  - sed
  - awk

Find

Cette commande permet de rechercher les fichiers qui répondent aux critères donnés :

- Syntaxe : find chemin expression
  - -atime n : dernier accès n*24h
  - -cmin n : dernière modif n minutes
  - -empty : fichier vide
  - -type type : spécifie le type de fichiers
  - -group group
  - -gid GID
  - -user utilisateur
  - -name motif

Find

- L 'évaluation de l 'expression est faite de gauche à droite et considère des « ET » entre les éléments
- il est possible de considérer des « OU »
  - Exemple :
    - find . -name nom_fichier
    - find / -name [^a]

PLAN DU COURS

- Les expressions régulières
  - Introduction
  - Les méta-caractères
- Les commandes administrateur
  - find
  - grep
  - sed
  - awk

Grep

- Cette commande permet de rechercher une expression à l 'intérieur des fichiers donnés :
  - Syntaxe : grep motif fichier
    - -l : indique le nom du fichier
    - -n : ajoute le numéro de la ligne
    - -q : affiche rien
    - -s : n 'affiche pas les messages d 'erreur
    - -w : recherche le mot complet

PLAN DU COURS

- Les expressions régulières
  - Introduction
  - Les méta-caractères
- Les commandes administrateur
  - find
  - grep
  - sed
  - awk

La commande sed

sed est un éditeur ligne non interactif

- il lit les lignes d'un fichier une à une (ou provenant de l'entrée standard)
- il applique un certain nombre de commandes d'édition et renvoie les lignes résultantes sur la sortie standard.
- il ne modifie pas le fichier traité, il écrit tout sur la sortie standard.
- Les Options :
  -e permet de spécifier les commandes à appliquer
  -f lit les commandes depuis un fichier

La commande sed

Exemples :

- sed -e "s/[Ff]raise/FRAISE/g" fichier
  → substitue toutes les chaînes Fraise ou fraise par FRAISE
- sed "20,30d" fichier
  → supprime les lignes 20 à 30 du fichier fichier
- sed "/begin/d" fichier
  → supprime les lignes contenant la chaîne begin
- sed -e "s/\([0-9][0-9]*\)/aa\1aa/" fichier
  → La sous expression (sous chaîne) \([0-9][0-9]*\) désigne un ou plusieurs chiffres, chacun sera entouré des caractères aa. La chaîne to2to deviendra toaa2aato.

La commande sed

Exemples : Les commandes d'insertions a,i

```
 sed -f fich_commandes /etc/passwd
----------\
    ----------
LOGIN USER          \       LOGIN           USER
----------\
    ----------
s/:/!/                      root            Operator
s/:/-/                      jd              Jean Dupond
s/:/-/                      vm              Vincent Martin
s/:/!/
s/!.*!/ /
s/:.*//
```


La commande sed : les fonctions

- p : affiche la ligne sélectionnée sur la sortie standard.
- l : affiche la ligne sélectionnée sur la sortie standard avec en plus les caractères de contrôles en clair avec leur code ASCII (deux chiffres en octal).
- = : donne le numéro de la ligne sélectionnée sur la sortie standard.
- q : termine l'exécution de sed, la ligne en cours de traitement est affichée sur la sortie
- r : lit le contenu d'un fichier et écrit le contenu sur la sortie standard.
- w : écrit la ligne sélectionnée dans un fichier.
- a : place un texte après la ligne sélectionnée.
- i : place un texte avant la ligne sélectionnée.

PLAN DU COURS

- Les expressions régulières
  - Introduction
  - Les méta-caractères
- Les commandes administrateur
  - find
  - grep
  - sed
  - awk

La commande awk : présentation

awk (Aho, Weinberger et Kernighan) est une commande très puissante, c'est un langage de programmation qui permet **l'écriture des filtres.**

Elle permet :

- une recherche de chaînes et l'exécution d'actions sur les lignes sélectionnées.
- de récupérer de l'information, générer des rapports, transformer des données.

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 commande awk : syntaxe

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

ou

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

Les arguments :

- -F doit être suivi du séparateur de champ (-F: pour un ":" comme séparateur de champ).
- -f suivi du nom du fichier de configuration de awk.
- -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.

La commande awk : principe

Un **enregistrement** est une chaîne de caractères séparée par un retour chariot.

Un **champs** est une chaîne de caractères séparée par un espace (ou par le caractère spécifié par l'option -F)

La variable **NF** contient le nombre de champs de l'enregistrement courant. On accède à chaque champs par la variable $1, $2, ... $NF. $0 correspond à l'enregistrement complet.

La commande awk : premier exemple

```
~> cat /tmp/passwd
kmouhri:x:1410:130:Karim Mouhri:/user1/info1/groupe2/kmouhri:/bin/tcsh
djiar:x:1411:130:Hanane DJIAR:/user1/info1/groupe3/djiar:/bin/tcsh
slhomme:x:1412:130:Sylvain LHOMME:/user1/info1/groupe1/slhomme:/bin/tcsh
```

$1 **séparateur** **enregistrement**

```
~> awk -F ":" '{ print $5,$6 }' /tmp/passwd
Karim Mouhri
Hanane DJIAR
Sylvain LHOMME
```


La commande awk : forme générale d'un programme

Un programme awk est divisé en 3 sections :

- La section initiale
  BEGIN { actions initiales }
  Les actions sont exécutées avant l'examen de la première ligne du fichier
- Le corps est constitué d'une suite de couples de la forme
  motif1 { action1 }
  motif2 { action2 } ....
- La section terminale
  END { actions finales }
  Les actions sont exécutées après l'examen de la dernière ligne du fichier

La commande awk : mécanisme d'exécution d'un programme awk

Le mécanisme d'exécution est le suivant :

Exécuter les actions de la section initiale
Tant que la fin du dernier fichier n'est pas atteinte
  lire l'enregistrement suivant
  pour chaque motif vérifié par l'enregistrement
  exécuter les actions correspondantes à ce motif
fintq
Exécuter les actions de la section terminale

La commande awk : syntaxe du motif

Le motif peut être :

- une expression régulière
- une expression BEGIN ou END
- une expression de comparaison: <, <=, == , !=, >=,
- 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

La commande awk : exemples de motif

'/[A-Z]/'	lignes avec une majuscule
' /^[a-z]+$/ '	un mot constitué uniquement de minuscules
'$2 ~ /^[ABC]/'	2ème champs commençant par A, B ou C
'$3 !~ /^0/'	3ème champs ne commence pas par 0
'NR%2==1'	lignes impaires
'NR==3,NR==6'	de ligne 3 à 6

La commande awk : exemple d'un programme

```
~> cat /tmp/group
intranet:x:502:lecuppre,lenarczy,leu,wolny,zagrodni
web_bd:x:503:brunet,leroyj,martiaux,wattiaux
aei:x:504:forcinit,lefranco,loison,reschke,saisf

~> awk 'BEGIN { print "Vérification du fichier /tmp/group";
            print "le groupe 502 s'appelle t-il bien intranet ? " ;
            FS=":"}
      $1 == " intranet" && $3 ==502 { print "groupe "$1" a le GID "$3" !" }
      END   { print "Fin" }
' /tmp/group
```

Résultat :

```
Vérification du fichier /tmp/group
le groupe 502 s'appelle t-il bien intranet ?
groupe intranet a le GID 502 !
Fin
```


La commande awk : les variables

Les propriétés d'une variables :

- Le nom des variables est formé de lettres, de chiffres (sauf le premier caractère).
- Pas d'initialisation des variables. Par défaut :
  - Égale à 0 si c'est une variable numérique
  - Égale à une chaîne, si c'est une chaîne

Il existe 3 types de variables :

- **les variables systèmes** : non modifiables donnent des informations sur le déroulement du programme.
- **les variables utilisateurs** : définies par l'utilisateur.
- **les variables de champs** : désignent les champs d'un enregistrement

La commande awk : les variables prédéfinies

ARGC	Nombre d'arguments de la ligne de commande
ARGV	tableau des arguments de la ligne de commande
ENVIRON	tableau contenant les variables de l'environnement courant
FILENAME	nom du fichier sur lequel on applique les commandes
FNR	Nombre d'enregistrements du fichier
FS	séparateur de champs en entrée
NF	nombre de champs de l'enregistrement courant
NR	nombre d'enregistrements déjà lus
OFMT	format de sortie des nombres
OFS	séparateur de champs pour la sortie
ORS	séparateur d'enregistrement pour la sortie
RLENGTH	longueur de la chaîne trouvée
RS	séparateur d'enregistrement en entrée
RSTART	début de la chaîne trouvée
SUBSEP	séparateur de subscript

La commande awk : les variables de champs

Une variable champ commence par $ et a les mêmes propriétés que les autres variables.

- $0 : indique l'enregistrement
- $1 : le premier champs de l'enregistrement

….

- $NF : le dernier champs de l'enregistrement

Remarques :

- Le signe $ peut être suivi par une variable (ex **$(i+1)**)
- 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.
- Le séparateur est celui défini par OFS.

La commande awk : les actions

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 et opérateurs,
- fonctions d'affichage.

La commande awk : les fonctions numériques

Nom	signification
atan2(y,x)	arctangente de 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)	reinitialiser le générateur de nombre aléatoire
Et ..	
systime()	date courante en nombre de secondes p/r à 1/1/1970

La commande awk : les fonctions sur les chaînes de caractères

- s et t sont des chaînes de caractères
- r est une expression régulière
- i et n sont des entiers

Nom **signification**

- **gsub(r,s,t)** sur la chaîne t, remplace toutes les occurrences de r par s. r peut être une chaîne ou une expression régulière.

exemples :

gsub ("user","utilisateur",$0) → remplace dans l'enregistrement courant toutes les chaînes user par utilisateur.

gsub (/[abc]/,"??",$1) → remplace dans le 1er champs les caractères a, b ou c par la chaîne ??.

La commande awk : les chaînes de caractères

- **index(s,t)** retourne la position la plus à gauche de la chaîne t dans la chaîne s

  exemple : **n=index("patate","ta")** → n = 3

- **length(s)** retourne la longueur de la chaîne s

  exemple : **long=length("iut de lens")** → long = 11

- **match(s,r)** retourne l'index où s correspond à r et positionne RSTART et RLENGTH. r est une chaîne ou une expression régulière.

  exemple :

  n=match("PX1235D",/[0-9]+/) → n=3, RSTART=3 et RLENGTH=4

La commande awk : les chaînes de caractères

- **printf(format,valeur)** permet d'envoyer des affichages (sorties) formatées, la syntaxe est la même qu'en C

  exemple : **printf("La ligne est %s",$0)**

- **split(s,t,fs)** scinde la chaîne a dans un tableau t, fs est le séparateur de champ. Le résultat est le nombre d'éléments dans t.

  exemple :

  n=split("iut de lens",t," ") → t[1]="iut", t[2]="de", t[3]="lens", n=3

- **sprintf(fmt,liste expressions)**

  retourne la liste des expressions formatée suivant fmt

  exemple :

  st=sprintf("%d lens cedex",i) → st="62307 lens cedex" , avec i=62307

La commande awk : les chaînes de caractères

- **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

  exemple : machaine=substr("iut de lens",5,4) → machaine="de l"

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

  exemple :

  commande=sprintf("ps | wc -l")

  system(commande) → Exécution de la commande UNIX "ps |wc -l"

La commande awk : les chaînes de caractères

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

  exemple :

  ch=tolower("IUT DE LENS") → ch="iut de lens"

- **toupper(chaine)** retourne la chaîne de caractères convertie en majuscule

  exemple :

  ch="iut de lens"

  ch2=toupper(substr(ch2,1,1))substr(ch,2,10) → ch2="Iut de lens"

La commande awk : les fonctions de contrôle

On distingue 3 types de fonctions de contrôle :

- Alternative : if
- Itératives : while, for, do-while
- Les sauts : next, exit, continue, break

La commande awk : fonction alternative

La syntaxe est la suivante :

if (condition)
{instructions1}
else
{instructions2}

Exemple :

```
if ($2=="")
{ print $1 "n'a pas de mot de passe"}
else
{print $1 " a un mot de passe"}
```


La commande awk : fonctions itératives (while)

La syntaxe du while est la suivante :

while (condition)
{instructions}

Exemple

```
i=1
while (i<=NF) {
    print $i
    i++
}
```


La commande awk : fonctions itératives (do-while)

La syntaxe du do-while est la suivante :

```
do {
    instructions
} while (condition)
```

Exemple

```
i=1
do {
    print $i
    i++
} while (i<=NF)
```

La différence avec while, est que la boucle est exécutée au moins une fois.

La commande awk : fonctions itératives (for)

La syntaxe du for est la suivante :

for (initialisation; condition; incrémentation) {
instructions
}

ou

for (var_boucle in tableau) {
instructions
}

Exemples :

```
for (i=1;i<=NF;i++) { print $i }
tab[1]="patate"; tab[2]="courgette";tab[3]="poivron
for (cpt in tab) {
        print "legume :", tab[cpt]
}
```


La commande awk : les sauts

- **break** : permet de sortir d'une boucle courante,
- **continue** : poursuite à l'itération suivante de la boucle.
- **next** : permet d'interrompre le traitement sur l'enregistrement courant et de passer au suivant.
- **exit** : permet d'abandonner la commande awk, les instructions suivant END sont exécutées (s'il y en a)

La commande awk : les fonctions d'affectation

Les opérateurs d'affectation : =, +=, -=, *=, /=, %=, et ^=.

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". La variable var continent "toto est grand"
- var=var-valeur et var-=valeur sont des expressions équivalentes
- var=var+1 et var+=valeur sont des expressions équivalentes

La commande awk : les opérateurs

Les opérateurs suivants sont classés par ordre de priorité :

- (…)
- $ opérateur de référence d'un champs
- ++, -- incrémenter, décrémenter
- ^ opérateur de puissance
- +, - et ! opérateurs unaires
- *, / et % multiplier, diviser et modulo
- +, - additionner, soustraire
- <espace> opérateur de concaténation de chaîne
- <>, <=, >=, != et == opérateurs relationnels
- ~, ~! égalité, différence des expressions régulières
- in membre d'un tableau
- &&, || opérateurs logiques
- ?: expression conditionnelle. *cond ? inst_alors: inst_sinon*

La commande awk : fonction d'affichage print

La syntaxe de la fonction print :

print exp, exp ou print (exp , exp)

ou

printf format , exp, exp ou printf (format, exp , exp)

Un format est une chaine de caractères et des constructeurs commençant par %.

Les caractères de format sont :

- %c caractère
- %d ou %i nombre entier
- %e ou %E nombre réel au format de –d.dddddd[+-]dd
- %f nombre réel au format de –d.dddddd
- %o nombre octal

La commande awk : fonction d'affichage print

- %u nombre entier non signé
- %s chaîne de caractères
- %x nombre hexadécimal
- %% caractère %

Un paramètre optionnel peut être ajouté entre le caractère % et le caractère de format.

Les paramètres optionnels sont :

- **-** expression justifiée à gauche
- <espace> conversion numérique.
- **largeur** largeur d'affichage
- **.précision** longueur maximale d'une chaîne de caractères ou du nombre de décimales

La commande awk : exemples de la fonction print

awk ' { print NR " :" $0 } ' fic > fichier.numerote

→ le fichier fich.numerote contient le fichier fic avec les lignes numérotées

awk ' { printf "%3d : %s " , NR , $0 } ' fic >fichier.numerote

→ le fichier fich.numerote contient le fichier fic avec les lignes numérotées sur 3 caractères

La commande awk : fonctions utilisateurs

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
}
```

La commande awk : les tableaux

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.

Les tableaux sont unidimensionnels

Un tableau n'a pas besoin d'être déclaré

La valeur initiale des élément est une chaîne vide ou zéro

On dispose de la fonction delete pour supprimer un tableau (**delete tab**). Pour supprimer un élément de tableau on tapera **delete tab[index].**

Exemple :

var=1

La commande awk : les tableaux

Exemple :

```
var=1
for (i=1;i<=NF;i++) {
mon_tab[i]=var++
}
```


La commande awk : 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" }
```

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.

La commande awk : 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"