UNIVERSITÉ DE LORRAINE | FST FACULTÉ DES SCIENCES ET TECHNOLOGIES Département Informatique

L2 informatique

Outils système

Traitement de chaînes de caractères

Alexis Scheuer

Université de Lorraine (FST, Dpt info.)

Plan

Plan

Opérateurs et interprétation

Différents opérateurs sont utilisables lors de l'interprétation d'une variable comme une chaîne de caractères :

- pour déterminer la taille de la variable,
- pour en extraire une sous-chaîne,
- pour supprimer des motifs, à droite ou à gauche.

Taille d'une chaîne, sous-chaîne

La taille d'une chaîne de caractères stockée dans une variable s est donnée par `${#s}`.

```
$ echo $PWD
/home/toto
$ echo ${#PWD}
10
```

La sous-chaîne de s constituée des caractères d'indice $\geq i$ ($i \geq 0$) est donnée par `${s:i}`, celle contenant n caractères par `${s:i:n}`.

```
$ s="abcdefgh"
$ echo ${s:2}
cdefgh
$ echo ${s:2:3}
cde
```

Suppression à droite

Pour supprimer à droite dans une chaîne *s* la chaîne la plus courte correspondant à un motif *m*, utilisez `${s%m}`.

```
$ s="rep/dudule.tar.gz"
$ echo ${s%.*}
rep/dudule.tar
```

Pour supprimer à droite dans une chaîne *s* la chaîne la plus longue correspondant à un motif *m*, utilisez `${s%%m}`.

```
$ s="rep/dudule.tar.gz"
$ echo ${s%%.*}
rep/dudule
```

Suppression à gauche

Pour supprimer à gauche dans une chaîne *s* la chaîne la plus courte correspondant à un motif *m*, utilisez `${s#m}`.

```
$ s="rep/dudule.tar.gz"
$ echo ${s#*du}
dule.tar.gz
```

Pour supprimer à gauche dans une chaîne *s* la chaîne la plus longue correspondant à un motif *m*, utilisez `${s##m}`.

```
$ s="rep/dudule.tar.gz"
$ echo ${s##*du}
le.tar.gz
```

Plan

Analyse d'un fichier

La commande `file` donne des informations sur le format d'un fichier.

```
$ file operateurs.tex
operateurs.tex: LaTeX document, UTF-8 Unicode text
```

La commande `wc` compte le nombre de caractères (option `-c`), de mot (`-w`) et de ligne (`-l`) du fichier fourni en paramètre.

```
$ wc operateurs.tex
99 284 3469 operateurs.tex
```

$\rightsquigarrow$ 99 lignes, 284 mots et 3469 octets.

Traitement d'un chemin

La commande `dirname` extrait le nom du répertoire final d'un chemin fourni.

```
$ dirname `which roslaunch`
/opt/ros/kinetic/bin
```

La commande `basename` extrait le nom du fichier d'un chemin fourni.

```
$ basename /etc/NetworkManager/system-connections/*aine
Personnels Univ-Lorraine
```

Tri du contenu de fichiers

La commande `sort` concatène le contenu des fichiers passés en paramètre, trie les lignes et affiche le résultat. Ses options sont (entre autres) :

- `-n` tri de valeurs numériques ;
- `-r` trier dans l'ordre inverse ;
- `-t` choix d'un séparateur (entre les champs) autre que l'espace ;
- `-k` tri à partir d'une clé jusqu'à éventuellement une autre après une virgule, chaque clé étant un numéro de champ (≥ 1), éventuellement suivi d'un point et d'un numéro de caractère (bis), et d'options de tri (`n`, `r`, ...) ;

...

Tri de contenu : exemple

```
$ cat inventaire.csv
fruit citrons 12
légume citrouilles 3
fruit ananas 10
légume carottes 35
```

```
$ sort inventaire.csv
fruit ananas 10
fruit citrons 12
légume carottes 35
légume citrouilles 3
```

```
$ sort -k3n inventaire.csv
légume citrouilles 3
fruit ananas 10
fruit citrons 12
légume carottes 35
```

```
$ sort -k2r inventaire.csv
légume citrouilles 3
fruit citrons 12
légume carottes 35
fruit ananas 10
```

Plan

Expressions régulières : définition

Définition : Expressions régulières

Une expression régulière est une formule décrivant un motif (ou un modèle, en anglais "*pattern*") que l'on souhaite retrouver dans un ensemble de chaînes de caractères.

Les expressions régulières, dénommées par le raccourci "*regexp*" (pour "*regular expressions*"), sont utilisées par de plusieurs commandes dont on parlera plus loin : `grep`, `find`, `sed` et `awk`.

Expressions régulières : formules (1/2)

Voici quelques exemples de formules définissant une expression régulière, et leur signification :

- `^` en début de motif, représente le début de la ligne ;
- `$` en fin de motif, représente la fin de la ligne ;
- `.` représente un seul caractère (**pour certains outils**) ;
- $s?$ représente 0 ou 1 fois le symbole s ;
- $s*$ représente 0 ou plusieurs fois le symbole s ;
- $s+$ représente 1 ou plusieurs fois le symbole s ;
- $s\{n\}$ représente exactement n fois le symbole s ;
- $[\mathscr{S}]$ représente un des caractères présents dans l'ensemble $\mathscr{S}$;
- $[\hat{}\mathscr{S}]$ représente un des caractères qui n'est pas présent dans l'ensemble $\mathscr{S}$.

Expressions régulières : exemples (1/2)

Si on considère l'ensemble de chaînes de caractères suivant :

```
image.bmp image_120.jpg image_120_old.jpg
Image_125.jpg Image_222.jpg Image_1658.jpg image
```

les expressions régulières qui suivent auront pour résultat :

`Image_.{3}.jpg`	→	`Image_125.jpg` et `Image_222.jpg`
`^[Ii]mage.*`	→	tout (commence par `I` ou `i`, suivi de `mage` et de n'importe quoi)
`^[^I].*` ou `^i.*`	→	`image.bmp`, `image_120.jpg`, `image_120_old.jpg` et `image` (ne commence pas par `I`, ou commence par `i`)

Le sens de `.*` n'est pas le même que pour la commande `ls`.

Expressions régulières : formules (2/2)

L'ensemble $\mathscr{S}$ des caractères qui doivent être présents ou absents du motif peuvent être :

Liste	Équivalent	Description
a-z	[:lower:]	lettres de a à z, minuscules
A-Z	[:upper:]	lettres de a à z, majuscules
a-zA-Z	[:alpha:]	lettres
0-9	[:digit:]	chiffres de 0 à 9
a-zA-Z0-9	[:alnum:]	lettres et chiffres
	[:space:]	caractères d'espacement
	[:print:]	Caractères imprimables
	[:cntrl:]	Caractères de contrôle
	[:punct:]	Caractères de ponctuation

Dans la liste, tout caractère peut être utilisé : un point, une virgule, \ (double ?), un espace, ..., et même ^ (sauf au début), un tiret (en dernier) ou] (en premier ou après un \).

Expressions régulières : exemples (2/2)

Motif	Signification
`^[A-M]`	commence par une majuscule entre A et M
`[a-zA-Z0-9]`	équivalent à `[[:alnum:]]`
`[^a-zA-Z0-9]`	équivalent à `[^[:alnum:]]`
`[home]`	chaîne contenant un h, un o, un m ou un e, par exemple maison
`[-^[]`	chaîne contenant un tiret, un ^ ou un [
`^[a-z]+$`	chaîne composée uniquement de minuscules avec au moins une lettre
`^[a-z]{3}[^a-z]{5}`	chaîne commençant par 3 minuscules suivies de 5 lettres non minuscules
`^[.+][a-z]?$`	chaîne composée uniquement d'un . ou +, suivi ou non par une lettre en minuscule
`^([^:]*):\1:\1$`	chaîne de la forme m:m:m, avec m n'importe quel motif ne contenant pas :

Plan

Filtrage du contenu de fichiers

La commande `grep` recherche le motif donné en premier paramètre dans les fichiers fournis ensuite, et affiche sur la sortie standard l'ensemble des lignes contenant le motif.

Avec l'option `-F` ou la commande `fgrep`, le motif est considéré comme une chaîne de caractère.

Avec l'option `-E` ou la commande `egrep`, le motif est une expression régulière.

Avec l'option `-r` ou la commande `rgrep`, les répertoires donnés en paramètres (après le motif) sont parcourus récursivement.

Filtrage : options complémentaires

Les autres options de la commande `grep` sont :

- `-i` pas de distinctions entre minuscule et majuscule ;
- `-n` commence chaque ligne trouvée par son numéro ;
- `-v` affiche les lignes qui ne valident pas la condition ;
- `-c` affiche le nombre de lignes trouvées (à la place des lignes) ;
- `-l` affiche le nom des fichiers contenant le motif (à la place des lignes) ;
- `-L` affiche le nom des fichiers ne contenant pas le motif (à la place des lignes) ;

Filtrage du contenu : exemple

```
$ cat inventaire.csv
fruit citrons 12
légume citrouilles 3
fruit ananas 10
légume carottes 35
```

```
$ grep -n ro inventaire.csv
1:fruit citrons 12
2:légume citrouilles 3
4:légume carottes 35
```

```
$ grep -c fr inventaire.csv
2
```

```
$ egrep "o.*e" inventaire.csv
légume citrouilles 3
légume carottes 35
```

Filtrage du contenu : valeur de retour

Comme la majorité des commandes, la commande `grep` retourne un code donnant le résultat de sa recherche :

0. une ou plusieurs lignes ont été trouvée(s) ;
1. aucune ligne n'a été trouvée ;
2. erreur de syntaxe ou d'accès aux fichiers.

Utilisation possible :

```
$ grep -i TR inventaire.csv &> /dev/null ;
if (($? == "0")); then echo OK; else echo KO;
fi
OK
```

Recherche d'un nom de fichier

La commande `find` [*r*] *c* [*a*] recherche un fichier respectant une condition *c* dans le répertoire *r* et exécute l'action *a*.

Si aucun répertoire n'est donné, le répertoire courant est considéré.

Les actions possibles sont :

`-print` (défaut) affiche le nom du fichier ;

`-exec cmd` exécute la commande donnée, dans laquelle le nom du fichier est remplacé par `{}` ;

`-ok cmd` exécute la commande après vérification.

...

Recherche : condition

La condition peut être :

`-name motif` le nom du fichier doit correspondre au motif (format `ls`) ;

`-regex exp` le nom du fichier doit correspondre à l'expression régulière ;

`-size` *n* la taille du fichier doit correspondre à *n*, l'unité étant indiquée par le suffixe (`b` = 512 octets, défaut, `c` = octet, `k` = ko, `M` = Mo, `G` = Go) ;

`-newer fch` le fichier soit être plus récent que le fichier fch (modification) ;

`-atime` [+|-]*n* la date de la dernière consultation est de [+ ou - de] n jours ;

...

Recherche : exemples

La commande

```
$ find -name "*.c" -mtime 3
```

recherche dans le répertoire courant les fichiers C modifiés il y a 3 jours.

La commande

```
$ find -regex ".*D.*a.*" -exec grep -l "test" {} \;
```

recherche dans le répertoire courant les fichiers dont le nom contient un D suivi d'un a et contenant le mot test (ne fonctionne pas sans les `.*` aux extrémités).

Transformation d'un contenu

La commande `sed` permet différentes transformation du contenu d'un fichier, dont la suppression et le remplacement, et affiche le résultat.

La commande `sed "1,10d"` supprime les lignes 1 à 10 du fichier donné ensuite.

La commande `sed "/exp/d"` supprime les lignes du fichier donné ensuite qui correspondent à l'expression régulière exp.

La commande `sed "/exp/!d"` supprime les lignes du fichier donné ensuite qui ne correspondent pas à l'expression régulière exp.

Remplacement d'un contenu

La commande `sed "s/exp1/exp2/"` remplace dans le fichier donné ensuite l'expression régulière exp1 par la chaîne exp2, dans laquelle & désigne le motif et les caractères \1 à \9 d'éventuelles sous-expressions, et affiche le résultat.

La commande `sed "s/ficheir/fichier/g"` remplace toutes les occurences (option `g` à la fin) de « ficheir » par « fichier ».

La commande `sed "s/\([^- ]*\)-\([^- ]*\)/\2-\1/"` remplace le premier mot de chaque ligne contenant un tiret par le mot dont les parties séparées par le tiret ont été échangées.

Suppression de contenu

La commande `cut` supprime ou filtre une partie des lignes du fichier donné ensuite.

La commande `cut -c1-3` affiche les caractères 1 à 3 des lignes du fichier donné ensuite.

La commande `cut -d: -f6-9` découpe les lignes du fichier donné ensuite en champs séparés par `:` et affiche les champs 6 à 9.

Plan

awk : principe

La commande awk :

- permet un traitement avancé de chaînes de caractères ;
- manipule des *enregistrements* (*records*) séparés en *champs* (*fields*) ;
- est un interpréteur (écriture de scripts).

```
$ cat inventaire.csv
fruit citrons 12
légume citrouilles 3
fruit ananas 10
légume carottes 35
```

```
$ awk -f liste.awk inv...
Ma liste est la suivante :
* 12 citrons (fruits) ;
* 3 citrouilles (légumes) ;
* 10 ananas (fruits) ;
* 35 carottes (légumes) ;
Soit 60 produits.
```

awk : appel

```
awk [-F sep] [-v var=val] ( (-f script)+ | 'prog') (data)+
```

- `-F` modifie le séparateur de champs (espace par défaut) ;
- `-v` définit une variable transmise au programme ;
- `-f` spécifie le fichier de script à lire.

Le traitement est décomposé en :

enregistrement chaîne de caractères délimitée par un séparateur d'enreg. (par défaut, un retour chariot $\rightsquigarrow$ une ligne) ;

champ chaîne de caractères délimitée par un séparateur de champ (par défaut, un espace $\rightsquigarrow$ un mot) ;

- désignés par `$1`, ... (`$NF` = dernier) ;
- `$0` = enregistrement courant en entier.

awk : exemple d'appel

```
$ cat inventaire.csv
fruit citrons 12
légume citrouilles 3
fruit ananas 10
légume carottes 35
```

```
$ awk '{printf("* %d %s (%ss) ;\n", $3, $2, $1)}' \
   inventaire.csv
* 12 citrons (fruits) ;
* 3 citrouilles (légumes) ;
* 10 ananas (fruits) ;
* 35 carottes (légumes) ;
```

awk : blocs d'un programme

Un programme ou un script awk se compose d'au plus trois blocs :

1. une phase (optionnelle) de pré-traitement de la forme `BEGIN{ ... }` ;
2. le corps du traitement `{ ... }` ;
3. une phase (optionnelle) de post-traitement de la forme `END{ ... }`.

```
$ cat liste.awk
BEGIN{ nb = 0; printf("Ma liste est la suivante :\n") }
{ printf("* %d %s (%ss) ;\n", $3, $2, $1); nb += $3 }
END{ printf("Soit %d produits.\n", nb) }
```

awk : variables

- Typage implicite, chaîne de caractères ou réel
  `var ""` $\rightsquigarrow$ chaîne, `var + 0` $\rightsquigarrow$ réel
- Valeur par défaut ("" ou 0),
  initialisation préférable (dans `BEGIN` ou par l'option `-v`)
- Déréférencement automatique ($\sim$ C) :
  valeur ou variable, selon l'utilisation
- Opérateurs ($\sim$ C) :

numériques	+, -, *, /, %, ^ (ou **)
affectation	=, +=, -=, *=, /=, %=, ^= (ou **=), ++, --
booléens	! (négation), && (et), || (ou)
comparaison	==, !=, <, >, <=, >=
exp. reg.	$\sim$ (correspondance), !$\sim$

`awk` : variables prédéfinies

Variable	Signification	Par défaut
FILENAME	Nom du dernier fichier traité	
FS/OFS	Séparateur de champs (E/S)	" "
RS/ORS	Séparateur d'enregistrements (")	"\n"
NF	Nombre de champs de l'enregistrement courant	
NR	Nombre d'enregistrements déjà lu	
FNR	Nombre d'enregistrements du dernier fichier traité	
OFMT	Format de sortie des nombres	"%.6g"

Et beaucoup d'autres (*cf* `man`).

awk : structures de contrôle

Très semblables à celles du C :

- `if (condition) instruction [else instruction]`
- `switch (condition) {`
  `(case valeur | exp. reg. : instructions)+`
  `[default: instruction] }`
- `break`, `continue`, `exit`
- `while (condition) instruction`
- `do instructions while (condition)`
- `for (expr1; expr2; expr3) instruction`
- `for (var in tableau) instruction`

awk : exemples

```
$ cat inventaire.csv
fruit citrons 12
légume citrouilles 3
fruit ananas 10
légume carottes 35
```

```
$ awk '{if (NR > 2) printf("%s\n", $0)}' inventaire.csv
fruit ananas 10
légume carottes 35
```

```
$ awk '{if ($1 ~ /e$/) printf("%s\n", $0)}' inv...
légume citrouilles 3
légume carottes 35
```

awk : tableau

Les tableaux sont indexés par une chaîne de caractères. Ils sont donc associatif, et correspondent plutôt à une *map*. L'indice peut être une variable simple (`t[i]`) ou une liste (`t[i, j, ...]`). Cela permet dans le second cas de créer un tableau multidimensionnel. On peut aussi multiplier les indices (`t[i][j]...`).

Les tableaux sont dynamiques, comme les variables. L'ajout d'éléments se fait par l'affectation, la suppression (d'un élément ou du tableau) avec l'opérateur `delete`.

L'opérateur `in` permet de parcourir un tableau (boucle `for`) ou de tester l'existence d'un indice dans ce tableau (`if`).

awk : fonctions sur les chaînes

Fonctions	Description
`sprintf(fmt,...)`	Retourne la liste des expressions formatée suivant fmt.
`length(s)`	Retourne la longueur de la chaine `s`.
`index(s,t)`	Retourne la position la plus à gauche de la chaine `t` dans la chaine `s`.
`substr(s,i,n)`	Retourne la sous-chaine de `s` commençant en `i` et de taille `n`.
`split(s,a[,fs])`	Découpe `s` dans le tableau `a` avec `fs` comme séparateur, et retourne le nombre de champs.
`match(s,re)`	Retourne l'index où `s` correspond à `re` et modifie `RSTART` et `RLENGTH`.
`sub(re,s,t)`	Remplace dans la chaine `t` la première occurrence de `re` par `s`.
`gsub(re,s,t)`	Comme `sub`, mais pour toutes les occurrences, et retourne leur nombre.

awk : fonctions autres et nouvelles

Fonctions arithmétiques

- valeur entière : `int(x)`)
- valeur aléatoire dans $[0, 1[$: `rand()` ; après initialisation : `srand([x])`
- calculs : `sqrt(x)`, `exp(x)`, `log(x)`, `cos(x)`, `sin(x)`, `atan2(y,x)`

Vérifier si une variable `v` correspond à un tableau : `array(v)`

Définir une fonction :

```
function f(...,    ...) { ...; return ... }
```

les premiers `...` sont les paramètres, les suivants sont des variables locales.

awk : syntaxe simplifiée

- print est une version simplifiée de printf
- les accolades globales et le if peuvent être omis
- expr1,expr2 est valide de l'enregistrement qui vérifie la première expression jusqu'à celui qui vérifie la seconde

```
$ awk '($2 ~ /^a/) {print "ligne",NR,":",$3,$2}' inv...
ligne 3 : 10 ananas
```

```
$ awk 'NR==2, NR==3 {print "ligne",NR,":",$3,$2}' inv...
ligne 2 : 3 citrouilles
ligne 3 : 10 ananas
```

awk : compléments

Plus de détails :

français `www.shellunix.com/awk.html`

anglais
- `man awk`
- `www.gnu.org/software/gawk/manual`