Le système d'exploitation GNU/Linux

Nicolas Burrus <nicolas.burrus@laposte.net>

- Shell avancé
- Commandes avancées

26 Avril 2005

Contenu du cours

Philosophie Unix

Collection d'outils simples :

- Chaque outil fait peu de choses ...
- ... mais il le fait bien
- En combinant les outils de bases, on peut faire des choses compliquées
- Besoins de communication inter-programmes

Shell avancé : entrée / sorties

Chaque processus a :

- Une entrée standard (stdin)
- Une sortie standard (stdout)
- Une sortie d'erreur (stderr)

Le shell peut les modifier avant de lancer un processus :

- Pour rediriger une sortie vers un fichier
- Pour rediriger une sortie vers l'entrée d'un autre programme
- Pour lier une entrée à un fichier
- ...

En pratique

`command > file`
Redirige la sortie de `command` dans le fichier `file`

`command » file`
Ajoute la sortie de `command` à la fin du fichier `file`

`command 2> file`
Redirige la sortie d'erreur de `command` dans le fichier `file`

`command &> file`
Redirige la sortie standard et d'erreur de `command` dans `file`

`command < file`
Utilise le contenu de `file` comme entrée standard pour `command`

`command 2>&1`
Envoie le flux de la sortie d'erreur dans la sortie standard

Communication entre programmes

`command1 | command2`
Envoie la sortie standard de `command1` comme entrée standard de `command2`
Ex: `ls | less`
Ex: `ls /nowhere /bin 2>&1 | more`

`command1 |& command2`
Envoie la sortie standard et la sortie d'erreur de `command1` comme entrée standard de `command2` (zsh seulement)

Séquences de commandes

Chaque commande à un "exit status", un code de sortie :

- 0 ➔ la commande a réussie
- $\neq 0$ ➔ la commande a échoué
- `echo $?` donne le code de sortie de la commande précédente

`command1 ; command2 [; command3] ...`
Enchaîne les commandes quelque soient leurs codes de sortie

`command1 && command2 ...`
Exécute `command2` seulement si `command1` a réussi

`command1 || command2 ...`
Exécute `command2` seulement si `command1` a échoué

Boucles

`for var in list; do commands; done`
Exécute `commands` avec `var` prenant chaque valeur de `list`
Les éléments de `list` sont séparés par des espaces
Ex: `for f in *; do echo $f; cp $f $f.sav; done`
Ex: `for s in "hello" "world"; do echo $s; done`

`while cond; do commands; done`
Exécute `commands` tant que le code de sortie de command est 0

Expressions régulières (regex) [1/2]

Objectif : "matcher" des chaînes de caractères

- Plus puissantes que les wildcards du shell ...
- ... mais légèrement plus complexes
- Utilisées par plusieurs outils de manipulation de texte
- Généralement appliquées ligne par ligne
- But : est ce que la chaîne `s` matche un certain motif ?

Caractères spéciaux

- `.` : n'importe quel caractère
- `(expr)*` : `expr` 0 ou plusieurs fois
- `(expr)?` : `expr` 0 ou une fois
- `(expr)+` : `expr` 1 ou plusieurs fois
- `[adz]` : `a` ou `d` ou `z`
- `[a-e]` : une lettre entre `a` et `e`
- `^` et `$` : début et fin de ligne

Expressions régulières (regex) [2/2]

Exemples de regex :

- `/Hello Mr .*/`
- `/Numéro de sécurité sociale : ([0-9])+/`
- `/.* doit terminer par ça$/`

Attention :

- Différentes variantes
- Parfois, les caractères spéciaux doivent être préfixé d'un backslash
- Syntaxe étendue plus simple

Commandes évoluées [1/2]

`grep` `[-r] [-E] pattern [file1] [file2] ...`
Affiche les lignes matchant `pattern` dans les `fileN` ou dans l'entrée standard
`[-r]` Parcours récursivement les répertoires
`[-E]` Expressions régulières étendues
Ex: `ls / | grep -E 'r/$'`
Ex: `grep -rE network /etc`

`sed` `[-r] -e action [file1] [file2] ...`
Execute l'action `action` sur chaque ligne des `fileN` ou sur l'entrée standard
`[-r]` Expressions régulières étendues
Action usuelle :`s/regex/nouveau-text/`
Remplace la regex matchée par `nouveau-text`
Ex: `sed -re 's/[0-9]+/nonumbers/g'`
Mémoire : `sed -re 's/Number: ([0-9]+)/\1/`

Commandes évoluées [2/2]

`find` `path condition`
Recherche les fichiers dans `path` selon `condition`
Ex : `find / -name 'less'`
Ex : `find /var -type d`

`head` `[-n N] [files]`
Affiche seulement les N premières lignes de `files` ou de l'entrée standard

`tail` `[-n N] [files]`
Affiche seulement les N dernières lignes de `files` ou de l'entrée standard

`wc` `-l [files]`
Affiche le nombre de lignes de `files` ou de l'entrée standard

TP

http://nburrus.objectis.net/teaching/linux/