Programmation III

Unix & programmation shell

Basile Schaeli
Roger D. Hersch
Laboratoire de Systèmes Périphériques (LSP)
EPFL

Programmes Unix

- Quelques programmes connus
  - ls affiche le contenu d'un répertoire
  - rm supprime un fichier
  - cd change de répertoire
  - chmod modifie les permissions d'un fichier ou d'un répertoire
  - cat affiche le contenu de fichiers
- Mais aussi
  - javac/java compilateur et interpréteur Java
  - gcc/g++ compilateurs C et C++
  - make interprète les makefiles
  - lineardict les programmes du cours

Programmes Unix

- Il y a d'autres programmes utiles
  - wc compte les lignes/mots/caractères d'un fichier
  - ssh permet de se connecter sur une autre machine à travers le réseau
  - find recherche des fichiers/répertoires
  - less affiche le contenu d'un fichier
  - ps liste les processus en cours d'exécution
  - top affiche la charge de la machine
  - man affiche le manuel de chaque programme
  - wget télécharge des fichiers via http ou ftp
- Editeurs de texte
  - nano, vi, vim, emacs
- Navigateur web
  - lynx

grep

- Signifie "Global Regular Expression Parser"
- Affiche les lignes qui contiennent un pattern

```
% grep timer wordlist.txt
timer
timers
```

- Wildcard:
  - '.' signifie n'importe que caractère

```
% grep t.m. wordlist.txt
```

retourne 827 mots!

  - "systematic" contient "t.m."
- Caractères spéciaux de début et fin de ligne
  - ^ et $

grep

- Nouvel essai:

```
% grep ^t.m.$ wordlist.txt
tame
tamp
tams
temp
time
tomb
tome
toms
```

Expressions régulières

- L'argument de **`grep`** est une expression régulière (regex)
  - Permet de faire du "pattern matching" ("correspondance à un modèle"?)
- Utilisées partout où l'on doit traiter des chaines de caractères
  - Programmes en ligne de commande
    - grep, sed, awk
  - Recherche dans un éditeur ou dans un système de fichiers
    - find, Emacs, Visual Studio, Word (dans une moindre mesure)
  - Langages de programmation
    - Perl, python, Java (java.util.regex), C et C++ (librairies externes), …
- Il y a évidemment des variantes selon les langages...

Expressions régulières

- Multiplicateurs
  - `*` caractère précédent répété 0 ou plusieurs fois
  - `?` caractère précédent répété 0 ou 1 fois
  - `+` caractère précédent répété 1 ou plusieurs fois
- Exemples

• Mots	• pil*e	• pil?e	• pil+e
pieds	pieds	pieds	pile
pile	pile	pile	piller
piller	piller		

- Pour représenter n'importe quelle chaine: .*
  p.ex: A.*B représente A*n'importe quoi*B

Expressions régulières

- Ensemble de caractères
  - `[]` Un des caractères de l'ensemble: [adh], [a-zA-Z]
  - `[^]` Complémentaire de l'ensemble: [^ab]
  - `|` OU logique entre expressions: ab|ac
- Exemples

• Mots	• [dr]	• [^i]e	• [^piller]
pieds	pieds	pile	pieds
pile	piller	piller	
piller			

- Détails: *man grep*

sed: Stream EDitor

- Effectue des opérations sur un flux (stream)
- Travaille ligne par ligne
- Substitution de chaine de caractères:
  `% sed "s/pattern/remplacement/" fichier.txt`
  - Remplace la première sous-chaine trouvée

  `% sed "s/pattern/remplacement/g" fichier.txt`
  - Remplace toutes les sous-chaines trouvée
- **`Pattern`** est une expression régulière
  - Attention! le multiplicateur * est glouton (*greedy*)!
    `s/A.*C/ab/` transforme 'ABCC' en 'ab' et pas en 'abC'

sed: exemples

```
% cat mots.txt
pieds
pile
piller
% sed s/eds/ive/ mots.txt
pive
pile
piller
```

```
% sed s/./a/ mots.txt
aieds
aile
ailler
% sed s/./a/g mots.txt
aaaaa
aaaa
aaaaaa
```

- Supprimer les balises d'un fichier html

```
% cat fichier.html
<html><body>Texte <b>en gras</b></body></html>
% sed "s/<[^>]*>//g" fichier.html
Texte en gras
```

awk: Aho, Weinberger & Kernighan

- Language de traitement de chaines de caractères
- Utilisation:
  `awk –F`*`fs`*` '`*`programme`*`' fichier.txt`
- Un programme est structuré en trois parties
  - BEGIN { … } exécuté avant la lecture du fichier
  - { … } exécuté sur chaque ligne du fichier
  - END { … } exécuté une fois toutes les lignes lues
- Chaque partie peut contenir des structures de contrôle ou d'affichage (**`print`**)
- Chaque ligne est découpée en champs séparés par un espace/tab (*whitespace*) ou par l'expression régulière *fs*
- Le *n*ème champ est référencé à l'aide de la variable **$n**

awk: exemples

- Sélectionner certains champs uniquement

```
% cat f.txt
Quelques mots dans l'ordre
% awk '{ print $4" "$3" "$4 }'
l'ordre dans l'ordre
```

```
% cat f.html
<body><h1>Mon HTML</h1></body>
% awk '{ print $2","$1 }' f.html
HTML</h1></body>,<body><h1>Mon
```

- Travailler sur les balises html

```
% awk -F"<[^>]*>" '{ print $3 }' f.html
Mon HTML
% awk –F">[^<]*<|[<>]" '{ print $3" "$4 }' f.html
h1 /h1
```

awk: exemples

- Calcule la somme de nombres stockés dans un fichier (un nombre par ligne)

```
% cat nbres.csv
12
34
15
7
% awk '{s+=$1} END {print "Somme: " s}' nbres.csv
Somme: 68
```

- La variable spéciale **NR** contient le nombre de lignes (records) parcourues
- Calcule la moyenne de nombres stockés dans un fichier (un nombre par ligne)

```
% awk '{s+=$1} END {print "Moyenne: " s/NR}' nbres.csv
Moyenne: 13.6
```

awk: exemples

- La variable spéciale **NF** contient le nombre de champs (fields) dans la ligne courante
- Calcule la somme des nombres sur chaque ligne

```
% cat nombres2.csv
1,2,3,4
5,6
% awk -F, '{s=0; for(i=1;i<=NF;i++) s+=$i; print s}\
  nombres2.csv
10
11
```

- Afficher toutes les balises html

```
% awk –F">[^<]*<|[<>]" \
 '{ for(i=1;i<=NF;i++) print "<"$i">" }' fichier.html

<body>
<h1>
</h1>
</body>
```

Redirection d'entrées-sorties

- Tous les exemples précédent lisent leurs entrées dans un fichier, et affichent le résultat dans le terminal.
- La redirection d'entrées-sorties résoud ce problème
- Flux existants
  - entrée standard — stdin, std::cin, System.in
  - sortie standard — stdout, std::cout, System.out
  - erreur — stderr, std::cerr, System.err

Redirection d'entrées-sorties

- Redirection de l'entrée depuis un fichier
  `% commande < fichier`
- Redirection de la sortie vers un fichier
  `% commande > fichier` (écrasement)
  `% commande >> fichier` (concaténation)
- Redirection de l'erreur
  `% commande >& fichier`
- Les *pipes* permettent d'utiliser la sortie d'une commande comme entrée de la commande suivante
  `% commande1 | commande2`

Exemples

- Toutes ces formulations sont équivalentes

```
% grep t.m. wordlist.txt > /tmp/toto
% wc –l /tmp/toto
827

% grep t.m. wordlist.txt | wc –l
827

% cat wordlist.txt | grep t.m. | wc –l
827

% grep t.m. < wordlist.txt | wc –l
827
```

- Le fichier spécial /dev/null est un "trou sans fond"

```
% find / -name "apache" >& /dev/null
```

Exemples

- Les redirections sont valable pour n'importe quel programme

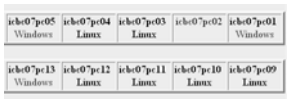
```
% cat lowercase.c
#include <stdio.h>
#include <ctype.h>

int main() {
  char c=0;
  while((c=fgetc(stdin))!=EOF)
    printf("%c",tolower(c));
  return 0;
}

% gcc -Wall -o lowercase lowercase.c
% cat sample.txt
Bonjour Maitre ZEN
% ./lowercase < sample.txt
bonjour maitre zen
% cat sample.txt | ./lowercase
bonjour maitre zen
```

Exemple "utile"

- Récupération de données depuis le web
- Une page web indique l'état des machines
  - http://ic-it.epfl.ch/it/sallesinfo/bc0708/bc0708_users.html



- Code source de la page

```
<TD COLSPAN=2 WIDTH=80>
<font color="black"><B>icbc07pc12</B></font>
<BR>
<font color="green"><b>Windows</b></font><BR>
</TD>
<TD COLSPAN=2 WIDTH=80>
<font color="black"><B>icbc07pc11</B></font>
<BR>
<font color="blue"><b>Linux</b></font><BR>
</TD>
```

Exemple "utile"

- wget permet de télécharger des fichiers sur le réseau
  - L'option '**-q**' supprime son affichage, et '**-O -**' lui demande d'écrire le résultat sur la sortie standard (plutôt que dans un fichier)
- On filtre les lignes intéressantes avec grep
  - L'option '**-B**' spécifie le nombre de lignes à afficher avant chaque correspondance

```
% wget –q –O – \
http://ic-it.epfl.ch/it/sallesinfo/bc0708/bc0708_users.html |
grep –B 2 Linux
...
<font color="black"><B>icbc07pc11</B></font>
<BR>
<font color="blue"><b>Linux</b></font><BR>
...
```

On récupère les noms de machines avec une passe de grep et de awk

```
% wget –q –O – http://... |
grep –B 2 Linux | grep icbc07 | awk -F"[<>]" '{print $5}'
icbc07pc04
icbc07pc11
...
```

Qu'est-ce qu'un shell?

- Un interpréteur de commandes
- Interface entre l'utilisateur et le système
- C'est lui qui exécute les commandes, analyse leurs arguments, et gère leurs entrées-sorties
- C'est aussi un langage de programmation à part entière

- De nombreux dialectes existent: *sh*, *csh*, *tcsh*, *bash*, *ksh*, *zsh*… Ici on utilisera *bash*, ou Bourne again shell
  - Shell par défaut sous Solaris et BC 07-08: tcsh
  - Linux et Mac OS X: bash
- Un shell est un programme, et peut donc être invoqué sur la ligne de commande

Variables

- Affectation et utilisation

```
var="quelque chose"        # Sans espaces autour du signe '='
$var
```

- Elles ne sont pas typées et n'ont pas besoin d'être déclarées
  - Utiliser une variable non initialisée ne produit pas d'erreur, elle est considérée comme vide
- Les variables sont par défaut locales, i.e. accessible uniquement au shell courant
- Elles peuvent être rendues globales, et donc accessibles par les processus fils en utilisant

export VAR=valeur	sh et bash (Mac OS X, Linux)
setenv VAR valeur	tcsh (Solaris, BC 07-08)

- Les variables globales sont aussi appelées variables d'environnement

Example

```
% var="Cette variable est locale"
% echo $var
Cette variable est locale
% bash
> echo $var

> exit
exit
% export VAR="Cette variable est globale"
% bash
> echo $VAR
Cette variable est globale
```

- Conventions: variables locales en minuscule, variables globales en majuscules

Variables d'environnement

- Un shell (et par extension les processus fils du shell) utilisent de nombreuses variables globales

PATH	Localisation des exécutables
SHELL	Shell par défaut de l'utilisateur
HOSTNAME	Nom de la machine
HOME	Répertoire de l'utilisateur courant
USER	Login de l'utilisateur courant
PWD	Répertoire courant

- D'autres sont utilisées par vos programmes

LD_LIBRARY_PATH	Localisation des librairies dynamiques (C et C++)
CLASSPATH	Localisation de classes Java

- La liste des variables globales est accessibles avec la commande env

Structures de contrôle

- If
  if *test*; then
  instructions-if-true
  else
  instructions-if-false
  fi
- Test est fait sur la valeur de retour d'un programme
  if wc –l fichier; then
  echo The program ran correctly
  fi
- ! inverse la valeur du test
- La variable spéciale $? contient la valeur de retour de la dernière commande exécutée

Commandes *test* et *[*

- test int1 -eq int2 ou [int1 -eq int2]
  - équivalent à (int1==int2)
- Opérateurs arithmétiques
  - -ne: inégalité
  - -lt et -gt: plus petit et plus grand que
  - -le et -ge: plus petit ou égal et plus grand ou égal
- Opérateurs sur des fichiers
  - -r: vrai si le fichier existe et est lisible
  - -d: vrai si le fichier est un répertoire
  - -e: vrai si le fichier existe
- *man test* pour la liste exhaustive

Boucles

- Boucle for
  for *variable* in *value-list* ; do
  instructions
  done;
- Exemple
  for i in 1 2 3 4 5; do echo $i; done
- Boucle while
  while *test* ; do
  instructions
  done

Substitution de commande

- L'affichage d'une commande est souvent plus utile que sa valeur de retour
  - Dans une boucle *for*
    for i in *all-cpp-files* ; do …
- Une commande placée entre ` ` est exécutée et est remplacée par sa sortie

```
% wc –l `find . –name "*.cpp"`
% for i in `ls *.cpp`; do echo $i; done
% cpp_files=`ls *.cpp`
```

- La commande *seq* est très utile pour obtenir des suites de nombres

```
% seq 1 10      # Affichage: 1 2 3 4 5 6 7 8 9 10
% seq 2 2 10    # Affichage: 2 4 6 8 10      (un par ligne)
```

Scripts shell

- Les commandes sont placées dans un fichier
  - Evite de retaper des commandes à chaque fois
  - Permet de développer des programmes plus sophistiqués
- Spécifier le shell à utiliser sur la première ligne
  #!/usr/bin/bash
  echo Ceci est un script shell
- Le chemin du shell peut être obtenu à l'aide de *which*
  which bash
- Les commentaires commencent par #
- Les scripts peuvent prendre des arguments de ligne de commande, accessibles à l'aide de variables spéciales
  - $0 contient le nom du script (e.g. ./script.sh)
  - $n contient le n^{ème} argument
  - $# contient le nombre d'arguments passés au script

Exemple: connectBC.sh

```
#!/bin/bash
# Prefix of computer host name
name=icbc07pc0

# Look for a host where we can connect
for i in `seq 1 9`; do
    ssh –o ConnectTimeout=3 $name$i exit && break
done

# Connect if successful
if [ $? –eq 0 ]; then
    ssh –X $name$i
else
    echo No host is booted under Linux
fi
```

Exemple: connectBC.sh, v2

```
#!/bin/bash
# Web page URL
url="http://ic-it.epfl.ch/it/sallesinfo/bc0708/bc0708_users.html"
# Temporary file
tmpfile=/tmp/linuxhostsbc07.tmp
if [ -e $tmpfile ]; then echo Temp file already exists; exit; fi

wget -q -O - $url | grep Linux -B 2 | grep icbc07 > $tmpfile
if [ $? –ne 0 ]; then rm -f $tmpfile; exit; fi

host=`head -n 1 $tmpfile | awk –F"[<>]" '{print $5}'`
rm $tmpfile

ssh –X $host
```

Utiliser un script

- Le rendre exécutable
  `% chmod u+x connectBC.sh`
- On l'exécute comme n'importe quel programme
  `% ./connectBC.sh`
- Pour que le shell puisse exécuter un script (ou un programme) depuis n'importe où, il faut que le répertoire contenant le script se trouve dans le PATH
  `% mkdir ~/bin; mv connectBC.sh ~/bin/`
  `% export PATH=~/bin:$PATH`
- Pour que le PATH soit modifié de manière permanente, utilisez ~/.bashrc
  - Le fichier ~/.bashrc (ou ~/.cshrc) est exécuté à chaque login

  `% echo "export PATH=$HOME/bin:$PATH" >> ~/.bashrc`

Exemple I

```
#!/bin/bash
# Take first and last word that start with every letter
for letter in a b c d e f g h i j k l m n o p q r s t u v w
  x y z; do
  grep "^$letter" list.txt | head -n 1 >> list-short.txt
  grep "^$letter" list.txt | tail -n 1 >> list-short.txt
done

# Build lists with specified number of words
for n in 1000 5000 10000 50000; do
#  awk '{ if(i<'$n') { print $1; i++; } }' list.txt >> list-
  $n.txt
```

Exemple II

```
#!/bin/bash
if [ ! $1 ]; then
    echo "Please specify a pattern"; exit
fi

pattern=$1
dir=../LaboB/ex2
exec1=$dir/lineardictM
exec2=$dir/treedictM

for i in 1000 5000 10000 50000; do
    t1=`$exec1 $dir/list-$i.txt 1 $pattern | grep matching | awk '{print $4}'`
    t2=`$exec2 $dir/list-$i.txt 1 $pattern | grep matching | awk '{print $4}'`
    echo $i $t1 $t2 >> data.dat
done
```

Exemple II

```
% ./timings.sh advanced
% cat data.dat
1000   0.281261   0.000961
5000   1.29195    0.001001
10000  2.68818    0.001087
50000  14.7705    0.001157
```

Exemple II: gnuplot

- Gnuplot permet de génèrer des graphes
  - Programme en ligne de commande
  - Peut être utilisé de manière interactive ou non
- Références
  - http://www.gnuplot.info/
  - http://t16web.lanl.gov/Kawano/gnuplot/index-e.html

Exemple II: gnuplot

- Les commandes suivantes génèrent un graphe avec les mesures effectuées

```
# Output graph as a PNG image
set terminal png

set title 'Impact of data structure and\
 dictionary size on word search performance'
set xlabel 'Number of words in dictionary'
set ylabel 'Running time for 1000 searches [s]'

plot "data.dat" using 1:2 title 'Linear dictionary' with linespoints, \
     "data.dat" using 1:3 title 'Tree dictionary' with linespoints
```

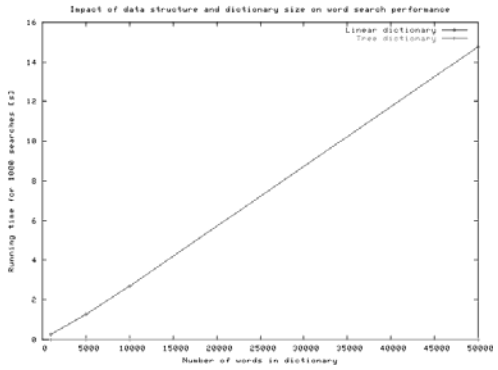
Exemple II

```
#!/bin/bash
if [ ! $1 ]; then
   echo "Please specify a pattern"
   exit
fi

./timings $1

gnuplot < data.plot > timings.png
```

Exemple II



Exemple II

```
#!/bin/bash
…
# Generate plot file within the script
file=/tmp/data.plot
rm -f $file
echo "set terminal png" >> $file
echo "set title 'Impact of data structure and\" >> $file
echo " dictionary size on word search performance' " >> $file
echo "set xlabel 'Number of words in dictionary' " >> $file
echo "set ylabel 'Running time for 1000 searches [s]' " >> $file
echo "plot \"data.dat\" using 1:2 title 'Linear dictionary' with linespoints, \ "
    >> $file
echo " plot \"data.dat\" using 1:3 title 'Tree dictionary' with linespoints " >>
    $file

gnuplot < $file > timings.png
rm -f $file
```

Exemple III: logs serveur web

```
…
128.178.75.30 - - [29/Jul/2005:11:02:49 +0200] "GET /cgi-bin/viewcvs.cgi/ HTTP/1.1" 200 5620 "-" "Mozilla/5.0
    (Windows; U; Windows NT 5.1; en-US; rv:1.7.10) Gecko/20050716 Firefox/1.0.6"
128.178.75.30 - - [29/Jul/2005:11:02:49 +0200] "GET /viewcvs/images/logo.png HTTP/1.1" 200 7262
    "http://lsppc54/cgi-bin/viewcvs.cgi/" "Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7.10)
    Gecko/20050716 Firefox/1.0.6"
…
207.14.166.253 - - [29/Aug/2005:02:59:03 +0200] "GET /phpMyAdmin-2.5.1/main.php HTTP/1.0" 404 291 "-"
    "pmafind"
207.14.166.253 - - [29/Aug/2005:02:59:04 +0200] "GET /phpMyAdmin-2.2.3/main.php HTTP/1.0" 404 291 "-"
    "pmafind"
207.14.166.253 - - [29/Aug/2005:02:59:04 +0200] "GET /phpMyAdmin-2.2.6/main.php HTTP/1.0" 404 291 "-"
    "pmafind"
66.135.32.166 - - [29/Aug/2005:10:45:01 +0200] "GET /phpmyadmin/index.php HTTP/1.0" 404 286 "-" "-"
211.200.228.133 - - [29/Aug/2005:22:37:33 +0200] "GET /level/16/exec/-///pwd  HTTP/1.0" 404 287 "-" "-"
218.166.62.47 - - [29/Aug/2005:22:53:32 +0200] "GET /level/16/exec/-///pwd  HTTP/1.0" 404 287 "-" "-"
219.95.37.155 - - [29/Aug/2005:23:12:24 +0200] "GET /level/16/exec/-///pwd  HTTP/1.0" 404 287 "-" "-"
221.139.69.68 - - [29/Aug/2005:23:56:18 +0200] "GET /level/16/exec/-///pwd  HTTP/1.0" 404 287 "-" "-"
65.78.169.195 - - [30/Aug/2005:13:00:08 +0200] "SEARCH /\x90\x04H\x04H\x04H\x04H\x04H\x04H\x04H…
65.78.169.195 - - [30/Aug/2005:13:00:08 +0200] "POST /_vti_bin/_vti_aut/fp30reg.dll HTTP/1.1" 404 307 "-" "-"
65.78.169.195 - - [30/Aug/2005:13:27:36 +0200] "POST /_vti_bin/_vti_aut/fp30reg.dll HTTP/1.1" 404 307 "-" "-"
65.78.169.195 - - [30/Aug/2005:13:27:36 +0200] "SEARCH /\x90\x04H\x04H\x04H\x04H\x04H\x04H\x04H…
…
```

Exemple III

- Provenance des clients

```
ips=`grep Windows /var/log/apache/access.log |
  awk '{print $1}' | sort | uniq`

for ip in $ips; do
   nslookup $ip
done
```

Pour terminer

- De manière générale, les scripts shell sont surtout utiles pour
  - Administrer des systèmes
  - Traiter des résultats de mesures, de monitoring, …
  - Tirer parti de différents programmes et utilitaires
- Bien qu'on puisse faire beaucoup de choses avec des scripts shell, passé un certain niveau de complexité, des "vrais" langages sont plus adaptés
  - Perl en particulier permet de mélanger les syntaxes C et shell
  - Si vous avez besoin de choses non décrites ici, il est probablement plus simple d'utiliser un autre langage de script
  - http://www.tldp.org/LDP/abs/html/why-shell.html

Références

- Bourne shell
  http://www.ooblick.com/text/sh/
- Bash
  http://www.beforever.com/bashtut.htm
  configuration
  http://www.hypexr.org/bash_tutorial.php
  avancé
  http://www.tldp.org/LDP/abs/html/
- sed one-liners
  http://www.unixguide.net/unix/sedoneliner.shtml
- awk one-liners
  http://www.softpanorama.org/Tools/Awk/awk_one_liners.shtml