Gebruik van command-line operating systems

Mattias Holm & Kristian Rietveld

Universiteit Leiden
The Netherlands

Overzicht

- Waarom hier meer over leren?
- Wat is een shell?
- Hoe werkt een shell?
- Pipes en redirectie
- Handige utilities
- Shell scripting
  - `Back ticks`
  - Control flow
  - Shell configuratie
- Verdere informatie & opdracht

Waarom hier meer over leren?

- Goed begrip van de werking van de shell belangrijk voor het bevatten van operating systems.
- En ter uitbreiding van je “gereedschapskist”
- Door te leren programmeren in shell kan je in de toekomst veel tijd uitsparen!

Wat is een shell?

- Iedereen heeft ooit met een shell gewerkt.
- Windows: command.com, powershell
- UNIX: bash, tcsh, zsh, etc.
- Er bestaan ook “graphical shells”, maar wij zullen ons richten op command line.
- Shell waarmee we vandaag werken: bash.

Wat is een shell?

- De voornaamste taak van een shell is het opstarten van programma's.
- Daarnaast beheert het de “environment” en “current working directory”.
- Shells hebben vaak ingebouwde commando's, waar de scheiding ligt verschilt per shell en besturingssysteem.

Shell environment

- Opslag van verschillende variabelen belangrijk voor het functioneren van het systeem.
- Voorbeelden:
  - PATH
  - HOME
  - USER

Hoe werkt een shell?

- Stel we typen in “ls”.
- “ls” is geen ingebouwd commando.
- Voor elke directory in PATH kijken we of deze een “ls” executable bevat.
- Zodra we een executable vinden wordt deze geladen en uitgevoerd.
- Als de executable “klaar” is komt de controle terug bij de shell.

Executie in de achtergrond

- Je kan een programma ook in de achtergrond uitvoeren, zodat de controle meteen weer terug komt bij de shell.
- Plaats het teken “&” direct achter de opdracht.

Job control

- Met het commando “jobs” kun je bekijken welke processen de shell beheert.
- Je kan refereren naar proces n met %n
- Zo kan je een proces naar de voorgrond brengen met “fg %1”
- Of naar de achtergrond met “bg %1”
- Proces op de voorgrond tijdelijk stoppen kan met control-Z.
- Control-C beeindigt het proces normaliter.
- Control-D stuurt EOF signaal.

Pipes en redirection

- Veel programma's produceren output.
- Vaak de keuze of dit naar file is of *stdout* of *stderr*.
- Input komt ook of vanuit een file of vanuit *stdin*.

Pipes

- Met de shell kan je *stdout* van een proces knopen aan *stdin* van een ander.
- Symbool: “|” (pipe)
- Zo'n aaneenschakeling wordt ook wel een “pipeline” genoemd.

```
$ cat bestand | sort
$ cat bestand | sort | uniq
```

Redirectie

- Daarnaast kan je met de shell de *stdout* van een proces knopen aan een file.
- Analoog voor *stdin*.
- Symbolen:
  - Output: “>” (“uit het proces”)
  - Input: “<” (“in het proces”)

```
$ cat bestand | sort > gesorteerd
$ cat bestand | sort | uniq > unieken
$ sort < bestand
$ sort < bestand > gesorteerd
```

Redirectie (vervolg)

- Standaard zal “>” de huidige file overschrijven. Om dit te voorkomen:
  - Output en append: “>>”

```
$ cat bestand > alles
$ cat bestand1 >> alles
$ cat bestand2 >> alles
```

Redirectie (vervolg)

- Ook kunnen we specifieke *file descriptors* selecteren voor input of output.
- In het algemeen:
  - 0 = stdin
  - 1 = stdout
  - 2 = stderr

```
$ gcc fout.c > warnings   # werkt niet!
$ gcc fout.c 2> warnings
```

Redirectie (vervolg)

- Je kan file descriptors ook dupliceren.
- Syntax:
  - “2>&1” file descriptor 2 is nu een kopie van 1.
- Alles wat wordt geschreven naar *stderr*, komt nu terecht in *stdout*.
- Volgorde is belangrijk!

```
$ gcc fout.c | less   # werkt niet!
$ gcc fout.c 2>&1 | less
$ gcc fout.c > warnings 2>&1
```

Utilities

- sort — Sorteren
- uniq — Van elk maar 1 doorlaten
- less — Pager
- grep — Filteren
- sed — “Stream editor”
- cut — Stukken uit regels halen
- wget — Files downloaden

```
$ cat bestand | grep " is " | less
$ cat bestand | grep "^a" | less
$ cat bestand | sed "s/is/was/" | less
$ cat bestand | cut -b 1-4,6
$ wget http://www.test.nl/test.txt
```

Utilities (vervolg)

- Er zijn veel meer van dit soort utilities.
- Zoeken kan met “*man -k <woord>*”.
- Uiteraard kan je ook zelf dit soort utilities schrijven!
- Sommige utilities maken gebruik van regular expressions, erg handig om te bestuderen.

Regular Expression

- Enkele symbolen:
  - ^ betekent begin van de regel
  - $ betekent eind van de regel
  - . elk mogelijk karakter
  - * 0 of meer keer
  - ? 1 of meer keer
  - {n,m} tussen n en m keer
  - /.../ "match" operator

Regular Expressions (verv.)

- Voorbeelden:
  - /^foo/ foo aan het begin van de regel.
  - /foo$/ foo aan het eind van de regel.
  - /^foo$/ regel bevat alleen foo.
  - /^.foo$/ elk mogelijk karakter, daarna foo.
  - /9?foo/ 1 of meer negens, gevolgd door foo.

Regular Expression (verv.)

- /.*foo/ allerlei mogelijke combinaties, daarna foo overal in de string.
- /.*foo$/ zelfde als bovenstaande, maar na foo mag niks volgen.

Shell scripting

- Pipelines kunnen lang worden en je wilt deze niet keer op keer intypen.
- We kunnen deze hergebruiken door een script file te maken.
- Deze scripts kunnen dan worden uitgevoerd alsof het programma's zijn.

Shell scripting

- De “shebang” regel (#!) geeft aan welk programma het script kan uitvoeren.
- Daarvoor moet het script executable zijn: *chmod + x < script_file>*
- *$1, $2* enzovoort bevatten de argumenten.
- *$** bevat alle argumenten.
- *$?* bevat exit code van laatste programma

```
#!/bin/bash
echo $1 $2
echo $*
cat $1 | sort | uniq > $2
```

Shell scripting (vervolg)

- We zagen al dat je *echo* kunt gebruiken om strings te printen.
- Vaak is er ook een geavanceerder *printf* commando.

Shell scripting (vervolg)

- Je kan variabelen aanmaken en gebruiken.
- En er ook mee rekenen.
- Let op: geen spaties rond de “=”!
- Met “*export*” plaats je een variable in het “environment”.

```
$ a=1
$ b=$((a+1))
$ b=$((b/2))
```

Verschil in quotes

- “..”: Interpreteer variabelen.
- '..': Geen interpretatie van variabelen.
- `..`: Interpreteer variabelen en het resultaat als commando en voer deze uit.

```
$ a=1
$ echo "ls $a"
$ echo 'ls $a'
$ echo `ls $a`
$ a=/usr
$ echo `ls $a`
```

Control flow

- Zoals de meeste andere programmeertalen, heeft bash ook “compound commands” voor control flow:
  - if/else,
  - for,
  - while,
  - case.
- Het is overigens zo dat control flow niet alleen in een script file werkt, ook gewoon op de command line.

For compound command

- Itereert over een lijst elementen, heeft het meest weg van *foreach*.

```
for i in 1 2 3; do
     echo $i;
done

for i in ls -1; do
     echo $i;
done

for i in `ls -1`; do
     echo $i;
done
```

If compound command

- Kies een lijst commando's om uit te voeren, gebaseerd op de exit status van de “conditie”.

```
if true; then
      echo "test";
else
      echo "false";
fi
```

“test” utilitity

- Als “conditie commando” wordt vaak gebruik gemaakt van het progamma “test”
- [is een symlink naar of kopie van “test”
- De exit status geeft het resultaat van de evaluatie aan: nul = true, niet-nul = false

“test” utility (vervolg)

- Met “-e” kunnen we bijvoorbeeld bekijken of een file bestaat:

```
$ test -e /dev
$ echo $?
0
$ test -e /def
$ echo $?
1
```

“test” utility (vervolg)

- Of met het if compound commando:

```
if test -e /dev; then
     echo "/dev exists";
fi

if [ -e /dev ] ; then
     echo "/dev exists";
fi
```

“test” utility (vervolg)

- Andere mogelijkheden:
  - Meer file tests
  - String comparison (=, !=, <, >)
  - Integer comparison (-eq, -ne, -gt, -lt)
  - Boolean AND en OR (-a, -o)
- Je kan gewoon shell variabelen gebruiken, aangezien deze worden geexpandeerd voordat “test” wordt aangeroepen.

Case compound command

- Werkt als een simpel switch statement.

```
case $1 in
     'start')
          echo "start selected";
          ;;
     'end')
          echo "end selected";
          ;;
     *)
          echo "nothing selected";
          ;;
esac
```

Shell aanpassen

- De shell kan je uiteraard aanpassen aan je wensen.
- Vaak door het aanpassen van environment variabelen.
- Bijvoorbeeld het prompt, variabele PS1:
  - *export PS1= ”\u@ \h:\w\$ “*
- Daarnaast kun je dingen instellen met “shopt” en “ulimit”

Wanneer de shell start

- Je kan commando's laten uitvoeren wanneer de shell wordt gestart.
- Gebruik daarvoor de volgende shell scripts:
  - Voor een “login shell”: ~/.bash_profile
  - Anders: ~/.bashrc

Voorbeelden

- /etc/init.d/postgresql

Meer informatie

- man bash
- *Classic Shell Scripting*. Robbins, Beebe. O'Reilly.
- */etc/init.d* op Linux systemen staat meestal vol met (goede en slechte) voorbeelden.

Volgende week

- Meer utilities die je samen met shell kunt gebruiken.
- Awk.
- Perl.
- Ins en outs van Makefiles.
- Makefiles en shell combineren.

Opdracht / Practicum

- Practicum in zaal 411.