Introduction to Bash video lecture

14 - Advanced topics: editing the command line, exit status, conditionals, globbing, expansion...

Editing the command line - get efficient

- Arrow up goes up the history of commands, arrow down goes back
- One single arrow up gives you the previous command
- Escape and then a dot (or Alt-.) gives you the previous command’s last argument
- Ctrl-A moves the cursor to the beginning of the line, Ctrl-E to the end
- Ctrl-arrow left skips one word to the left, Ctrl-arrow right, the opposite
- Ctrl-W erases the word to the left, Ctrl-K erases all words to the right
- Ctrl-Y pastes whatever was erased last

More nifty tricks

- !! issues the last command again
- Ctrl-R - searches the history interactively

Demonstration

Exit status

- All commands report their exit status to the shell
- Numeric value, where 0 means expected good result
- Other numbers mean some kind of failure, see the manual to learn what
- Is stored in the variable $?
- Allows you to use && for commands that depend on success of previous commands and || for commands to issue if previous failed

```
rikard@newdelli:~/bash-intro/text-files$ find lorem80.txt && wc -l lorem80.txt
lorem80.txt
38 lorem80.txt
rikard@newdelli:~/bash-intro/text-files$ find lorem90.txt && wc -l lorem90.txt
find: ‘lorem90.txt’: No such file or directory
rikard@newdelli:~/bash-intro/text-files$ find lorem80.txt > /dev/null && wc -l lorem80.txt
38 lorem80.txt
rikard@newdelli:~/bash-intro/text-files$ find lorem90.txt &> /dev/null && wc -l lorem90.txt
```

Some words on the if-statement

- `if` takes one or more commands as arguments
- if the command (or last command) has exit status 0, then the `then`-branch is executed
- otherwise the `elif`- or `else`-branch is executed
- ends with `fi`
- There is a version of the if-statement that uses double parentheses:

```
if (( points > 37 ))
then
  grade="VG"
fi
```

Example IF

```
$ if date | grep Mon
> then
>   echo "Week starts now"
> elif date | grep Tue
> then
>   echo "Only four more days"
> else
>   echo "It's not Monday or Tuesday"
> fi
Tue Jul 30 14:38:59 CEST 2019
Only four more days
```

Example conditional

```
$ date | grep -q Tue && echo "Tuesdays rock" || echo "It's not Tuesday"
Tuesdays rock
```

Globbing

- Used to expand filenames (and directory names)
- * means “Anything”
  - *.txt - all files ending with .txt
- [0-9] means any one character between 0 and 9
- [A-H] means any one character between A and H
- ? means any one character

Brace expansion

- Curly braces allow us to expand combinations

```
$ echo SVT{1,2,24}
SVT1 SVT2 SVT24
$
```

Brace expansion

- Can be nested and very powerful

```
music/
├── classical
│   ├── classicism
│   ├── modernism
│   ├── modernist
│   └── renaissance
├── jazz
│   ├── bebop
│   ├── free_jazz
│   └── fusion
└── rock
    ├── hard_rock
    ├── metal
    └── rockabilly
```

Brace expansion

```
# the directory tree was created with one single command line:

$ mkdir -p
music/{classical/{modernist,renaissance,classicism,modernism},rock/{hard_rock,
metal,rockabilly},jazz/{bebop,free_jazz,fusion}}

# all on one line
```

Variables

- A named memory location
- Use $variable to expand the value
- Environment variables are shared between shells and initialized when the shell starts
- Variable you create are local to the shell where they were created

Arguments to scripts end up in special variables

```
rikard@newdelli:~/bash-intro/text-files$ cat arguments.sh
#!/bin/bash

echo "Script name: $0"
echo "Number of arguments: $#"
echo "All arguments $*"
echo "First argument: $1"
echo "Second argument: $2"

rikard@newdelli:~/bash-intro/text-files$ ./arguments.sh one two
Script name: ./arguments.sh
Number of arguments: 2
All arguments one two
First argument: one
Second argument: two
rikard@newdelli:~/bash-intro/text-files$
```

Use quotes around variables

- If a variable contains spaces, Bash will treat the value as many words if used unquoted
- Using double quotes around a variable when used, will tell Bash to treat it as one single string (which may or may not contain spaces)

Forgetting to use quotes

```
rikard@newdelli:~/bash-intro/text-files$ name="Rikard Fröberg"
rikard@newdelli:~/bash-intro/text-files$ mkdir $name  ← oops! should have used quotes
rikard@newdelli:~/bash-intro/text-files$ ls
a_few_urls.txt      group_genre.txt                   lorem.txt
apa                 latin_uniq_frequencies.txt        replaceme.txt
four.txt            latin_words_sorted_lower_case.txt  Rikard
frequency_table.txt  latin_words_sorted.txt            small_text.txt
Fröberg             latin_words.txt                   swe.txt
group_album.txt     lorem80.txt
rikard@newdelli:~/bash-intro/text-files$ ls -ltr
total 68
... (cut to fit the slide)...
-rw-rw-r-- 1 rikard rikard 2073 jul 25 14:42 frequency_table.txt
-rw-rw-r-- 1 rikard rikard  131 jul 25 14:53 group_album.txt
-rw-rw-r-- 1 rikard rikard   55 jul 25 14:53 group_genre.txt
drwxrwxr-x 2 rikard rikard 4096 jul 26 11:37 apa
drwxrwxr-x 2 rikard rikard 4096 jul 29 10:02 Rikard
drwxrwxr-x 2 rikard rikard 4096 jul 29 10:02 Fröberg
```

<<last page>>