CHALMERS

GÖTEBORGS UNIVERSITET

Introduction to Linux

Lecture 5

Prof. Dr. Philippe Tassin
Chalmers University of Technology

Dr. Evangelos Siminos
Chalmers University of Technology and University of Gothenburg

Göteborg, 24 April 2019

The Bash shell

- Bash is the standard GNU shell

- Different types of invocation:
  - Login vs non-login shells
  - Interactive vs. noninteractive shells
  - Remote shell (rbash): never use this (use ssh/scp/sftp instead)

- Different startup scripts are run for different invocations:
  - Login: `/etc/profile` and `~/.bash_profile`
  - Interactive, non-login: `~/.bashrc`
  - Noninteractive: $BASHENV (normally none)

Bash shell features

- Bash is also a scripting language
- Interpreted language (the shell is the interpreter)
- High-level (abstract) programming language
- Some specific features:
  - Expansion
  - Aliases and functions
  - Variables
  - Conditionals and test expressions
  - Search order for commands (alias, functions, builtins, $PATH)

Bash scripts

- Commands can be saved in a file and executed as a script
- A noninteractive shell is the interpreter
- Scripts in Linux should start with a “shebang”:

  ```
  #!/bin/bash
  ```

  (for other scripting languages, use the right interpreter)
- Scripts run in a noninteractive shell, so any aliases in the startup scripts are not available!

Running and debugging scripts

- Set file permissions (`chmod +x filename`)
- Command search does not look in the current directory, so you need `./scriptname` to run if the script is not in the search path
- Comment lines start with #
- Debugging with -v and -x options
  - `bash -xv scriptname`
  - `set -xv` (`set +xv` to disable)
  - `#!/bin/bash -xv`

Example script

```
tom:~> cat mysystem.sh
#!/bin/bash
clear
printf "This is information provided by mysystem.sh.  Program starts now.\n"

printf "Hello, $USER.\n\n"

printf "Today's date is `date`, this is week `date +"%V"`.\n\n"

printf "These users are currently connected:\n"
w | cut -d " " -f 1 - | grep -v USER | sort -u
printf "\n"

printf "This is `uname -s` running on a `uname -m` processor.\n\n"

printf "This is the uptime information:\n"
uptime
printf "\n"

printf "That's all folks!\n"
```

Aliases

- Replace a string by a single word
  - E.g.: `alias ll='ls -la'`
  - `alias grep='grep -E'`
- `unalias` to undo
- Highest priority in search order!
- Typical use only for common commands in interactive shells; not often used in scripts
- Normally expanded only in the first word of a command
- Aliases may form chains, but if first word is same as alias name, it is not expanded again (to prevent infinite loops)

Shell expansion

- The shell will expand certain characters and words
  - Brace expansion
  - Tilde expansion
  - Parameter and variable expansion
  - Command substitution
  - Arithmetic expansion
  - File name expansion

Brace and tilde expansion

- `preamble{alt1,alt2,alt3}postscript`

is expanded to the space separated list

```
preamblealt1postscript preamblealt2postscript
preamblealt3postscript
```

- ~ is expanded to your home directory

Variable and parameter expansion

- `${}` means variable expansion
-
```
VAR1="some text"
echo "${VAR1}"
```
- Arguments to a script are represented by the positional parameters $1, $2, … , $9, ${10}, ${11}, etc.

Character	Definition
`$*`	Expands to the positional parameters, starting from one. When the expansion occurs within double quotes, it expands to a single word with the value of each parameter separated by the first character of the `IFS` special variable.
`$@`	Expands to the positional parameters, starting from one. When the expansion occurs within double quotes, each parameter expands to a separate word.
`$#`	Expands to the number of positional parameters in decimal.
`$?`	Expands to the exit status of the most recently executed foreground pipeline.
`$-`	A hyphen expands to the current option flags as specified upon invocation, by the **set** built-in command, or those set by the shell itself (such as the `-i`).
`$$`	Expands to the process ID of the shell.
`$!`	Expands to the process ID of the most recently executed background (asynchronous) command.
`$0`	Expands to the name of the shell or shell script.
`$_`	The underscore variable is set at shell startup and contains the absolute file name of the shell or script being executed as passed in the argument list. Subsequently, it expands to the last argument to the previous command, after expansion. It is also set to the full pathname of each command executed and placed in the environment exported to that command. When checking mail, this parameter holds the name of the mail file.

Command substitution

- When you need the output of a command in another command

```
[tassin@hebbe ~]$ echo $( date )
Wed Apr 18 15:15:48 CEST 2018
```

- Don’t use backticks (it works but it has its problems)

Arithmetic expansion

- Perform arithmetic operations on variables
- Only integers! (including some binary operations)
- Operators:

Operator	Meaning
VAR++ and VAR--	variable post-increment and post-decrement
++VAR and --VAR	variable pre-increment and pre-decrement
- and +	unary minus and plus
! and ~	logical and bitwise negation
**	exponentiation
*, / and %	multiplication, division, remainder
+ and -	addition, subtraction
<< and >>	left and right bitwise shifts
<=, >=, < and >	comparison operators
== and !=	equality and inequality
&	bitwise AND
^	bitwise exclusive OR
|	bitwise OR
&&	logical AND
||	logical OR
expr ? expr : expr	conditional evaluation
=, *=, /=, %=, +=, -=, <<=, >>=, &=, ^= and |=	assignments
,	separator between expressions

File name expansion

- The characters `*`, `?` and `[]` are used for filename expansion

- `*` matching any file
- `*/` matches any directory
- `*t*` matches all filenames containing a “t”

- `?` matches a single character

- `[2-9]` is a range
- The character classes like `[[:upper:]]` etc. can be used

Conditional statements

- General syntax:

  ```
  if teststatement; then commands; fi
  ```

- The test statement is a condition to be checked; the condition appears within [ ]

  e.g.: `[ -d filename ]` checks if a filename exists and is a directory

  - See the list in the textbook (or the Bash manual) for more examples

- `$#` is the number of arguments

Example of conditional statements

```
anny ~> cat msgcheck.sh
#!/bin/bash

echo "This scripts checks the existence of the messages file."
echo "Checking..."
if [ -f /var/log/messages ]
  then
    echo "/var/log/messages exists."
fi
echo
echo "...done."

anny ~> ./msgcheck.sh
This scripts checks the existence of the messages file.
Checking...
/var/log/messages exists.

...done.
```

If…then…else

```
if TEST-COMMANDS; then
  CONSEQUENT-COMMANDS;
elif MORE-TEST-COMMANDS; then
  MORE-CONSEQUENT-COMMANDS;
else
  ALTERNATE-CONSEQUENT-COMMANDS;
fi
```

Exit

- `exit` terminates a script and allows to set the exit status
- `exit 1`
- Exit status can be checked in the parent using the `$?` variable

Case statements

```
case EXPRESSION in
  CASE1)
    COMMAND-LIST;;
  CASE2)
    COMMAND-LIST;;
  ...
  CASEN)
    COMMAND-LIST;;
esac
```

Any other questions?