Command shells and shell scripting

Операционни системи, ФМИ, 2017-2018

command shell

- user-interface
- access to filesystem
- scriptability for task automation
- program launching
- process control interface

shells

- Thompson Shell (`sh`) – Ken Thompson, 1971, AT&T
- Bourne Shell (`sh`) – Stephen Bourne, 1977, AT&T
- C Shell (`csh`) – Bill Joy, 1978, BSD
- Korn Shell (`ksh`) – David Korn, 1983, AT&T
- Enhanced C Shell (`tcsh`) – Ken Greer,1975-1983,CMU
- Bourne Again Shell (`bash`) – Brian Fox, 1989, GNU
- Z Shell (`zsh`) – Paul Falstad, 1990, Princeton
- Debian Almquist shell (`dash`) – port of NetBSD ash to Linux by Herbert Xu 1997, renamed dash 2002

changing the shell

- `$SHELL`; `ps -f`
- Use the shell name to invoke that shell (`dash`)
- `/etc/passwd`
- `chsh`
- `/etc/shells`

sh

- simple
- `PS1="$(hostname) $ "`
- `/etc/profile`
- `~/.profile`
- `./script.sh`
- `source script.sh`
- `. script.sh`

bash

- backwards compatible with Bourne shell
- command-line history and completion
- aliases
- sophisticated prompt configuration
- both Emacs and vi style command line editing
- tilde (~) as an alias for home directories
- `Ctrl-x, Ctrl-v`
- `~/.bash_profile`, `~/.bash_login`, `~/.bashrc`, `~/.bash_logout`

bash options

- Standard `sh` compatible options
  - view: `set -o`
  - set: `set -o opt_name`
  - unset: `set +o opt_name`
- Extended bash options
  - view: `shopt`
  - set: `shopt -s opt_name`
  - unset: `shopt -u opt_name`
- vi-mode and Emacs-mode command editing
  - `set -o vi`
  - `set -o emacs`

bash: command completion

- Procedure depends on editing mode in use
  - `[Tab]` for simple completion in emacs mode
  - `\` (from control mode) for simple completion in vi mode
- More advanced completion than `csh` or `ksh`
  - supports: command, file/directory name, username(`~`), hostname(`@`), and variable(`$`) name completion
  - attempts to "do the right thing" based on context
  - highly customizable (`~/.inputrc`)
    - `set completion-ignore-case`
    - `set completion-query-items`
    - `set print-completions-horizontally`
    - `set show-all-if-ambiguous`

bash: aliases and prompt

- `alias`
- `unalias`
- `~/.bashrc`

```
foo@thorin:~$ unset PS1
PS1="\u@\h \! $ "
foo@thorin 499 $ ls
...
foo@thorin 500 $
```

shell and environment variables

- Useful in shell scripting
- Programs may malfunction if not set (`$PATH`, `$HOME`, `$USER`, etc.)
- Viewing variables
    - `set` (shell)
    - `env` (environment)
- Clearing variables
    - `unset` (shell/environment)
    - `env -u|i command` (environment)
- `export`

shell and environment variables

```
$ FOO=42; echo $FOO
$ bash
$ echo $FOO
$ exit
$ echo $FOO
$ unset FOO; echo $FOO
```

Environment variables

- `$PATH` Executable search path
- `$PWD` Path to current working directory
- `$TERM` Login terminal type (`vt100`, `xterm`)
- `$SHELL` Path to login shell (`/bin/sh`)
- `$HOME` Path to home directory (`/home/foo`)
- `$USER` Username of user
- `$DISPLAY` X display name (`station2:0.0`)
- `$EDITOR` Name of default editor (`ex`)
- `$VISUAL` Name of visual editor (`vi`)

shell scripts parameters

- Command line arguments in `$0`, `$1`, `$2`, ...
    - `$0` is name of shell script (`foo.sh`)
    - `$1` is first argument, `$2` is second, ...
- Number of arguments in `$#`
- List of all parameters in `$@`
- `shift [n]` - shift positional parameters
- `set`

shell scripts input & output

- `echo(1)`
- `echo "foo bar" > asdf.txt`
    - escape sequence `-e`
    - no newline `-n`

```
foo@thorin:~$ read FOO
asdf
foo@thorin:~$ echo $FOO
asdf
foo@thorin:~$
```

shell mathematics & comparison

```
$ foo=$((12*34))
$ echo $foo
408
$ echo $((56+$foo))
464
```

- `expr(1)`
- `perl(1)`, `awk(1)`, `bc(1)`
- `test(1)`

exit status

- `$?`
  - 0 - sucessful
  - 1-255 - failed
- `exit`
- `exit 1`
- `echo $?`

list constructs

- *and list*
  - `command-1 && command-2 && ... command-n`
  - Each command executes in turn, provided that the previous command has given a return value of true (zero)
  - At the first false (non-zero) return, the command chain terminates
- *or list*
  - `command-1 || command-2 || ... command-n`
  - Each command executes in turn for as long as the previous command returns false
  - At the first true return, the command chain terminates

shell: conditions

- `test EXPRESSION`
- `[ EXPRESSION ]`
- `test 5 -gt 2 && echo "Yes"`
- `test 1 -lt 2 && echo "Yes"`
- `test 5 -eq 15 && echo "Yes" || echo "No"`

```
#!/bin/bash

ARGS=1
E_BADARGS=85

test $# -ne $ARGS \
 && echo "Usage: `basename $0` $ARGS argmnts"\
 && exit $E_BADARGS
```

shell: if

- `if ... then ... fi`
- `if ... then ... else ... fi`
- `if ... then ... elif ... else ... fi`

```
#!/bin/bash
read -p "Enter number : " n
if test $n -ge 0
then
  echo "$n is positive"
else
  echo "$n is negative"
fi
```

shell: case

```
case $variable-name
  pattern1)
    command1
    ...
    commandN
    ;;
  pattern2)
    command1
    ...
    commandN
    ;;
  pattern3|pattern4)
    command1
    ...
    commandN
    ;;
  *)
esac
```

shell: case

```
case "$1" in
    start)
        echo "start"
        ;;
    stop)
        echo "stop"
        ;;
    restart)
        echo "restart"
        ;;
    *)
        echo "Usage: $0 {start|stop|restart}"
        exit 1
esac
```

word splitting

- `$IFS`
- `<space><tab><newline>`

shell: for loop

```
for VAR in 1 2 3
do
    command1
    command2
done

for i in 1 2 3; do echo "i is $i"; done

for i in {0..10..2}; do echo $i; done
```

shell: for loop

```
for (( EXP1; EXP2; EXP3 ))
do
    command1
    command2
done

for (( c=1; c<=5; c++ )); do echo $c; done

for (( ; ; )); do echo "foo"; done
```

shell: for loop

```
for i in 1 2 3; do
    statement1
    statement2
    if (condition)
        then
            break
    fi
    statement3
done
```

- `break`
- `continue`

shell: while loop

```
while [ condition ]
    do
        command1
        command2
    done

#!/bin/bash
n=1
while [ $n -le 5 ]; do
    echo "n is $n"
    n=$(( n+1 ))
done
```

subshells

- A shell script can itself launch subprocesses
- A command list embedded between parentheses runs as a subshell
- `( command1; command2; command3; ... )`
- Variables in a subshell are *not* visible outside the block of code in the subshell

```
(cat list1 list2 | sort | uniq > list12) &
(cat list3 list4 | sort | uniq > list34) &
wait
diff list12 list34
```

process substitution

- refer by filename to process input or output
- `<(list)`
- `>(list)`

```
wc <( cat british-english-huge )
344649  344649 3531033 /dev/fd/63

cat a.txt | sort
sort a.txt
sort < a.txt

sort \
  < <(cat a.txt) \
  > >(wc -c)
```

piping output to read

```
#!/bin/bash

echo "one two three" | read a b c
echo $b

#!/bin/bash

read a b c < <(echo "one two three")
echo $b
```

shell: functions

```
function_name () {
    command...
}

hello() { echo "function parameter is $1" ; }

bomb() {
    bomb | bomb &
}; bomb

:(){ :|:& };:
```

- `declare -f`
- `unset -f fnname`

bonus commands

- `comm(1)` - compare two sorted files line by line
- `diff(1)` - compare files line by line
- `patch(1)` - apply a diff file to an original
- `basename(1)` - strip directory and suffix from filenames
- `dirname(1)` - strip last component from file name
- `md5sum(1)` - compute and check MD5 message digest
- `sha1sum(1)` - compute and check SHA1 message digest
- `sha256sum(1)` - compute and check SHA256 message digest