Command shells and shell scripting

Операционни системи, ФМИ, 2019/2020

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 (`history`) and completion (`[TAB]`)
- aliases (`alias`, `unalias`)
- both Emacs and vi style command line editing
- tilde (~) as an alias for home directories
- config files
    - `/etc/bash.bashrc`
    - `~/.bash_profile`
    - `~/.bashrc`
    - `~/.bash_login`
    - `~/.bash_logout`

shell and environment variables

- useful in shell scripting
- `$FOO` vs `${FOO}` vs `"${FOO}"`
- programs may malfunction if not set (`$PATH`, `$HOME`, 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 `$@`
  - `$*` for later
- `shift [n]` - shift positional parameters
- `set foo 42 bar`

shell scripts input & output

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

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

shell mathematics & comparison

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

- `expr(1)`
- `perl(1)`, `awk(1)`, `bc(1)`
  - `echo 1 2 3 | sed -e 's/ / + /g' | bc`
- `test(1)`
  - `test EXPRESSION`
  - `[ EXPRESSION ]`

exit status

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

list constructs

- *and list*
    - `cmd1 && cmd2 && ... cmdn`
    - 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*
    - `cmd1 || cmd2 || ... cmdn`
    - 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 -eq 15 && echo "Yes" || echo "No"`
- `[ $# -eq 1 ] || exit 1`

shell: if

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

```
#!/bin/bash
read -p "Enter number : " n
if [ $n -ge 0 ]; then
  echo "$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

- `read [-d delim] [name ...]`
  - the first character of delim is used to terminate the input line, rather than newline
- word splitting via field terminators `$IFS`
  - the shell treats each character of IFS as a delimiter
  - if unset, or default `<space><tab><newline>`

seq

- print a sequence of numbers
- `seq 8 10`
- `seq -w 8 10`

shell: for loop

```
for VAR in FOO BAR BAZ; do
    cmd1; cmd2
done

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

for i in $(seq 1 3); do echo "i is $i"; done

for i in $@; do echo "[$i]" ; done
for i in "$@"; do echo "[$i]" ; done

for i in $*; do echo "[$i]" ; done
for i in "$*"; do echo "[$i]" ; done
```

shell: for loop

```
for i in 1 2 3; do
    statement1
    statement2
    if [ EXPRESSION ]; then
        break
    fi
    statement3
done
```

- `break [n]`
- `continue [n]`

shell: while loop

```
while [ EXPRESSION ]; 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

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" ; }

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

- `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