CS2720 Practical Software Development

Scripting Tutorial Srping 2011

Instructor: Rex Forsyth
Office: C-558
E-mail: forsyth@cs.uleth.ca
Tel: 329-2496

Tutorial Web Page: http://www.cs.uleth.ca/~forsyth/cs2720/lab/lab.html

Procedure for testing our project

- checkout current version from the repository to a testing directory
- change to the testing directory
- build the test program (make)
- if build is successful, run the test and record the results
- if build is not successful, record an error message

We would like to automate this process

- place appropriate commands in a shell script
- then we just need to run the script
- we would like to have the system automatically run the script for us at a convenient time and send us a message.

Shell Scripting

1. the first line of each script must be

   `#!/bin/sh`

2. comments start with a #
3. there is a need to hide things from the shell
   - `\` – next char is not treated as a shell command
   - `'  '` – all text between single quotes is ignored by the shell
   - `"  "` – most text between double quotes is ignored, some exceptions include `$ \`
4. command substitution
   - `$(  )`
   - result of the command inside the brackets is returned
   - eg `$(date)`
     - use **man** to find out about date eg. `man date`
     - can be used to create unique file names

5. `test` – returns 0 if successful(true), not 0 if failure(false)
   often use `[ ]` instead but remember the spaces

6. control flow
   - selection
     – **if**

```
if commandList
then

fi
```

```
if commandList
then

else

fi
```

```
if commandList
then

elif commandList
then

else

fi
```

makes the selection based on the exit status of the last command in the command List

- **case**

```
case string in
   pattern1) commands ;;
   pattern2) commands ;;

   ...
   *) commands ;;        #default case
esac
```

patterns may include wildcards as the shell does

you may have more than 1 pattern for a set of commands; separate with |

- repetition
  - **while**

    ```
    while commandList
    do


    done
    ```

    executes the statements between **do** and **done** as long as the condition is true. The condition is based on the exit status of the last command in the commandList.
  - **for**

    ```
    for variable in list
    do


    done
    ```

    executes the statements between **do** and **done** once for each item in the list

both **if** and **while** may use a **test** as the commandList. eg

```
if test -r filename
then
   echo "The file exists and can be read"
fi
```

or

```
if [ -r filename ]
then
   echo "The file exists and can be read"
fi
```

and

```
while [ expression ]
do

  ...

done
```

7. the ; is used to simulate a new line

   eg

   ```
   if [ -r filename ] ; then; echo "The file exists"; fi
   ```

8. shell variables
   - syntax `VAR=value`
   - no spaces allowed
   - variable names may have letters only
   - $VAR – evaluates the variable VAR

9. positional parameters, (read only)
    - $0 – the command name
    - $1 through $9 – the parameters
10. Others
    - $# – the number of parameters
    - $* – equivalent to $1 $2 $3 ...
    - $@ – equivalent to $1 $2 $3 ...
    - “$*” – equivalent to “$1 $2 $3 ...”
    - “$@” – equivalent to “$1” “$2” “$3” ...
    - $? – the exit status of the last command
    - $$ – the pid of the current command; used for unique filenames
    - $! – pid of the last background command

11. using files
    - often refered to by the file number
      - 0 – standard in
      - 1 – standard out
      - 2 – standard error
    - /dev/null – no where
    - redirection
      - `> filename` – write stdout to filename; same as `1 > filename`
      - `>> filename` – append stdout to filename
      - `>& m` – write stdout to file m; same as `1 >& m`
      - `< filename` – read stdin from filename; same as `0 < filename`
      - `<& m` – read stdin from file m; same as `0 <& m`
      - `a <& m` – read a from file m
      - `a >& m` – write a to file m

- to automate the testing process, write a script to do the desired steps
- make the script executable using the `chmod` command.
  eg `chmod u+x` *scriptname*
- to do the tests, we just have to run the script

```
#!/bin/sh
FROM=http://svnhost/rex/cppunit
TO="$HOME/testdir"
FNAME=$(date +%y%m%d)
ulimit -t 10
if [ $# -eq 2 ]
then
   TO="$1"
   FROM="$2"
elif [ $# -eq 1 ]
then
   TO="$1"
fi

svn checkout "$FROM" "$TO"
if [ $? -eq 0 ]
then
   cd "$TO"
   make clean-all
   make -k > "$FNAME.comp.$$" 2>&1
   if [ $? -eq 0 ]
   then
      ./ptest > "$FNAME.res.$$"
   fi
   cd -
fi
```

- now get the system to run the script as a **cron** job
- need to create a cron table
  - cron tables are specific to a machine
  - to modify a cron table, you must be logged on the machine where it was created.
- use the `crontab` command; see `man -s5 crontab`

crontab

options

- `crontab -l` – display your cron table for this machine
- `crontab -r` – remove your cron table from this machine
- `crontab -e` – create or edit your cron table on this machine

You may put two things into a cron table, environment variable settings and cron commands. Comments begin with a #

- environment variable settings

```
#required; use the most portable shell
SHELL = /bin/sh
#optional, if not provided, cron mails to owner of cron table
MAILTO = forr9000
```

- cron commands
  - consist of 5 fields and a command to be executed

```
minute  hour monthDay month dayOfWeek command
0-59    0-23  0-31    1-12   0-7
```

  - for dayOfWeek, 0 and 7 are both Sunday
  - you may use ranges; * is the same as first-last
  - 3,6,9,12
  - 3-8, 15-20
  - 1-9/2
  - 0-9/2
  - */2
  - names can be used for month and dayOfWeek,use first 3 letters

eg

```
SHELL = /bin/sh
15 2 * * * /home/forsyth/testScript
```