Introduction to Shell Programming

- what is shell programming?
- about cygwin
- review of basic UNIX™
- pipelines of commands
- about shell scripts
- some new commands
- variables
- parameters and shift
- command substitution
- `cut`
- `if-then-else-fi`
- `for-in-do-done`
- `sed`
- a final example
- gotchas
- exercises
- references

conducted by

Mike Kupferschmid

Scientific Programming Consultant

VCC 322, x6558, `kupfem@rpi.edu`

What is Shell Programming?

- putting UNIX™ commands in a file
- almost always special-purpose code
- often one-time code
- seldom used where speed is important
- often used to manipulate files

About `cygwin`

a good but not perfect emulation of unix
- included in standard RPI laptop image
- to download (long) go to `www.cygwin.com`

if you want real unix get Linux (site licensed)
- it is possible to dual-boot Linux with Windows
- ask at the Help Desk
- watch for an ACM installfest

starting `cygwin`
- start → All Programs → Cygwin → XTerm
- opens unix window with command prompt
  - `$`
- `RCShome` is a link to your RCS home directory

to print a file from `cygwin`
- open the file with `notepad` *filename*
- use the `notepad` print function

to cut and paste you need a 3-button mouse

`cygwin` names executables `a.exe`

Review of Basic UNIX™

familiar commands often useful in shell scripts

`cat`	concatenate files
`cp`	copy a file
`date`	print the date and time
`grep`	scan for a string
`head`	show first lines of a file
`tail`	show last lines of a file
`mv`	move or rename a file
`rm -f`	remove files (silently)
`wc`	count lines, words, characters
	`wc` output format varies between systems

path names of files and directories

`schedule`	relative
`/home/37/jones/schedule`	absolute

wild cards in filenames

`*` matches zero or more characters
`?` matches exactly 1 character

redirection

`>` redirects std-out to a file
`>>` appends std-out to a file
`<` redirects std-in from a file

Pipelines of Commands

send std-out of one command to std-in of another

`look e`

shows spelling words that begin with e

`look e | more`

displays the words one page at a time

often use `echo` to feed a pipeline

`echo "count me" | wc`

prints `1 2 9`

`echo * | wc -w`

counts files in current directory

About Shell Scripts

type shell program text in a file using an editor:

```
#! /bin/sh
#  this is a comment
    body of program
    to continue a line append \
    this is the rest of the continued line
exit 0
```

`chmod +x` *scriptfile* — make the file executable

not needed in `cygwin`

scriptfile — execute the program

`sh -v` *scriptfile* — print input lines as read

`sh -x` *scriptfile* — print commands as executed

shell programs often use temporary files in `/tmp`
and send unwanted outputs to `/dev/null`

Some New Commands Useful in Shell Programs

`basename`	extract file name from path name
`cmp -s`	compare files (silently)
`cut`	extract selected parts of a line
`expr`	evaluate an expression
`mail`	send email (not in `cygwin`)
`sed -e`	stream editor
`sleep`	suspend execution for given time
`tr`	translate characters
`true`, `false`	provide truth values
`whoami`	print current username
`head -1`	read a line from the keyboard

Some Example Scripts

In the following examples, the text of the shell script is shown on top, and the result of executing it interactively is shown below. The text of each example script is in a file available for downloading, so that you can try the scripts without having to type them in. The name of each file is given in a comment in each script.

These examples show the scripts being executed in the directory /home/mike/Classes/shell. When you run a script in your own directory, some of the names appearing in the output will be different.

```
#!/bin/sh
#  hi
echo "Hello, world!"
exit 0


unix[1] hi
Hello, world!
unix[2]
```

```
#!/bin/sh
#  himike
name=Mike
echo "Hello, $name!"
exit 0


unix[3] himike
Hello, Mike!
unix[4]
```

```
#!/bin/sh
#  rem
rm junk
echo "The return code from rm was $?"
exit 0
```

```
unix[5] touch junk
unix[6] rem
The return code from rm was 0
unix[7] rem
rm: junk: No such file or directory
The return code from rm was 2
```

```
#!/bin/sh
#  quiet
rm junk 2> /dev/null
echo "The return code from rm was $?"
exit 0
```

```
unix[8] touch junk
unix[9] quiet
The return code from rm was 0
unix[10] quiet
The return code from rm was 2
```

```
#!/bin/sh
#  pars
echo "There are $# parameters."
echo "The parameters are $@"
echo "The script name is $0"
echo "The first parameter is $1"
echo "The second parameter is $2"
exit 0
```

```
unix[11] pars apple orange
There are 2 parameters.
The parameters are apple orange
The script name is ./pars
The first parameter is apple
The second parameter is orange
unix[12]
```

```
#!/bin/sh
#  shifter
echo $1
shift
echo $1
shift
echo $1
shift
echo $1
exit 0
```

```
unix[13] shifter one two three four five
one
two
three
four
unix[14] shifter one two three
one
two
three

unix[15] shifter one two
one
two

shift: shift count must be <= $#

unix[16]
```

```
#!/bin/sh
#  sorter
rm -f /tmp/sorted
sort $1 > /tmp/sorted
cp /tmp/sorted $1
rm -f /tmp/sorted
exit 0
```

```
unix[17] more names
Jeff
Alan
Nancy
Yossl
Scott
Harriet
Chris
unix[18] sorter names
unix[19] more names
Alan
Chris
Harriet
Jeff
Nancy
Scott
Yossl
unix[20]
```

```
#!/bin/sh
#  hiyou
name=‘whoami‘
echo "Hello, $name!"
exit 0


unix[21] hiyou
Hello, kupfem!
unix[22]
```

```
#!/bin/sh
#  hiyou2
echo "Hello, `whoami`!"
exit 0


unix[23] hiyou2
Hello, kupfem!
unix[24]
```

```
#!/bin/sh
#  countem
echo "File \"$1\" contains \
exactly ‘wc $1 | cut -c6-7‘ lines."
exit 0


unix[25] countem text
File "text" contains exactly 21 lines.
unix[26]
```

cut

reads std-in, extracts selected fields, writes std-out

`cut -c1,4,7`	characters 1, 4, and 7
`cut -c1-3,8`	characters 1 thru 3, and 8
`cut -c-5,10`	characters 1 thru 5, and 10
`cut -c3-`	characters 3 thru last
`cut -f1,4,7`	tab-separated fields 1, 4, and 7
`cut -d" " -f1`	blank-separated field 1

`echo a.b.c | cut -d"." -f2` yields `b`

```
#!/bin/sh
#  compile
if [ "$SRCDIR" = "" ]
then
  echo "using default source directory"
  SRCDIR=$HOME/src
else
  echo "using source directory $SRCDIR"
fi
g77 $SRCDIR/$1
exit $?
```

```
unix[27] export SRCDIR=‘pwd‘
unix[28] compile hello.f
using source directory /home/mike/Classes/shell
unix[29] echo $?
0
unix[30] a.out
 hello
unix[31] export SRCDIR=""
unix[32] compile hello.f
using default source directory
g77: /home/mike/src/hello.f:
     No such file or directory
unix[33] echo $?
1
unix[34]
```

```
#!/bin/sh
#  finder
grep $1 text > /dev/null
if [ $? -eq 0 ]
then
  echo "found"
fi
exit 0


unix[35] finder ancient
found
unix[36] finder modern
unix[37]
```

```
#!/bin/sh
#  compares
echo "true yields 0, false yields 1"
x="005"
[ "$x" = "005" ]
echo "Are strings 005 and 005 equal? $?"
[ "$x" = "5" ]
echo "Are strings 005 and 5 equal? $?"
[ $x -eq 005 ]
echo "Are integers 005 and 005 equal? $?"
[ $x -eq 5 ]
echo "Are integers 005 and 5 equal? $?"
exit 0
```

```
unix[38] compares
true yields 0, false yields 1
Are strings 005 and 005 equal? 0
Are strings 005 and 5 equal? 1
Are integers 005 and 005 equal? 0
Are integers 005 and 5 equal? 0
unix[39]
```

```
#!/bin/sh
#  empty
if [ -s $1 ]
then
  echo "The file $1 has contents."
  exit 0
else
  echo "The file $1 is absent or empty."
  exit 1
fi


unix[40] empty text
The file text has contents.
unix[41] empty xxxx
The file xxxx is absent or empty.
unix[42] echo $?
1
unix[43]
```

```
#!/bin/sh
#  adder
sum=0
for x in $@
do
  sum=‘expr $sum + $x‘
done
echo "The sum is $sum."
exit 0


unix[44] adder 1 2 3 4 5
The sum is 15.
unix[45]
```

```
#!/bin/sh
#  fixfor
for fyle in *.for
do
  new=‘echo $fyle | sed -e"s/\.for$/\.f/"‘
  mv $fyle $new
done
exit 0
```

```
unix[46] ls *.for
a.for b.for pgm.for xyz.w.for
unix[47] fixfor
unix[48] ls *.f
a.f   b.f   pgm.f   xyz.w.f
```

```
#!/bin/sh
#  suffix
for fyle in *.$1
do
  new=‘echo $fyle | sed -e"s/\.$1$/\.$2/"‘
  mv $fyle $new
done
exit 0
```

```
unix[49] ls *.f
a.f   b.f   pgm.f   xyz.w.f
unix[50] suffix f for
unix[51] ls *.for
a.for b.for pgm.for xyz.w.for
unix[52]
```

sed

reads std-in, edits line(s), writes std-out

`sed -e"s/old/new/"` replace first `old` by `new`
`sed -e"s/old/new/g"` replace each `old` by `new`

`old` can be a regular expression

`^`	matches the beginning of the line
`$`	matches the end of the line
`.`	matches any single character
`.*`	matches zero or more characters
`[tT]`	matches `t` or `T`

escape these and other special characters with \

```
unix[53] echo banana | sed -e"s/a$/\.x/"
banan.x
unix[54] more fruit
xapple
xpear
xplum
xcherry
unix[55] sed -e"s/^x/ /" < fruit
 apple
 pear
 plum
 cherry
```

A Final Example

```
#! /bin/sh
#  list names of all files containing given words

if [ $# -eq 0 ]
then
  echo "findtext word1 word2 word3 ..."
  echo "lists names of files containing all given words"
  exit 1
fi

for fyle in *
do
  bad=0
  for word in $*
  do
    grep $word $fyle > /dev/null 2> /dev/null
    if [ $? -ne 0 ]
    then
      bad=1
      break
    fi
  done
  if [ $bad -eq 0 ]
  then
    echo $fyle
  fi
done
exit 0
```

Gotchas

Never use `test` as the name of a variable or a shell script file.

When using `=` as an assignment operator, do not put blanks around it.

When using `=` as a comparison operator, you *must* put blanks around it.

When using `if [ ]` put spaces around the brackets (except after `]` when it is the last character on the line).

Exercises

some hints are given in the files
`exer1, exer2, exer2.aix, and exer3.`

1. Write a script that counts files. (a) First make it count the files in the current directory. (b) Now modify your script to accept a parameter that is the name of a directory, and count the files in that directory. Try this version on the current directory (`.`) and on the `/afs/rpi.edu/campus/doc` directory. (c) Further modify your script so that if it is invoked without a parameter it prints out an explanation of how to use it.

2. If the `ls` command is given the name of a single extant file it merely prints that filename back out. (a) Write a script `myls` that behaves like `ls` except that when a single filename parameter is supplied it produces the output that `ls -l` would give for the file. (b) Revise your script so that when a single filename parameter is given the output produced is the filename followed by the date and time of its most recent change and then the size of the file in bytes.

3. A script `isyes` is required that sets its exit code to 0 if its parameter is some variation of y or yes, and to 1 otherwise. (a) Assume the only acceptable parameter values meaning "yes" are y, yes, Y, and YES, and solve the problem using only shell programming features we have discussed. (b) Simplify and generalize your script by using `tr a-z A-Z`, which reads from std-in, translates to upper case, and writes to std-out.

4. Write a script that adds up the sizes reported by `ls` for the files in the current directory. The script should print out only the total number of bytes used.

References

UNIX™ Shell Programming, Revised Edition, by Stephen G. Kochan and Patrick H. Wood, Hayden Books, 1990, ISBN 0-672-48448-X.

The UNIX™ Programming Environment, by Brian W. Kernighan and Rob Pike, Prentice Hall, 1984, ISBN 0-13-937681-X.

`sed` & `awk`, by Dale Dougherty, O'Reilly & Associates, 1991, ISBN 0-937175-59-5.

Mastering Regular Expressions, by Jeffrey E. F. Friedl, O'Reilly & Associates, 1997, ISBN 1-56592-257-3.