ELEC 377 – Operating Systems

Week 7 – Class 3

Last Class

- Started Shell Programming
- ◊ Variables
- ◊ Command Syntax

Today

- Continue Shell Programming
- ◊ Conditions..
- ◊ Key Commands (find, grep awk)

Control flow - if statement

if *test command* ; **then**
  true commands
elif *another test command* ; **then**
  otherwise true commands
else
  false commands
fi

◊ check the exit status of the test commands to determine if the statements are executed

Control flow - if statement

```
if test -f xxx ; then                    xxx exists and is a file
   cat xxx
elif test -d yyy ; then
   ls yyy
else
   echo “Neither xxx nor yyy exist!!”
fi
```

◊ test -f *name* returns 0 (success) if file *name* exists, 1 otherwise.

◊ test -d *name* returns 0 (success) if directory *name* exits, 1 otherwise

Control Flow - while

```
((i = 1))
while (( i < 10)) ; do
   echo $i
  (( i++ ))
done
```

◊ note that ((...)) does not *require* $ in front of variables (also in if statements too!!)

◊ [[ ]] and commands with exit status can also be used for while condition.

Control Flow - for

for var **in** wordlist ; **do** commands; **done**

for p **in** /proc/[0-9]* ; **do**
   echo -n “$p: “   *-n = no newline, also space in string*
   grep ‘State’ $p/status   *find line with State*
done

output:

/proc/1: State: S (sleeping)
….

Control Flow - shift

shift shifts the position arguments
$2 → $1, $3 → $2, $4 → $3, etc.

```
while (( $# > 0 ))              while num args > 0
do
   echo $1                          print first arg
   shift                            discard first and shift
    rest
done
```

Find Command

- find command

  ◊ finds files or directories that match a pattern

  ◊ find /home/student -name ‘*.c’ -or -name ‘*.h” -print

```
/home/student/trd/lab0/lab0mod.c
/home/student/trd/lab0/lab0user.c
/home/student/trd/lab1/lab1.c
....
/home/student/trd/lab4/common.h
...
```

For and Find Command - Friends!!

- want to print all c (.c and .h) files in a particular directory (and any subdirectories)
  ◊ lpr *filename* *prints a file (or stdin if no file)*
  ◊ a2ps filename *converts file to postscript*

```
for i in `find /home/student -name ‘*.c’ -or -name ‘*.h” -print`
do
   a2ps $i | lpr        create a postscript version and print
done
```

◊ $(command) same as `command`

The path least taken..

- two other useful commands

  ◊ dirname *path* *directory name of path*

  ◊ dirname /a/b/c/defg.c

  /a/b/c

  ◊ dirname defg.c

  .

  ◊ basename path extension *base name of file*

  ◊ basename /a/b/c/defg.c .c

  defg

Find, Grep, Sed & awk

- the shell provides control and variables.

◊ control structures can use return value or output of any command

◊ rather than build everything into the shell language (early unix machines had 64 -256K of memory) the functionality was put into other programs.

◊ since the output of one can be the input of another, the result is a very flexible set of primitives. Much like the ability to wire together a set of primitive gates to build arbitrary complex circuits...

Find

- searching command that examines every file in a given list of directories.

◊ can print a list (use with *for* or with other commands)

◊ can execute a command on each file that matches

find *list_of_directories patterns_and_actions*

```
    find /home/student -name ‘lab3.c’
    find /home/student -name ‘*.c‘          quotes!!
    find /home/student -iname ‘*.txt‘case insensitive
```

Find

- multiple conditions can be applied together

-type *code*

codes: f = file, d = directory l = link (and others)

find /home/student -type f *all files in /home/student*

find /home/student -type d *all dirs in /home/student*

find configuration files for apache

find /etc/ -type f -name ‘apache*’

Find - size and time

- size

-size *sizespec*	
256c	*256 bytes*
+256c	*> 256 bytes*
-1025m	< 1025 megabytes

- time

-mtime +4	*modified more than 4 days ago*
-atime +4	*accessed more than 4 days ago*
-mmin *num*	*modified num minutes ago*
-amin *num*	*??*

Find - actions

- -print *default if no action provided*

◊ print the name of the file/directory

- -delete

◊ remove the file/directory

```
find / -name ‘core*’ -size >2M -atime +30 -delete
find / -name ‘core*’ -size >2M -atime +30 -print -delete
```

- -exec

◊ execute an arbitrary command

◊ {} matches the file name

◊ \; ends the command

```
find / -name ‘core*’ -size >2M -atime +30 -exec rm -f {} \;
```

Find - multiple conditions

- default is that conditions are ‘anded’ together

```
find /home/student -type f -name ‘*.c’
```

- but what if you want header files too?

```
find /home/student -type f -name ‘*.c’ -or
-name ‘*.h’
```

Grep

- generalized searching tool

  ◊ name comes from editor command (still available in vi)

g/re/p   *global regular expression print*

- searches line by line through files to find a regular expression (*looks through stdin if no files given*)

  ◊ powerful tool for finding things in code someone else has written...

```
grep pattern file1 file2 .....
grep init_module *.c
```

Grep - flags and exit codes

- -i *case insensitive match*
- -w *match must be entire word*
- -v *lines that do **not** match*
- -H *shows names of files*
- -h *do not show names of files*
- -l *only show the file names (not the lines)*
- -q *print nothing (only set exit status for use with conditions) (same as grep ... > /dev/null)*

exit status

0 - at least one match (i.e. success)

1 - no match (i.e. failure)

2 - error (i.e. failure)

Regular Expressions

- three types of regular expressions

◊ Globbing (file patterns such as *.c) used by shell patterns and by -name in find.

◊ normal regular expressions used by default by grep, sed and awk and by many editor such as vi

◊ extended regular expressions (-E flag on grep/sed)

Regular Expressions

- normal regular expressions

◊ . *(dot) matches any character*

grep “t.e” *matches the, tye, tie, txe, t#e ....*

◊ ^ *matches the beginning of the line*

grep “^init” *does not match “module_init”*

◊ $ matches the end of the line

grep “xyzzy$”

◊ \negates special meaning of next character

grep “\\.length” *search for “.length”, not “alength”*

grep “\\$1” *search for “$1”*

Regular Expressions

- normal regular expressions

◊ * *repeat previous regular expression*

grep ‘xy*z‘ *search for xz, xyz, xyyz, xyyyz ...*

◊ [] *list of characters (like globbing)*

grep ‘t[hie]e’ *search for “the” “tie” “tee”*

◊ [^] *anything not in the list of characters*

grep ‘/*[^*]‘ *search for non-java doc comments*

Regular Expressions

- normal regular expressions

◊ \{\} *match a specific number of previous match*

grep “xy\{3\}z” *matches xyyyz*

grep “xy\{2,\}z” *matches xyyz xyyyz, xyyyyz, ...*

grep “xy\{2,3\}z” *matches xyyz xyyyz*

how do you match a 3 digit number?

Sed

- how do you make changes to a file inside of a shell script

◊ vi < keystopress.txt

-- what about errors, conditional changing, etc.?

- sed - stream editor.

◊ reads from standard in (or a file), writes to standard out.

◊ edits one line at a time from a given edit script.

```
sed -e ‘s/foo/bar/’ infile.txt > outfile.txt
sed -f  sedfile.txt infile.txt > outfile.txt
```

Sed - patterns

- multiple matches

sed -e ‘s/foo/bar/’

foo bar bat foo -> bar bar bat foo *first match*

sed -e ‘s/foo/bar/g’

foo bar bat foo -> bar bar bat bar *all matches*

sed -e ‘s/foo/bar/2’

foo bar bat foo -> foo bar bat bar *2nd match*

Sed - patterns

- multiple matches

sed -e ‘s/x.*z/wand/’

xyyzyyzmm -> wandmm *longest match*

make any regular expressions as specific as possible

Sed - edit commands

- can have more than one edit command

◊ applied in order

```
sed -e s/a/b/ -e s/b/c/
```

foobar

remember first match

what if second match was s/b/c/g?

Sed - edit commands

- can have more than one edit command

◊ applied in order

```
sed -e s/a/b/ -e s/b/c/
```

foobar -> foobbr

remember first match

what if second match was   s/b/c/g?

Sed - edit commands

- can have more than one edit command

◊ applied in order

```
sed -e s/a/b/ -e s/b/c/
```

foobar -> foobbr -> foocbr *remember first match*

what if second match was s/b/c/g?

Sed - patterns

- use backslash(\) to negate special meanings.

sed -e ‘s/\//#/‘ *turn / into #*

◊ if going to be using a lot of slashes, sed uses the character following the s command to be the delimiter

sed -e ‘sx/x#x’ *turn / into #*

- ‘&’ in the replacement matches the pattern

sed -e ‘s/xy*z/---&---/’

the xyzzy fell -> the ---xyzzy--- fell

Sed - patterns

- & is the entire pattern. What about just part of the pattern?

◊ \(\) in pattern tags the part of the pattern

```
sed -e ‘s/\(a*\)\(.*\)\(b*\)/\3\2\1/’
```

```
weroiusoiuaaaxoiqweroistybbqoiuqa
```

Sed - patterns

- & is the entire pattern. What about just part of the pattern?

◊ \(\) in pattern tags the part of the pattern

sed -e ‘s/\(a*\)\(.*\)\(b*\)/\3\2\1/’

weroiusoiu*aaaxoiqweroistybb*qoiuqa

Sed - patterns

- & is the entire pattern. What about just part of the pattern?

◊ \(\) in pattern tags the part of the pattern

sed -e ‘s/\(a*\)\(.*\)\(b*\)/\3\2\1/’

weroiusoiu***aaa****xoiqweroisty****bb***qoiuqa

Sed - patterns

- & is the entire pattern. What about just part of the pattern?

◊ \(\) in pattern tags the part of the pattern

```
sed -e ‘s/\(a*\)\(.*\)\(b*\)/\3\2\1/’
```

weroiusoiu***bb****xoiqweroisty****aaa***qoiuqa

Sed - patterns

- & is the entire pattern. What about just part of the pattern?

◊ \(\) in pattern tags the part of the pattern

sed -e ‘s/\(a*\)\(.*\)\(b*\)/\3\2\1/’

weroiusoiu***bb****xoiqweroisty****aaa***qoiuqa

sed -e ‘s/b*\(.*\)a*/\1/

weroiusoiu*xoiqweroisty*qoiuqa

Sed

- What about only changing some lines??

◊ sed commands have the form:

startaddress,*endaddress* command

◊ by default, start address = first line (i.e. 1)

◊ by default, end address = last line (i.e. $)

s/a/b/ -> 1,$ s/a/b/

◊ only on lines 5 to 7

sed -e ‘5,7 s/a/b/’

◊ only on line 10

sed -e ‘10 s/a/b/’

Sed - address patterns

- addresses can also be patterns

/init_module/ sx()x(/*noparms*/)x

- sed -e ‘/foo/,/bar/ s/a/b/’
  for all lines between a line starting with foo and a line ending with bar, substitute b for a.

Sed - other commands

- d - delete

  sed -e ‘/foo/ d‘   *delete all lines containing foo*
- i - insert before

  sed -f script.txt

  script.txt:

  /init-module/ i\

  /* initialization routine */

  put a comment before the init_module routine
- a - append after

Shell Scripting Here Documents

- Sometimes the input is a constant, and you don't want to distribute another file. Here documents are documents embedded in the script:

... scripting commands that compute variable var1...

sed -e "s/XXX/$var1/ <<END1

This is some constant text embedded in

the shell script. The value of var1 is XXX.

END1

.... more scripting commands ...

looks for a line containing only *the marker given after* <<

Other Useful tidbits...

- tr command, translates characters in input to output, also knows printf sequences..
  ◊ only reads from stdin and writes to stdout
  -- use redirection to use files
  ◊ tr ‘[a-zA-Z]’ ‘[n-za-mN-ZA-M]’ < infile > outfile
  ◊ tr ‘\r’ ‘x‘   *convert newlines into x characters*
- $$ - variable containing process id of shell
  - useful for temporary files
  grep ‘pattern’ > /tmp/$$.temp1
  sed ‘s/pattern/replacement’ < /tmp/$$.temp1
  rm /tmp/$$.temp1

awk - report language

- to complex to go into here in detail.
  - ◊ processes line at a time
  - ◊ can do arithmetic, special formatting
  - ◊ examples: add up the numbers in a particular column of a text file
  - ◊ calculate statistics of patterns in a file
  - ◊ often used for quick formatting in a shell script
  - ◊ $1, $2 refer to columns in the input

  awk ‘{ printf “fmt”, $1, $2, $3 }‘   *takes column input without formatting and prints formatted*