Introduction to the C Shell

What is the Shell? (Ch.6)

- A command-line interpreter program that is the interface between the user and the Operating System.
- The shell:
  - analyzes each command
  - determines what actions to be performed
  - performs the actions
- Example:

```
wc -l file1 > file2
```

csh Shell Facilities

- Automatic command searching (6.2)
- Input-output redirection (6.3)
- Pipelining commands (6.3)
- Command aliasing (6.5)
- Job control (6.4)
- Command history (6.5)
- Shell script files (Ch.7)

I/O Redirection (6.2)

- **stdin** (fd=0), **stdout** (fd=1), **stderr** (fd=2)
- Redirection examples: (`<, >, >>, >&, >!, >&!`)

```
fmt
fmt < personal_letter
fmt > new_file
fmt < personal_letter > new_file
fmt >> personal letter
fmt < personal_letter >& new_file
fmt >! new_file
fmt >&! new_file
```

Pipes (6.3)

- Examples:

```
who | wc -l
ls /u/csc209h |& sort -r
```

- For a *pipeline*, the standard output of the first process is connected to the standard input of the second process

Filename Expansion (6.5 p170)

- Examples:

```
ls *.c
rm file[1-6].?
cd ~/bin
ls ~culhane
```

`*`	Matches any string (including null)
`?`	Matches any single character
`[...]`	Matches any one of the enclosed characters
`[.-.]`	Matches any character lexically between the pair
`[!...]`	Matches any character not enclosed

Command Aliases (6.5 p167)

- Examples:

```
alias md mkdir
alias lc ls -F
alias rm rm -i
\rm *.o
unalias rm
alias
alias md
alias cd 'cd \!*; pwd'
```

S-22

Job Control (6.4)

- A *job* is a program whose execution has been initiated by the user
- At any moment, a job can be running or stopped (suspended)
- Foreground job:
  - a program which has control of the terminal
- Background job:
  - runs concurrently with the parent shell and does not take control of the keyboard
- Initiate a background job by appending the "**&**" metacharacter
- Commands: **`jobs`**, **`fg`**, **`bg`**, **`kill`**, **stop**

S-23

Some Examples

`a | b | c`
- connects standard output of one program to standard input of another
- shell runs the entire set of processes in the foreground
- prompt appears after c completes

`a & b & c`
- executes a and b in the background and c in the foreground
- prompt appears after c completes

`a & b & c &`
- executes all three in the background
- prompt appears immediately

`a | b | c &`
- same as first example, except it runs in the background and prompt appears immediately

S-24

The History Mechanism (6.5 p164)

- Example session:

```
alias grep grep -i
grep a209 /etc/passwd >! ~/list
history
cat ~/list
!!
!2
!-4
!c
!c > newlist
grpe a270 /etc/passed | wc -l
^pe^ep
```

S-25

Shell Variables
(setting)

- Examples:

```
set V
set V = abc
set V = (123 def ghi)
set V[2] = xxxx
set
unset V
```

S-26

Shell Variables
(referencing and testing)

- Examples:

```
echo $term
echo ${term}
echo $V[1]
echo $V[2-3]
echo $V[2-]
set W = ${V[3]}

set V = (abc def ghi 123)
set N = $#V
echo $?name
echo ${?V}
```

S-27

Shell Control Variables (6.6)

`filec`	a given with tcsh
`prompt`	my favourite: `set prompt = "%m:%~%#"`
`ignoreeof`	disables *Ctrl-D* logout
`history`	number of previous commands retained
`mail`	how often to check for new mail
`path`	list of directories where *csh* will look for commands (†)
`noclobber`	protects from accidentally overwriting files in redirection
`noglob`	turns off file name expansion

- *Shell variables* should not to be confused with *Environment variables.*

S-28

Variable Expressions

- Examples:

```
set list1 = (abc def)
set list2 = ghi
set m = ($list2 $list1)

@ i = 10     # could be done with "set i = 10"
@ j = $i * 2 + 5
@ i++
```

- comparison operators: `==, !=, <, <=, >, >=, =~, !~`

S-29

File-oriented Expressions

Usage:

```
    -option filename
```

where 1 (true) is returned if selected option is true, and 0 (false) otherwise

`-r filename`	Test if *filename* can be read
`-e filename`	Test if *filename* exists
`-d filename`	Test if *filename* is a directory
`-w filename`	Test if *filename* can be written to
`-x filename`	Test if *filename* can be executed
`-o filename`	Test if you are the owner of *filename*

- See Wang, table 7.2 (page 199) for more

S-30

csh

S-31

csh Script Execution (Ch.7)

- Several ways to execute a script:
  1) `/usr/bin/csh script-file`
  2) `chmod u+x script-file`, then:
     a) make first line a comment, starting with "`#`"
        – (this will make your default shell run the script-file)
     b) make first line "`#!/usr/bin/csh`"
        – (this will ensure *csh* runs the script-file, preferred!)
- Useful for debugging your script files:
  "`#!/usr/bin/csh -x`" or "`#!/usr/bin/csh -v`"
- Another favourite:
  "`#!/usr/bin/csh -f`"

S-32

if Command

- Syntax:

```
    if ( test-expression ) command
```

- Example:

```
    if ( -w $file2 ) mv $file1 $file2
```

- Syntax:

```
    if ( test-expression ) then
       shell commands
    else
       shell commands
    endif
```

S-33

if Command (cont.)

- Syntax:

```
if ( test-expression ) then
   shell commands
else if ( test-expression ) then
   shell commands
else
   shell commands
endif
```

S-34

foreach Command

- Syntax:

```
foreach item ( list-of-items )
   shell commands
end
```

- Example:

```
foreach item ( 'ls *.c' )
   cp $item ~/.backup/$item
end
```

- Special statements:

`break`	causes control to exit the loop
`continue`	causes control to transfer to the test at the top

S-35

while Command

- Syntax:

```
while ( expression )
   shell commands
end
```

- Example:

```
set count = 0
set limit = 7
while ( $count != $limit )
   echo "Hello, ${USER}"
   @ count++
end
```

- **break** and **continue** have same effects as in *foreach*

S-36

switch Command

- Syntax:

```
switch ( test-string )
   case pattern1:
      shell commands
      breaksw
   case pattern2:
      shell commands
      breaksw
   default:
      shell commands
      breaksw
end
```

S-37

goto Command

- Syntax:

```
goto label
...
other shell commands
...

label:
 shell commands
```

S-38

repeat Command

- Syntax:

```
repeat count command
```

- Example:

```
repeat 10 echo "hello"
```

S-39

Standard Variables

`$0`	⇒	calling function name
`$N`	⇒	Nth command line argument value
`$argv[N]`	⇒	same as above
`$*`	⇒	all the command line arguments
`$argv`	⇒	same as above
`$#`	⇒	the number of command line arguments
`$<`	⇒	an input line, read from stdin of the shell
`$$`	⇒	process number (PID) of the current process
`$!`	⇒	process number (PID) of the last background process
`$?`	⇒	exit status of the last task

S-40

Other Shell Commands

```
source file
shift
shift variable
rehash
```

- Other commands … see Wang, Appendix 7

S-41

Example: *ls2*

```
# Usage: ls2
# produces listing that separately lists files and dirs

set dirs = `ls -F | grep '/'`
set files = `ls -F | grep -v '/'`

echo "Directories:"
foreach dir ($dirs)
    echo " " $dir
end

echo "Files:"
foreach file ($files)
   echo " " $file
end
```

S-42

Example: *components (Table 7.3)*

```
#!/usr/bin/csh -f
set test = a/b/c.d
echo "the full string is:" $test
echo "extension (:e) is: " $test:e
echo "head (:h) is: " $test:h
echo "root (:r) is: " $test:r
echo "tail (:t) is: " $test:t

### output:
# the full string is: a/b/c.d
# extension (:e) is:  d
# head (:h) is:  a/b
# root (:r) is:  a/b/c
# tail (:t) is:  c.d
```

S-43

Example: *debug*

```
#!/usr/bin/csh -x
while ( $#argv )
   echo $argv[1]
   shift
end
# while ( 2 )        ⇒ output of "debug a b"
# echo a
# a
# shift
# end
# while ( 1 )
# echo b
# b
# shift
# end
# while ( 0 )
```

S-44

Example: *newcopy*

```
#!/usr/bin/csh -f
### An old exam question:
# Write a csh script "newcopy <dir>" that copies files
# from the directory <dir> to the current directory.
# Only the two most recent files having the name progN.c
# are to be copied, however, where N can be any of 1, 2,
# 3, or 4. The script can be written in 3 to 5 lines:

set currdir = $cwd
cd $argv[1]
set list = (`ls -t -1 prog[1-4].c | head -2 |
             awk '{print $8}'`)
foreach file ($list)
   cp $file $currdir/.
end
```

S-45