Ch. 6: C Shell

- C shell supports all the core features discussed in Ch. 3
  plus the following:

  * several ways to set and access variables
  * built-in programming language
    (conditional branching, looping etc.)
  * command customization using aliases
  * access to previous commands using history mechanism
  * advanced job control
  * several new built-in and enhancements to existing commands

Startup

- The C shell is located in /bin/csh usually; it is a C program!
- When started as a login shell, the startup sequence is as follows:
  * Execute commands in $HOME/.cshrc, if it exists
  * Then, execute commands in global login initialization file /etc/login, if it exists
  * Then, execute commands in $HOME/.login

A sample .cshrc file

- The .cshrc file usually contains commands that set aliases or anything else that applies only to the current shell. (rc stands for run commands)

```
alias h history
alias ll ls -l
alias ls ls -F
alias rm rm -i
```

- The .login file usually contains commands to set the terminal, set environment variables, etc.

```
echo -n "Enter your terminal type (default is vt100): "
set termtype = $<
set term = vt100
if ("$ermtype" != "") set term = "$termtype"
unset termtype
set path=(. /bin /usr/bin /usr/local/bin )
stty erase "^?" kill "^U" intr "^C" eof "^D" crt crterase
set cdpath = (~)
set history = 40
set notify
set prompt = "! % "
set savhist = 32
```

Variables

```
- set name = value
  # if name is not specified, null string is assigned
  # if name = value is omitted, all variables are listed
  # value could be a list within parentheses
- Some examples:

% set flag
% set color = red
% set name = "Graham Glass"

Two new access methods:
(1) ${name}   ; use this if name immediately followed by string
(2) ${?name}  ; replaced by 1 if name is set and 0 otherwise

% set verb = sing
% echo I like ${verb}ing
  I like singing
```

The following is a script illustrating the (2) access method

```
set flag = abc
if (${?flag}) then
echo flag is set
endif
```

List Variables

```
% set colors = ( red green blue )
% echo $colors[1]
  red
% echo $colors[2-3]
  green blue
% echo $colors[*]
  red green blue
% echo $#colors
  3
```

Building lists:

```
% set colors[4] = yellow   # Not ok since [4] does not exists
% set colors = ( $colors pink )
% echo $colors
  red green blue pink
% set colors[4] = yellow   # ok since [4] exists
% set colors = $colors black  # forgot ()!
% echo $colors                # sets only first value
  red
% set s1 = (a b c)
% set s2 = (b c f)
% set s3 = ($s1 $s2)
% echo $s3
  a b c b c f
```

Some Predefined local variables

`$<`	The next line of standard input
`$argv`	A list containing all of the positional parameters; `$argv[1] = $1` etc.
`$cwd`	Current working directory
`$home`	Shell's home directory
`$ignoreeof`	Prevents shell from terminating when CTRL-D is pressed
`$noclobber`	Prevents existing files from being overwritten by > and nonexisting files from being appended using >>
`$noglob`	Prevents wildcard expansion
`$path`	Used to locate executables (list of directories extracted from PATH environment variable)
`$prompt`	The shell prompt
`$savehist`	Number of commands to save in the history file
`$shell`	Full path name of the login shell
`$status`	exit code of the last command

```
#!/bin/csh
echo -n "please enter your name: "
set name = $<
echo ho $name, your current directory is $cwd
```

- Environment Variables

```
setenv name word   # no list values allowed
% setenv TERM vt100
```

- predefined environment variable $LOGNAME (shell owner's user id)

String, Arithmetic and File-oriented Expressions

```
- String expressions:
     s1 == s2  true if exactly equal
     s1 != s2  true if unequal
     s1 =~ s2  Like == except rhs may contain wildcards
     s1 !~ s2  Like != except rhs may contain wildcards
  Note: if either side is a list, the first item is used
        for comparison
```

```
#!/bin/csh
echo -n "do you like the C shell? "
set reply = $<
if ($reply == "yes") then
echo you entered yes
else if ($reply =~ y*) then
echo I assume you mean yes
else
echo not yes
endif
```

- Arithmetic expressions (P 189 Table lists all operators)
  similar to C language
  &, &&, |, ||, <, >, <<, >> operations must be surrounded
  by () so that they are not misinterpreted as shell operators

```
ex. if (($a > $b) || ($a <= $c)) then
```

Do not use the set command to assign an expression to a variable
Instead use the @ command as follows: (@ by itself lists all shell variables)

```
% @a = 2 * 2
% echo $a
  4
% @a += 1
% echo $a
  5
```

- File-oriented Expressions:

```
-option fileName
```

Option	Meaning
r	Shell has read permission for fileName
w	Shell has write permission for fileName
x	Shell has execute permission for fileName
e	fileName exists
o	fileName is owned by the same user as that of the shell process
z	fileName exists and is 0 bytes in size
f	fileName is a regular file (not dir, special)
d	fileName is a directory

The following is a script called 63.csh

```
#!/bin/csh
echo -n "Enter name of file you wish to erase: "
set fileName = $<
if (! (-w $fileName) ) then
echo you do not have permission to erase the file
else
rm $fileName
endif
```

```
% 63.csh
  Enter name of file you wish to erase: /
  you do not have permission to erase the file
```

Filename completion

```
% set filec

- after setting filec, you do not have to type the entire file names
  instead you may type esc key after initial few letters
  if a unique match exists, the file name is automatically completed
  otherwise type * to view all matching files

- In tcsh the shell displays all matching files when esc is pressed.
```

Aliases

```
- alias word string
- unalias pattern

Some useful aliases:

alias ls ls -F
alias rm rm -i
alias h history
alias c clear
alias ls-l ls -l
alias ll ls -l
alias dir ls
```

History Mechanism

- The C shell keeps a record of the commands entered from the keyboard; These can be recalled, edited, and executed at a later stage. The meta character ! gives access to the history of commands.

- Numbered commands:

```
% set prompt = '\! % '  # set prompt to include command/event number

% set history = 40      # remember last 40 commands

% set savehist = 32     # save last 32 commands between sessions

% alias h history       # h is an alias for history which lists
                          the history
```

Command Reexecution:

```
!!             replaced with text of last command
!num           replaced with text of command numbered num
!prefix        replaced with the most recent command which
               started with prefix
!?substring?   replaced with text of the most recent command
               containing substring
```

Accessing pieces of a previous command: (These modifiers can be placed immediately after event specifier)

```
:0          first token
:number     (number+1)st token
:start-end  (start+1) through (end+1) tokens
:^          first token (colon optional)
:$          last token (colon optional)
:*          second through last token (colon optional)
```

```
48 % echo I like horseback riding
49 % !!:0 !!:1 !!:2 !!:4
echo I like riding
I like riding
50 % echo !48:1-$
echo I like horseback riding
I like horseback riding
```

Accessing portions of filenames

If the token extracted is a file name, the various parts of the filename can be extracted as follows:
let filename be /usr/include/stdio.h

```
:h  head       /usr/include
:r  root       /usr/include/stdio
:e  extension  h
:t  tail       stdio.h
```

```
53 % ls /usr/include/stdio.h
/usr/include/stdio.h
54 % echo !53:1:h
/usr/include
```

History substitution: !event:s/sss/ttt/

Control Structures

- If a control structure is entered on the keyboard on several lines, the C shell prompts with a ? for each subsequent line.

- foreach...end

```
foreach name (wordList)
command-list  # break/continue can be used as commands
end
```

example:

```
foreach color (red blue green)
  echo $color
end
```

- goto name

example:

```
echo gotta jump
goto endOfScript
echo I will never echo this
endOfScript:
echo the end
```

if..then..else..endif

```
- if (expr) command

  % if (5 > 3) echo five is greater than 3

- if (expr1) then
    commands1
  else if (expr2) then
    commands2
  else
    commands3
  endif

#!/bin/csh
echo -n "enter a number: "
set number = $<
if ($number < 0) then
  echo negative
else if ($number == 0) then
  echo zero
else
  echo positive
endif
```

- onintr label

```
#!/bin/csh
onintr controlC
while (1)
  echo infinite loop
  sleep 2
end
controlC:
  echo control C detected
```

```
- repeat expr command

% repeat 2 echo Hi there

- switch..case..endsw

  switch (expr)
    case pattern1:
       commands1
       breaksw
    case pattern2:
    case pattern3:
       commands2
       breaksw
    default:
       defaultCommands
  endsw
```

```
#!/bin/csh
echo menu test program
set stop = 0
while ($stop == 0)
cat << ENDOFMENU
  1   : print the date
  2,3 : print the current working directory
  4   : exit
ENDOFMENU
echo
echo -n 'your choice: '
set reply = $<
switch ($reply)
  case "1":
    date
    breaksw
  case "2":
  case "3":
    pwd
    breaksw
  case "4":
    set stop = 1
    breaksw
  default:
    echo illegal choice
    breaksw
endsw
end
```

```
- while (expr)
    commandList
  end

#!/bin/csh
set x = 1                     # set outer loop value
while ($x <= $1)              # outer loop
 set y = 1                    # set inner loop value
 while ($y <= $1)             # inner loop
   @ v = $x * $y              # calculate entry
   echo -n $v "       "       # display entry
   @ y ++                     # update inner loop counter
 end
 echo ""                      # newline
 @ x ++                       # update outer loop counter
end

% multi.csh 4
1        2        3        4
2        4        6        8
3        6        9        12
4        8        12        16
```

Sample Project: junk.csh

```
#! /bin/csh
# junk script
# author: Graham Glass
# 9/25/91
#
# Initialize variables
#
set fileList = () # a list of all specified files.
set listFlag = 0 # set to 1 if -l option is specified.
set purgeFlag = 0 # 1 if -p option is specified.
set fileFlag = 0 # 1 if at least one file is specified.
set junk = ~/.junk # the junk directory.
```

```
#
# Parse command line
#
foreach arg ($*)
 switch ($arg)
   case "-p":
     set purgeFlag = 1
     breaksw
    case "-l":
     set listFlag = 1
     breaksw
    case -*:
     echo $arg is an illegal option
     goto error
     breaksw
    default:
     set fileFlag = 1
     set fileList = ($fileList $arg) # append to list
     breaksw
 endsw
end
```

```
#
# Check for too many options
#
@ total = $listFlag + $purgeFlag + $fileFlag
if ($total != 1) goto error
#
# If junk directory doesn't exist, create it
#
if (!(-e $junk)) then
 'mkdir' $junk
endif
#
# Process options
#
if ($listFlag) then
 'ls' -lgF $junk             # list junk directory.
 exit 0
endif
#
if ($purgeFlag) then
 'rm' $junk/*       # remove contents of junk directory.
 exit 0
endif
#
```

```
if ($fileFlag) then
 'mv' $fileList $junk        # move files to junk directory.
 exit 0
endif
#
exit 0
#
# Display error message and quit
#
error:
cat << ENDOFTEXT
Dear $USER, the usage of junk is as follows:
 junk -p means "purge all files"
 junk -l means "list junked files"
 junk <list of files> to junk them
ENDOFTEXT
exit 1
```

Enhancements

- Command reexecution: A shortcut

  ^sss^ttt

  would replace sss in the previous command with ttt

- Metacharacters {}

  % cp /usr/include/{stdio,signal}.h
  copies two files in one command

- Filename Substitution

  * Disabling filename substitution

```
% set noglob
% echo a*
a*
```

  * No-match situations

```
% echo a* p*
p1.c p2.c
% echo a* b*
echo: no match
% set nonomatch   # causes wildcard to be disabled if no match
% echo a* b*
a* b*
```

- Protecting files from accidental overwrites

```
% set noclobber
% cat a.c > b.c
errors: file exists
```

- Redirecting the standard error channel (use >& and >>&)

```
% cc a.c >& errors   # error messages sent to file;
%
```

- To send std. output and error along pipeline use |&

Job Control

```
% jobs -l
lists all jobs (background processes) currently active

specifying a job:
  %integer     using job number (PID)
  %prefix      jobs beginning with prefix
  %+           last referenced job
  %%           same as $+
  %-           second to last referenced job

% bg %job
places job in the background (used with suspended jobs)
if no job specified, last referenced job is used

% fg %job
brings job to foreground
if no job specified, last referenced job is used
```

```
% stop %job
suspends job
if no job specified, last referenced job is used

% suspend
suspends the shell that invokes the command; useful only
when shell was invoked as a child process

% nice integer command
sets the run level priority for the command (larger the number
lower the priority; default = 4; negative numbers can be
used by super user only)
```

- Terminating the login shell

```
% set ignoreeof    # prevents ^D logout
% exit
```

.logout file is executed by login shell when it terminates

Some built-in commands:

```
% chdir    # works same as cd

% glob     # works same as echo except it NULL terminates its output
             useful in C programs.

% source file
executes commands stored in file

Directory Stack:

% pushd  dir   # same as cd except current dir is pushed on stack

% popd

read details
```

- Hash table of executables in $PATH directories (to speed up)
- Whenever a new executable is added to a directory in $PATH, use

```
% rehash
```

to reconstruct hash table.

```
% unhash
```

disables hash table; slows search process