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

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

Ch. 6: C Shell
Then, execute commands in $HOME/.login
/etc/.login, if it exists

* Then, execute commands in $HOME/.login initialization file
  and commands in $HOME/.cshrc, if it exists

- When started as a login shell, the startup sequence is as follows:
  - The C shell is located in /bin/csh usually; it is a C program

Startup
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
set sahlist = 32
set prompt = " %i"
set notity
set history = 40
set $catid = ("
set history erase "#?" $catid" inter "-C" get "-D" crt create
(set path=.

unset termtype

if ("$termtype" ieq "" set term = "$termtype"
set term = vt100
set term.major = vt100
set term.major = vt100

echo -n "Enter your terminal type (default is vt100):"

set environment variables, etc.

-the login file usually contains commands to set the terminal,
I like string
\{verb\}
\echo % echo I like \{verb\}
\set \verb = string %

\{verb\} is replaced by I if name is set and \0 otherwise
\{name\} is used this if name immediately followed by string

Two new access methods:

\set name = "Graham Glass"
\set color \= \red
\set flag \= \false

Some examples:

- Value could be a list within parentheses
  \# if name = value \& value is omitted, \all variables are listed
  \# if name is not specified, null string is assigned
  \# if name = value

- Variables
The following is a script illustrating the (2) access method:
x > w - x - y

T

G%#

T

8

cEd

T

8

cUd

T

8

cD#

T

8
Z

c

( % set colors = ( red green blue )

% set variables

3

echo $colors
red green blue
[*] colors
red green blue
[2-3] colors
red
[1] colors

List Variables
a b c d f

% echo $3
(% set s3 = $3
(% set s2 = (b c f)
(% set s1 = (a b c)

red
echo $colors

set colors = $colors block
forget ()

set colors = $[yellow [4 exists

red green blue pink

echo $colors

( pink $colors ) = $colors pink

set colors = $[yellow [4 does not exists

Not ok since [4 does not exists

Building lists:
exit code of the last command

full path name of the login shell

number of commands to save in the history file

The shell prompt

(extracted from PATH environment variable)

used to locate executable files (list of directories

prevents with expansion

and nonexistent files from being appended using ""

<

prevents existing files from being overwritten by

prevents shell from terminating when CTRL-D is pressed

shell's home directory

current working directory

eware[1] = $1 etc.

eware

Some predefined local variables

> $
(preferred environment variable $LOGNAME (shell owner’s user id))

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

- Environment Variables

  echo no $name, your current directory is $cwd
  set name = $>
  echo in "please enter your name:"
  i/bin/csh
#
for comparison

Note: if either side is a list, the first item is used

sl | s2
---|---
\text{like} \neq \text{except this may contain wildcards}
\text{sl} \neq \text{s2} \text{ except this may contain wildcards}
\text{sl} = \text{s2} \text{ if unequal}
\text{sl} \neq \text{s2} \text{ if exactly equal}

- String expressions:

- String Arithmetic and File-oriented Expressions
endIf

echo not yes
else

echo I assume you mean yes
else if ($reply =~ ^\* ) then

echo you entered yes
else if ($reply == "yes") then

set reply = $>

echo -n "do you like the c shell?"

#
5
\texttt{echo a}$
\%
\texttt{a ++ = 1}
\%
4
\texttt{echo a}
\%
\texttt{a = 2 * 2}
\%

\texttt{variables}

Instead of using the $ \texttt{set} $ command to assign an expression to a variable,

\texttt{ex. if ((a > b) || (a >= c)) then}

\texttt{by ( ) so that they are not mistinterpreted as shell operators}.

\texttt{Similar to C language.}

\texttt{- Arithmetic expressions (p. 189 Table lists all operators).}
Tilename is a directory
Tilename is a regular tilename (not dir, not special)
Tilename exists and is 0 bytes in size
of the shell process
Tilename is owned by the same user as that
Tilename exists
Shell has execute permission for tilename
Shell has write permission for tilename
Shell has read permission for tilename

Option

Option tilename

- File-oriented expressions:
The following is a script called 63.cash

if [ -w $TITLENAME ]
  then
    set TITLENAME = "
  echo -n "Enter name of title you wish to erase:"

  echo do not have permission to erase the title
  /% 63.cash

  endit
  rm $TITLENAME
  else
    echo you do not have permission to erase the title

  fi

# /bin/cash
% set TITLE

...
aliases dir is
aliases ll is -l
aliases ls -l is -l
aliases clear
aliases h history
aliases rm -i
aliases ls -p

Some useful aliases:

- unalias pattern
- aliases word string
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:

- History Mechanism:

% set history = 40
% set savehist = 32
% set prompt = '\!'
% set prompt to include command/event number

# h is an alias for history which lists
default history

# save last 32 commands between sessions

% alias h history

% alias h history
44

\textbf{runtime substructure}\n
\textbf{substructure} replaced with text of the most recent command

\textbf{prefix} replaced with the most recent command which

\textbf{num} replaced with text of command numbered num

\textbf{ii} replaced with text of last command

\textbf{Command Reexecution:}
I like horseback riding.

```
$0% echo %echo $1-$
I like riding
```

```
49 % echo %echo $1:2:1:0:1::0:
48 % echo %echo $1:2:1:0:1::0:
```

```
second through last token (colon optional)
*:
```

```
last token (colon optional)
$:
```

```
first token (colon optional)
_:
```

```
(start+1) through (end+1) tokens
```

```
number (number+1)st token
```

```
first token
```

```
placed immediately after event specifier
```

(These modifiers can be accessing pieces of a previous command.)
History substitution: \texttt{event:s/ss/tt}/

```
usr/include
64 \% echo \%t
usr/include/studio.h
53 \% ls /usr/include/studio.h

studio

```

Let \texttt{dirname} be /usr/include/studio.h.
The \texttt{dirname} can be extracted as follows:
If the token extracted is a file name, the various parts of
accessing portions of file names.
end

foreach color (red blue green)

Example:

end

command-list # break/continue can be used as commands

d - foreach.

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

end of script:

echo I will never echo this
go to end of script
go to gotta jump

example:

go to name
endit
  echo positive
  else
    echo zero
  else if (number == 0) then
    echo negative
  else if (number > 0) then
    set number = "enter a number:"
  "\n
endit

end commands

else
  commands2
  else if (expr2) then
    commands2
  else if (expr1) then
    -
  % if (5 < expr) echo five is greater than 3
  "
  - if (expr) command
  -
echo control c detected
  : control:
end

sleep 2

echo infinite loop

while (1)
  counter control

  #!/bin/csh

  - counter tablet
endsw

defaultcommands

default:
default:
break
break
command2
command2
break
break
command1
command1
break
break
switch (expr)

- switch. case. endsw

- switch expr command

% repeat 2 echo Hi there

% repeat expr command
end
end
break
echo Illegal choice
default:
break
set stop = 1
set reply = """"case "$stop"
    when 1
        case "$reply"
            when "1"
                echo menu test program
            else
                echo "invalid choice"
            fi
    when 2,3
        printf the current working directory
        printf the date
        exit
    when 4
        str """"grep"""" file
    when 5
        exit
    fi
echo "invalid choice"
fi
switch ($reply)
    set reply = "$reply"
end
#!/bin/csh

# Initialize variables
set fileList = () # a list of all specified files.
set purgeFlag = 0 # set to 1 if -P option is specified.
set junkFlag = 0 # set to 1 if at least one file is specified.
set junk = "/" # junk directory.

Sample Project: junk.csh

# Author: Graham Glass
# 9/25/91

# junk script
end

end

debug
break

set TITLELIST ( $TITLELIST $ARE ) # append to list

set TITLELIST = i

default:

break

Goto error

foreach $ARE in $TITLELIST option

case $- *

break

set TITLELIST = i

case $ - " "

break

set TITLELIST = i

case $ - ":

switch ($ARE )

foreach are $ ( )

# Parse command line

#
```bash
# Remove contents of junk directory.
if [ ! -e $junk ]
then
  if [ ! -d $junk ]
  then
    # List junk directory.
    exit 0
  fi
fi

# Process options

# If junk directory doesn't exist, create it
if [ ! -d $junk ]
then
  (total = 1) &
fatal $junk + $junk + $junk
fi

# Check for too many options
```
exit 1

ENDOFFILE

junk <input file> to junk them
junk -1 means "just junked files"
     junk -p means "purge all files"

Dear USER, the usage of junk is as follows:
can >> ENDOFFILE

error:

# Display error message and quit

# exit 0
#
endit
exit 0

# move files to junk directory.

if ($1 == junk)
    then
        mv "$2" $junk
        exit 0

endif
copies two files in one command
% cp /usr/include/stdio.h

{} Metacharacters

would replace sss in the previous command with ttt

Command re-execution: A shortcut
* * *
* % echo a * b *
  % set nonmatch # causes wildcard to be disabled if no match
  echo: no match
  % echo a * b *
  pt c p2 c
  % echo a * p *
  No-match situations
  * * *
  % echo a *
  % set noglob
  %
  Disabling filename substitution

- Filename Substitution
To send std. output and error alone pipe the use logfile

% cc a.c >% errors # error messages sent to title.

% cc a.c >% errors # error messages sent to title.

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

errors: title exists

% cat a.c > b.c
% set october

Protecting titles from accidental overwrites
Job Control

If no job specified, last referenced job is used

Prints job to foreground

If %job

If no job specified, last referenced job is used

Places job in the background (used with suspended jobs)

If %job

Second to last referenced job

Same as %

+$$

Last referenced job

+%

Jobs beginning with "prefix"

%integer

Integer number (PID)

Specifying a job:

Lists all jobs (background processes) currently active

% jobs -1
t

used by super user only

lower the priority; default = 4; negative numbers can be
sets the run level priority for the command (larger the number
% nice integer command

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

if no job specified, last referenced job is used
suspends job
% stop %job
logout file is executed by login shell when it terminates

```bash
% exit
% set -e
# prevents _D logout
- terminating the login shell
```
read details

pop %

push dir % same as cd except current dir is pushed on stack

directory stack:
exectues commands stored in title %
source title %

useful in C programs.

% glob # works same as echo except it NULL terminates its output

cd # works same as cd

some built-in commands:
disables hash table; slows search process
% unhash
to reconstruct hash table.
% rehash

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