Lecture #8 – Interactive C and TC Shells (Chapter 9)

- Introduction

  Both C and TC shells provide interactive features not available in Bourne shell
  Programming interfaces of C and TC shells are nearly identical
  TC Shell provides a number of interactive improvements over C shell

- Initialization files

  The global files /etc/csh.cshrc and /etc/csh.login are executed first
  The $HOME/.login file is executed once on login

  The user specific files $HOME/.cshrc or $HOME/.tcshrc are execute on login, and every time a new subshell is started.

- Search path

```
%  set path = (/usr/bin /usr/ucb /bin /usr .)
%  echo $path
/usr/bin /usr/ucb /bin /usr .
%  echo $PATH
/usr/bin:/usr/ucb:/bin:/usr:.
```

  Note: You will probably want to set the path in the .login file since .cshrc can be re-executed when a subshell is started.

- The prompts

  The C-shell has two prompts: the primary prompt (%) and the secondary prompt (?).
  The TC-shell uses a ‘>’ as its primary default prompt

```
%  set prompt = “$LOGNAME > “
richj >
```

- Exit status

```
%  grep “nicky” /etc/passwd
%  echo $status
1
```

- Command line history

set history = 100	(keep X commands for this login session)
set savehist = 20	(keep X commands across logins)
"history" displays saved commands	(may need to pipe through more)

Re-executing commands:

```
% date
Mon Apr 26 12:27:35 PST 2004

#  Re-execute the previous command
%  !!
date
Mon Apr 26 12:28:25 PST 2004

#  Run command number 3 from history
%  !3
date
Mon Apr 26 12:28:25 PST 2004

#  Rerun the latest command starting with “d”
%  !d
date
Mon Apr 26 12:28:25 PST 2004

#  Rerun last command with some substitutions
%  dare
dare:  Command not found
%  ^r^t
date
Mon Apr 26 12:28:25 PST 2004
```

- Command line editing (TC shell only)

  The command line can be edited by using the same key sequences that you use in either the emacs or vi editors.

  You can use editor commands to scroll up and down the history list.

  The “bindkey” built-in command is used to select either vi or emacs:

```
%  bindkey –v        #  select vi
%  bindkey –e        #  select emacs
```

- Aliases

  An alias is a user-defined abbreviation for a command.

  Example:

```
% alias ll ls -lg
% ll
```

Its also legal to alias existing commands (but not recommended)
(% alias ls ls -lg, commonly done with rm -i)

alias <cmd> gives the current contents of the alias for that cmd
unalias <cmd> removes the alias for that command
alias without any args displays all current aliases

Argument substitution (% alias last echo \!:$ would give you last argument)
(i.e. % last this is a string, would yield "string")

- Job Control

Jobs	Lists all the jobs running
^Z (control-Z)	Stops (suspends) the job; the prompt appears on the screen
bg	Starts running the stopped job in the background
fg	Brings a background job into the foreground
kill	Sends a kill signal to a specified job

```
% find /usr -name ace -print > findout &
[1] 26041

% jobs
[1] + Stopped vi filex
[2] + Running find ...

%  fg %1
vi filex
```

- Tilde expansion

The tilde (~) character by itself exapdns to the full name of the user’s home directory.
Tilde followed by a username expands to the full name of that user’s home directory.

```
%  echo ~
/home/richj
%  echo ~ann
/home/ann
```

- Filename completion

C shell provides a shortcut metghod for typing filenames called file completion.

```
%  set filec
%   ls
```

```
rum   rumple   rumplestilsken  run2

%  ls ru<ESC>                    #  terminal beeps since there are multiple matches
%  ls rum^D                      #  show all possible matches
rum   rumple   rumplestilsken

%  ls rump<ESC>
rumple
```

- I/O redirection

>& redirect stdout and stderr to a file

```
% cat x
cat: x: No such file or directory
% cat y
This is y.
% cat x y >& hold
% cat hold
cat: x: No such file or directory
This is y.
```

>>& appends stdout and stderr to a file

```
%  cat x y >>& hold
```

|& pipes stdout and stderr of left hand side to stdin of right hand side

- Variables

All variables are strings like Bash
Can treat strings that contain numbers as numeric (expr)

set, @, setenv to manipulate variables
set assumes a non-numeric string
@ works only with numbers
setenv is similar to export

```
% set name = fred
% echo $name
fred
% set
argv   ()
home /home/jenny
name fred
shell /bin/csh
```

status 0

set variable (without string makes it the null string)
unset variable (actually deletes variable)

- Environment variables

  setenv is used in C-shell much the same way as export in Bourne shell.

```
%  setenv TERM vt100
%  setenv PERSON “Nelly Nerd”
```

- Arrays of variables

  must be declared before use

```
% set colors = (red green blue orange yellow)
% echo $colors
red green blue orange yellow
% echo $colors[3]
blue
% echo $colors[2-4]
green blue orange

% set shapes = (" " " " ")
% set shapes[4] = square

%  set days = ( Monday  Tuesday)
%  shift days
%  echo $days
Tuesday
```

- Numeric variables

  @ variable operator expression
  See page 358 for list of expressions

```
% @ count = 0
% echo $count
0
% @ count = (5 + 2)
% echo $count
7
```

- Miscellaneous

  Braces {} can be used to separate variable from text without space
  Example: % set prefix = Rich
                 % echo ${prefix}ard

  $#variable is number of elements in array
  $?variable is true (1) if defined and false(0) otherwise

- Shell variables

  $argv for command line args
  argv[0] is name of calling program
  argv[1] is first arg, etc
  all args ($argv[*] or $*)
  specific arg ($argv[n] or $n)

  $#argv is number of command line args

  $cwd is current working directory
  $HOME, $PATH

  % setenv PATH (/usr/bin /usr/ucb .)

  $prompt similiar to PS1

  $status is status of last command

  $$ is current PID