Basic C Shell

helpdesk@stat.rice.edu

11th August 2003

This is a very brief guide to how to use cshell to speed up your use of Unix commands. Googling "C Shell Tutorial" can lead you to more detailed information.

Change Your Shell To `tcsh`

Open a terminal, if you can not use left/right arrow to edit your commands, you are NOT using tcsh. To formally check it, type command

```
echo $shell
```

If you see

```
/usr/local/bin/tcsh
```

you are using `tcsh`. Otherwise, enter command:

```
passwd -r nis -e
```

Enter

```
/usr/local/bin/tcsh
```

when prompted. The new setting will be effective after around 10 minutes. For the purpose of this tutorial, use command

```
tcsh
```

to start an instance of `tcsh`.

Exercise:

1. Check you shell and set it to tcsh if necessary.
2. If necessary, use `tcsh` command to enter tcsh. Note that you can use `exit` command to exit this shell.

A Tour of C Shell Features

.cshrc

All the features of cshell can be set in `.cshrc`. This is a plain text file located under you home directory. It will be loaded and executed whenever you open a shell window. After you edit this file (with any text editor of your choice), use `source ~/.cshrc` command to activate the change.

Exercise:

1. Open `.cshrc` with emacs. (Or vi if you prefer). Try to understand each line of it.

Change `$path`

`$path` contains a list of directories to be searched when running programs or shell scripts. For example, the command `Splus` is in `/usr/local/bin`. If `/usr/local/bin` is not in your `$path`, you will have to enter the full path name `/usr/local/bin/Splus` to start Splus, rather than simply `Splus`. Usually, we set `$path` in `~/.cshrc`, which will be executed whenever a new shell is started. The syntax of setting `$path` is:

```
set path = ( $path /home/helpdesk/bin ~/bin .)
```

To check the content of your current $path, use command

```
echo $path
```

Note that '~' means your home directory, '.' means current directory. The above `set path` command add `/home/helpdesk/bin ~/bin` and `.` to your `$path`.

Exercise:

1. check you `$path`.
2. Add `/home/helpdesk/bin` to your `$path` by editing .cshrc file.
3. `source .cshrc` to activate the change.
4. check your `$path` again.

Command Line Editing

Tcsh records up to `$history` your previous commands. You can set `$history` in your .cshrc file as follows

```
set history=50
```

Then, `history` command will list all your previous commands and

```
!27
```

will execute command 27. `!!` is used to repeat your alst command.

You can also use the arrow keys to navigate the command history and edit the command line. It is also very important to know how to use the middle button of your mouse. Try to select your previous command with the left button and paste it with your middle button.

Command Line Completion

The `<TAB>` key is very useful in `tcsh`. Whenever you press `<tab>`, tcsh will try to complete your command, file or directory name. That is to say, you can enter

```
xt<tab>
```

instead of `xterm` if `xterm` is the only command in your searching path that begin with `xt`. This is especially useful if you have very long file/directory names.

Exercise:

1. enter the following commands, try to use your mouse, arrrow and tab keys to simplify the task.
   (a) `cat .cshrc` have a look at .cshrc file. Try to use tab key.
   (b) `more .cshrc` try to use mousr copy and paste, (You might not be able to use this function if you are using windows++xwin32+ssh.) or up/down arrow keys.

Standard Input/Output, Pipe

You need to understand some basics about standard input/output of Unix commands. Briefly, Unix programs usually read from their standard input, like the keyboard, output to their standard output, like the terminal. We can redirect their standard input/output in the following ways:

1. Redirect standard output to a file:
   `cat a.c b.c > sum.c`
2. Add the standard output to another file
   `cat c.c >> sum.c`
3. Redirect standard input from a file:
   `cat < a.c`
4. Use the output of the first program as the input of another program: (pipe)
   `cat a.c | more`

Alias

Alias, as its name implies, maps a (shorter) name to another (long) name. It can be used to give shortcuts to existing commands, create new commands or even 'overwrite' existing commands. For example, command

```
alias ll 'ls -l'
```

maps `ll` to `ls -l` so that `ls -l` will be called whenever `ll` is entered as a command. It is perfectly all right to use command

```
alias ls 'ls -l'
```

to alias `ls` to `ls -l`. The result of this alias is to add a default option `-l` to the `ls` command. This trick is most frequently used on the dangerous `rm` command,

```
alias rm 'rm -i'
```

since the default rm command removes files without warning.

Alias can also handle parameters. Here I only demonstrate the simplest case where you use \!* as the parameters. For example:

```
alias mcd 'mkdir -p \!*; cd \!*'
```

When you call

```
mcd stat
```

it will be expanded to

```
mkdir stat; cd stat
```

Exercise:

Can you mak the following aliases? You can use `unalias` command to disable an alias.

1. `findjob`: list all running jobs and then find jobs that match a string (use `ps -ef, pipe` and `grep`) .
2. `ssh7`: ssh to stat007.stat.rice.edu.
3. `emacs`: Add an `&` at the end of `emacs` command automatically.
4. `rxvt`: set default foreground, background to `rxvt` command. (Use `-fg green -bg black` options.)

Other Useful Features

cdpath

```
set cdpath = (~ ~/public_html ~/stat )
```

and you can use `cd stat410` to change your current directory to ~/stat/stat410 even when you are not under `~/stat`.

prompt

```
set prompt= %n@%m:%~ %
```

will set your prompt to `username@machine name:  current directory %`.

Control Structure

You do need to understand more about variables, expressions before you can use control structure or write a shell script. Here are two very simple examples about how they can be used:

1. Configure for different platforms:
   ```
   if ( $OS = 'Linux' ) then
   source .rc.Linux
   else
   source .rc.Unix
   endif
   ```
2. Using loops to process multiple files:

   ```
   foreach file in (*)
   move $file $file.bak
   end
   ```

Starting from the above basics, you can learn:

- C Shell Script: automate a sequence of jobs
- Make: Similar to shell script but have multiple objects
- Perl: deal with texts

to simplify every routine of your Unix life.

`helpdesk@stat.rice.edu`
$Date: 2003/07/17 02:49:21 $
$Revision: 1.1 $