

Startup Files

Written by [Dave Pawlowski](#), September 23, 2012

Introduction

Every time a new shell session is started, Linux attempts to read several files that set certain variables, options, and other things that the system needs or makes your life easier. These are referred to as startup files. The first startup files that are read are buried in the `/etc/` directory and set things for all users. After that, Linux reads an individual's startup files, which are located in your home directory. If you do:

```
% ls -a
```

you will may see one or more files that start with `'.'`. In particular, the shell will try to read files called `.cshrc` (if you are using `csh` or `tcsh`), `.login`, `.Xresources`, `.Xauthority` to name a few. In addition, certain programs may have their own "dot" file. For example, when `emacs` starts up, it looks for the file, `.emacs`. These files don't have to exist, but if they do, Linux will read them and execute whatever commands are in them. In this tutorial we will just talk about `.cshrc` and `.emacs` in brief. There are endless possibilities for what you can put in these things. A good place to start is to just google `.cshrc` and copy what someone else uses. The point here is to introduce you to a few common options, and show you how to use these files.

`.cshrc`

`.cshrc` and `.login` basically do the same thing, except that `.cshrc` is read first, and `.login` is only read in the login shell. I actually don't know anyone that uses `.login`, but I'm sure there's a use for it. `.cshrc` is typically used to setup variables that the system may or may not use and things called aliases. Aliases are variables that are basically shortcuts to execute commands. For example, lets create an alias for the command `ls -l` using the command line:

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

After hitting enter, type `ll`. You should get a long list of all the files in your current working directory. Doing this is quite useful, but defining an alias this way means that the alias is only good for your current shell session. If you startup another one, i.e. by typing

```
% xterm
```

and pressing enter, you'll notice `ll` is no longer defined. The way to get around this is by putting the alias in a startup file that is executed each time you start a shell session, such as `.cshrc`.

Here is an example of a basic `.cshrc` file.

```
#!/bin/csh
# Sample .cshrc file
set history=200
setenv EDITOR emacs
set path = (. ~/bin /usr/local/mpi/bin /usr/local/bin
/opt/local/bin /opt/local/sbin ${path})
if ($?prompt) then
    set prompt='$user@${host}:%~>> '
endif
alias emcas emacs
alias emasc emacs
alias emcsa emacs
alias h history
alias j jobs
alias l 'ls'
alias ll 'ls -l'
alias la 'ls -a'

setenv term xterm-color
setenv CLICOLOR 1
setenv LSCOLORS cxfxcxdxbxegedabagacad
```

This `.cshrc` file sets a few variables, a few aliases, and changes the look of the prompt.

The first variable, `history`, is used by the `history` command. This sets the default number of commands for `history` to remember.

The second variable, `EDITOR`, is used by the system whenever Linux wants you to edit text. Linux knows that you prefer to use `emacs`. In this case, `EDITOR` is an environment variable, which means that its scope extends beyond the current shell. Any program that is called by the shell will inherit the value of `EDITOR`.

Next, the `path` is set. `Path` stores information telling Linux where to look for executable files. This way, you don't have to actually be in the directory where the executable is in order for it to work. Linux wouldn't work very well if you had to copy `ls` to every directory that you wanted to list files in.

After that, Linux checks to see if a prompt has been defined. `?prompt` is a variable

that returns 1 (or true) if a prompt can be defined. If so, then the code within the if statement is executed. The first thing done is to actually set the prompt. Mine is sort of complicated, but it basically prints out `username@hostname:'current working directory'>>`. The `'%~'` is interpreted as 'print current working directory'.

The next lines setup a few handy aliases. I misspell `emacs` alto.

Finally, the last lines tell the shell that you want files and directories to be color coded when you list them using `ls`. You can assign any color you want to different types of files. That's what that long line that looks like a bunch of garbage does. Take a look at <http://chuck.emich.edu/dpawlows/computational> to see how to set those.

There are lots of other options that you can put in the startup files, but these will get you started.

Restarting your startup file

When you make changes to `.cshrc`, you don't have to start a new shell session to make the changes take effect (most of the time). Instead, you can use the command

```
% source .cshrc
```

which re-executes the file.

.emacs

When you start an emacs session, emacs will look for the file `.emacs` in the home directory and execute it if it exists. When we first talked about emacs, I mentioned that you can define your own commands by binding them to a key sequence. This is where you would do that, so that those key-bindings automatically exist each time you use emacs. Below is a sample `.emacs` file.

```

;; Set up the keyboard so the delete key on both the regular keyboard
;; and the keypad delete the character under the cursor and to the right
;; under X, instead of the default, backspace behavior.
(global-set-key [delete] 'delete-char)
(global-set-key [kp-delete] 'delete-char)

;Use C-l to go to a specific line number
(global-set-key "\C-l" 'goto-line)

;;use C-t to start spell checking
(global-set-key [(control t)] 'ispell-buffer)

;; Enable wheelmouse support by default
(cond (window-system
      (mwheel-install)
    ))

;; Visual feedback on selections
(setq-default transient-mark-mode t)

;; Always end a file with a newline
(setq require-final-newline t)

;; Turn on font-lock mode(language specific colors and
;; Settings) for Emacs
(cond ((not running-xemacs)
      (global-font-lock-mode t)
    ))

```

```

;; Copy to the pasteboard
(defun pbcopy-region (beg end)
  "Copy the region to Mac OS X pasteboard using shell command 'pbcopy'"
  (interactive "r")
  (shell-command-on-region beg end "pbcopy"))
(global-set-key (kbd "M-c") 'pbcopy-region)

```

There are many, many commands that emacs can handle. If you can think of something that you want to do, chances are that you can do it. The syntax for binding a key combination is a bit tricky. A good place to start if you want to add something is to google it, as it's likely that someone has asked the same question before.

Other Startup Files

As mentioned there are other startup files that can be used. One of the most widely used is `.Xresources`. This file sets preferences for fonts, colors, and geometry of Xwindows,

including your xterminal and your emacs windows. This is only useful on your local machine though, as you don't want your window to change when you login to a remote machine. If you are using a windows machine with putty, you can set these preferences in the putty configuration screens.

When you logout, Linux reads a file called `.logout` if it exists. This is not used extensively, except to print neat "goodbye" messages when people log out.