

Customizing your Unix account



John W. Shipman

2008-01-02 19:15

Abstract

Describes special files used to customize Unix operations.

This publication is available in Web form¹ and also as a PDF document². Please forward any comments to tcc-doc@nmt.edu.

Table of Contents

1. Introduction	1
2. Your <code>.bashrc</code> file: Shell configuration	2
3. Your <code>.bash_login</code> file: Login initialization	3
4. Your <code>.bash_logout</code> file: Logout actions	3

1. Introduction

When you use Unix or Linux systems at the New Mexico Tech Computer Center, you can customize your configuration by editing certain files in your root directory. These files control user-level configuration—that is, the appearance and function of your applications.

These files are called “hidden” or “dot” files—that is, their names start with “.”, so they are not shown by the `ls` command unless you use the `ls -a` option.

If you are unfamiliar with Unix conventions, please see our publication *Summary of common Unix commands*³, which explains basic terms like *shell* and *prompt*.

We'll assume that your shell is *bash*. For a complete description of *bash*, see the online *Bash Reference Manual*⁴.

Here are some of the more important dot files you can use to configure *bash*:

- Your `.bashrc` file: Shell commands in this file are executed whenever a new shell or sub-shell is started. See Section 2, “Your `.bashrc` file: Shell configuration” (p. 2).
- Your `.bash_login` file: Shell commands in this file are executed whenever you log in. See Section 3, “Your `.bash_login` file: Login initialization” (p. 3).
- Your `.bash_logout` file: Shell commands in this file are executed whenever you log out. See Section 4, “Your `.bash_logout` file: Logout actions” (p. 3).

¹ <http://www.nmt.edu/tcc/help/pubs/dotfiles/>

² <http://www.nmt.edu/tcc/help/pubs/dotfiles/dotfiles.pdf>

³ <http://www.nmt.edu/tcc/help/pubs/unixcrib/>

⁴ <http://www.gnu.org/software/bash/manual/bashref.html>

To create or modify your dot files, use any basic text editor. Here are some popular choices:

- *pico* is a very simple text editor. See *Using the pico text editor*⁵.
- *emacs* is a full-featured text editor for serious professional use. See *The emacs text editor*⁶. This editor has its own configuration file, named “.emacs”.
- *vi* is another full-featured editor. See *Quick reference for the vi editor*⁷.

Choosing between *vi* and *emacs* is a matter of taste, and in most cases the author has found that people tend to stick with whichever one they learned first.

2. Your .bashrc file: Shell configuration

In your `.bashrc` file you place any shell commands that you want executed every time you start up a new shell.

Here are some lines you might want to add to your `.bashrc` file, with commentary by K. Scott Rowe.

```
## If we are not a login shell, source /etc/profile anyway
if [ "$0" != "-bash" ] ; then
    . /etc/profile
fi
```

The above lines cause the `/etc/profile` file to be read even if the current shell (`$0`) is not *bash*. That file sets up your default search path—the set of directories where *bash* looks to find executable files when you type a command.

```
## To add a directory to your path, do something like this:
export PATH=${PATH}:${HOME}/bin
```

There's a lot going on in the above line:

- The `export` command sets a variable in this script but also “exports” it so it affects things outside this script.
- The `${PATH}` part is a special function that expands to your current search path.
- The `${HOME}` function expands to your home directory's pathname, so effectively it adds your `~/bin` directory to your search path. The colon (“:”) must be used to separate elements of the search path.

```
## set up a happy editor for programs that want them
export EDITOR='pico'
export VISUAL='pico'
```

This sets up two environmental variables, `EDITOR` and `VISUAL`, that some programs expect to find, so they know which editor you like. Substitute *emacs* or *vi* for *pico* in the above lines if you prefer one of those editors.

```
set history=40
```

Tells *bash* to remember the last 40 commands you have typed. “History substitution” allows you to recall such previously entered commands and execute them again.

⁵ <http://www.nmt.edu/tcc/help/editor/pico.html>

⁶ <http://www.nmt.edu/tcc/help/pubs/emacs/>

⁷ <http://www.nmt.edu/tcc/help/pubs/vi/>

```
## some useful aliases, so new users don't hurt themselves
alias rm='rm -i'
alias cp='cp -i'
alias mv='mv -i'
alias ls='ls -F'
```

The `bash alias` command sets up new commands as shorthand for longer commands. For example, the first command makes `rm` an alias for the `rm` command with the `-i` option so that the shell will ask the user before removing files.

The `ls` command's `-F` option decorates the filenames with special characters to remind what they are: a `/` after directory names, a `*` after executable files, and `@` after soft links.

```
## this is to fool the automounter
cd ${HOME}
```

This command does a `cd` to your home directory so that `pwd` prints out your `/u/username` pathname and not the longer “real” (and confusing) path name that may appear due to our physical disk structures.

3. Your `.bash_login` file: Login initialization

Commands in the `.bash_login` file are executed whenever you log in.

For example, you may want to start up a mail reader automatically each time you log in. If, for example, *pine* is your mail reader, just add this line to your `.bash_login`:

```
pine
```

4. Your `.bash_logout` file: Logout actions

The `.bash_logout` file is read whenever you log out. For example, you might want to start housekeeping tasks, such as deleting temporary files or web browser caches.

