Mac OS X Terminal Basics FAQ v2.01

Neal Parikh / neal@macnn.com

August 14, 2001

1 Introduction

This FAQ is intended to be a quick primer on Mac OS X's BSD Subsystem. The BSD Subsystem is a powerful tool that gives you an immense array of new capabilities and access to a large number of new applications. If you learn to use them wisely, you can do some truly incredible things. You have the power; be careful.

2 Table of Contents

3 Why UNIX?

That is the main question, isn't it? Many people are confused as to why Apple has picked UNIX in the first place. There are several reasons why Apple has picked UNIX to be the core of their new OS (not in order of importance):

1. **The historical reason**. Mac OS X's roots trace back to NeXTSTEP, and that used UNIX.
2. **For developers**. Everyone is unfamiliar with this OS – especially Windows developers who have to work on ports of products to Mac OS X. UNIX provides an environment that they're all comfortable with. They may not know Aqua well, but they know UNIX and thus they know the filesystem and all the rules of play (where not to install, etc).
3. **UNIX is established**. Other than developers, a lot of users know UNIX. UNIX was invented in the 1960's, and it's matured considerably since then. Pretty much everyone who studies CS in school/college must have a working knowledge of UNIX.
4. **UNIX is robust**. There's a reason almost every server in the world runs UNIX. At NetCraft, the longest uptime of any server listed is a UNIX (Solaris) server, with an uptime of 1100 days. Basically, the admin sets up Apache and any other servers they're running, and then they just leave the server running. Very low maintenance.
5. **UNIX is flexible**. Never before have Mac users had the privilege of using an OS where working knowledge of the core OS is widespread, and in fact, the entire core OS is opensourced. For example, there's no way of upgrading the OS 9 version of Personal WebSharing without Apple's permission. They wrote it; you have to wait for an update from them. If you find that Apache or SSH (two of the servers bundled with Mac OS X) are out of date, you can just go to a website, download the source code, and update that part of the OS. If you're a DP G4 user and you have a modem, you've likely heard of Louis Gerbarg's DP/Modem fix for Mac OS X. He's a Darwin developer, and all he did was basically get the source code for that PPP extension and just fix it. Then he compiled it and distributed it to people who needed it, and this solved a lot of problems for many people – this kind of power is impossible without a core OS that people know very well, and is also opensource.
6. **Application availability**. While Microsoft may take their sweet time porting Office 10, there are a lot of apps immediately available for OS X because it's UNIX-based. The GIMP, for example, is a free Photoshop replacement (there are some things it can't do, of course, but it's pretty darn good for a free program) that would never have run on the Mac platform if OS X had not been UNIX-based.
7. **Market base**. Developers and techies love OS X because they don't have to use two machines any more. You can run MS Office while you're compiling a Java app for deployment on BSD systems. The CLI is required for writing simple I/O apps (in OS 9, you have to do all this Mac ToolBox junk to get I/O going in an app – in UNIX, include the standard I/O library, and you're on your way). Mac OS X, and thus the Mac platform on the whole, is now available to a lot more people as it's finally a viable solution. Once more developers start using OS X, that means the end user gets more apps. It's a win-win-win situation: for developers, you, and Apple.

There may be other reasons, of course, but this pretty much covers it. UNIX is the most well-known, robust, experienced, fast, and flexible core OS Apple could have picked for Mac OS X. It's a choice that will work well – it'll take them through the next decade.

4 What's Darwin?

Darwin is an open source operating system based in part of BSD UNIX and is in the same family as FreeBSD, NetBSD and OpenBSD (other Unices). Darwin was meant to be the base of Mac OS X but it is, on its own, a complete operating system, albeit one devoid of benefits like a GUI. (Of course, you can install Xfree86 on top of Darwin, which provides full GUI facilities via the standard X11 system, but Xfree86 is not bundled with Darwin.)

Darwin is the UNIX core of Mac OS X. UNIX is an operating system world-renowned for its stability, flexibility, speed, and power. Mac OS X uses a UNIX core (called Darwin) to improve system stability and responsiveness. Because Aqua is essentially running on top of another OS (Darwin), you can have full access to a wide array of UNIX tools within Mac OS X. Terminal is your window to UNIX.

To see a history of the development of various Unices, go to *http://perso.wanadoo.fr/levenez/unix/*. NeXTSTEP, Darwin, and Mac OS X are at the very end of the 12-page timeline.

5 Darwin Basics

Q: What is a Terminal?
A: A Terminal is simply a text-based program that is used to send commands to the OS and interact with it. In the case of Mac OS X, the Terminal program allows the user to interact with the BSD Subsystem directly.

Q: What can I do with the Terminal?
A: You can do almost anything, because the Terminal is basically a window into another OS. Mac programs usually don't take advantage of the UNIX layer, and UNIX programs rarely take advantage of the Mac layer. In the Terminal, you can run an IRC client like BitchX, browse the web with Links or Lynx, read Usenet newsgroups with the Tin newsreader, play Tetris in emacs, write programs, write documents, manage your filesystem, run maintenance and/or system checks, inspect network traffic, and so on and so forth. You can do almost anything in Terminal that doesn't require a full-blown GUI if you're so inclined. In fact, many applications bundled or written for Mac OS X are actually just Cocoa shells for UNIX commandline applications: BrickHouse, a program that lets you configure the built-in firewall, is a shell for ipfw; Virex 7 is a Cocoa shell for the Virex 7 commandline scanner; TeXShop, the program that this document is written in, provides a GUI for accessing the fully standard teTeX distribution of the TeX typesetting language; ProjectBuilder uses gcc, gdb, and other standard UNIX programming tools and wraps them in a GUI with an editor; Cronnix wraps a GUI around the crontab and makes it easier to edit; and so on and so

forth. In fact, people are often using the Terminal at times when they don't really think they are (albeit indirectly).

Q: I've heard it's easy to wreck my system using the Terminal. Is this true?
A: Due to file permissions, users are essentially in a protected space and don't have access to critical system files unless they login as the superuser, or system administrator account (root). You can, however, delete all your own files, but then again, you could do that anyway. You can't really cause any damage you couldn't have caused with the GUI, although one should note that typing a careless delete command is much easier than accidentally dragging all your files to the Trash, selecting Empty Trash, and clicking the OK button. There are no "Are you sure?" prompts; the shell assumes that you know what you're doing. Being careful should stand you in good stead.

6 What's a shell?

Q: I've heard there's more than one shell (bash/tcsh/zsh/ksh). What are the differences?
A: To the average user, none. Shells are merely programs that take in input from the user, and they handle certain things different (such as shell scripting, aliases, command completion). To read very lengthy essays on the specifics of different shells, you can open Terminal and type at the prompt "man tcsh" or "man zsh" for information on the two shells bundled with OS X. Use the down arrow to scroll, space for page down, and q to stop reading. tcsh is the default OS X shell.

Since this is a beginners' FAQ, we won't be doing anything that's shell-specific, and on the whole, you only need to know about the differences between shells when you begin writing shell scripts, although there are a large number of shell-specific features that don't come up that often in everyday use. Read the man pages for zsh and tcsh (the shells bundled with OS X) if you're interested. bash, the default shell on Linux systems, isn't included in OS X, but you can download it (either as source or as a precompiled binary).

7 Running system commands

To run a system command, just type its name at the prompt. **Warning**: There is no undo in UNIX. For example, if you tell the mv command (essentially rename) to rename a file to another existing file, it will just get rid of the first file. You can overwrite entire directories if you're in as root and use careless syntax. The bottom line is to be very, very careful about syntax (especially when moving, copying, or deleting – mv, cp, and rm respectively) and to (a) check the man page for the command and (b) try the command out on some test files to make sure you don't mess up something important. Wiping a blank folder will be much better than wiping your applications directory, obviously. So experiment, but be careful, and if you're in as root or you use the sudo command to run something as root – **think twice**.

A reader suggested using shell aliases to bypass this problem of having no prompt, which may be a good idea while you're learning - training wheels, if you will. Create a .tcshrc file in your home directory (/Users/you/.tcshrc) and paste the following into it:

```
alias rm 'rm -i'
alias mv 'mv -i'
alias cp 'cp -i'
```

Now quit and relaunch Terminal. Now, when you use mv, cp, or rm (explained below), the shell will give you a prompt making sure you want to do what you say you did when you ask to delete or overwrite files.

- `man` – the most useful of all. Type `man commandname` to get an explanation of how to use that command and what it is. Type `man man` for info on how to use `man`.
- `ls` – list the contents of the current directory. `ls -l` will do a more detailed listing (showing permissions, etc), and `ls -a` will show hidden files. You can combine arguments – i.e. `ls -al` will show hidden files and do a long listing.
- `pwd` – print working directory. Tells you the path to where you are. In Mac OS X, you can just type "." (no quotes – just the period) to print working directory. For a complete list of shortcut commands (called aliases, not to be confused with aliases in Mac OS), type `aliases`. To make your own aliases, edit the `/usr/share/init/tcsh/aliases` file if you want the change to be system-wide (for all users) – otherwise, create a `.tcshrc` file in your home directory and put the aliases there.
- `cd` – change directory. Type `cd /applications` to switch to the applications directory. One built-in alias in Mac OS X is to just type `cd` which will also take you back to your home directory. `..` is a nickname for the parent directory, so to go one level up you can type `cd..`
- `ps` – process listing. The PID is the process ID, and it's what you use to reference that specific process. More on this later.
- `ps aux` – complete process listing (it shows you everything that's running on the system, rather than just the processes that your user has started).
- `top` – dynamically updated process viewer app. Type `q` to stop it.
- `more` – if you pipe a command whose output is more than one screenful into `more`, it prompts you to go onto the next page. If you want to look through a directory listing of 500 files, for example (or even 50), you'll want to `ls | more` or else everything but the last screenful will just whizz by and you won't see it.
- `less` – less is a more user-friendly version of more because it allows you to scroll up through the output as well as down (using arrow keys). less is often better, if not more (horrible pun intended). Usage is exactly the same as more.
- `cat` - concatenate. Basically it displays a file's contents and is convenient if you just quickly want to read through a file. If you want to read through the INSTALL file of a UNIX app, for example, you'd type `cat INSTALL`. (Most install documents are longer than one screenful, so you would likely use `less INSTALL` in an actual situation.)

This is basically what you need to get on your way. There are so many UNIX system commands that more would be unnecessary. If you want to know about other commands (many are located

in `/usr/bin` and `/bin`), use the man pages, ask in MacNN's OS X UNIX forum, or visit Mac OS X Hints.

8 Running Your Own Programs

For budding UNIX developers: if you have a source file called hello.c, you compile it by typing `cc -o programname sourcefile.c`. This makes a file called `programname` in the same directory. To run it, type `./programname`. (Note: you do need the Developer Tools installed to be able to compile anything. I strongly recommend you install the Dev Tools because you can't compile UNIX applications (for installation, since most UNIX applications are distributed as source, not binaries) without them – and if you're a developer, then the reasons are obvious.)

Additionally, you can use ProjectBuilder to write your commandline application if you're not fond of Emacs or some other text-only editor. Open ProjectBuilder (`/Developer/Applications/`), go through the basic configuration screen if you haven't already, then press Command-Shift-N or File - New Project, and then choose Standard Tool. Note that while PB is a fine environment for writing even CLI apps, the Run environment is fairly buggy, and you'd do well to have a separate Terminal window open to run the program.

To run shell scripts (end in .sh) like the LimeWire installer (Mac OS X Gnutella client), you type `sh whatever.sh`.

9 What's NetInfo?

NetInfo is a database for system settings. Some people have said it's a bit like the Windows registry. It stores such things as the users of the system, and basic networking information. Mostly, you shouldn't need to know about it. You can poke around with it in NetInfoManager.app, or using the niutil command (`man niutil`).

NetInfo is generally quite thorny and poorly documented, and of little use to average users or even most developers. More information on NetInfo can be found at

http://www.macaddict.com/osx/xphiles/index.html

For a vast repository of highly technical NetInfo documentation, visit *http://www.darwinfo.org*. Additionally, Apple has recently published a document entitled *Understanding and Using NetInfo*, and although the actual usage of NetInfo is primarily targeted at system administrators running Mac OS X Server 10.x, the document is still useful if you're just trying to understand what kinds of things NetInfo is used for.

10 Basics of Compilation

http://www.macaddict.com/osx/xphiles/11_02_00.html is an excellent primer on getting started porting CLI apps. Even if you know what you're doing, porting apps can be extremely difficult, so I won't say much. Most apps, however, compile just fine – here's a quick primer:

Assuming you've downloaded and decompressed the source code:

1. Change to the directory containing the source code.
2. Read the README and INSTALL files with pico. Type "pico README" or "pico INSTALL" to read the files. Use your arrow keys to navigate. Control-X will quit the program and have you back to the prompt. pico lists some shortcuts for page up, page down, etc at the bottom of the screen. Learn to use pico to some extent; you'll need it at times.
3. Once you're out of pico, you need to copy Apple's custom configure files to the folder, so the installer script will recognize which UNIX you're using (Darwin isn't too popular yet). Type:

   ```
   [prompt] cp /usr/libexec/config.* .
   ```

4. Then run the configure script:

   ```
   [prompt] ./configure
   ```

5. Assuming there were no errors, compile the program:

   ```
   [prompt] make
   ```

6. Finally, install the program. This always requires root access.

   ```
   [prompt] sudo make install
   ```

   If this is the first time you've used sudo, it'll give you a little lecture and then prompt you for your password. Enter your user's password. It should start installing. Finally:

   ```
   [prompt] rehash
   ```

7. At this point, I usually update the locate database in case I need to search for any of these files:

   ```
   [prompt] sudo /usr/libexec/locate.updatedb
   ```

   Then you can just type "locate whatever" to find something with "whatever" in the path.

Often, you can get strange and very annoying errors, and if you do, post them at MacNN's OS X UNIX forum or ask a friend. It's often convenient to bounce errors off people to get some ideas.

11 Process Management

Every app you run spawns a new process and is given a PID (process ID). For right now, we can just cover killing off apps (sometimes Force Quit doesn't do it). Let's assume OmniWeb is the offending app for right now (random choice). Type:

```
[prompt] ps aux | grep Omni | grep -v grep
```
or
```
[prompt] ps auxc | grep "Omni"
```

This is a piped command. What it does is that the shell first executes ps aux, and then uses the output of that command as input for the command `grep 'Omni'`. The grep command is basically a search through text tool, so it will basically search through the listing of processes for any app with "Omni" in the name. Hopefully, you'll get OmniWeb's listing pop up, along with it's PID. To kill it, just type `kill PID` and insert the PID number. For example, `kill 419`. That should knock it off. If even that doesn't work, you could use `kill -9 419`, which basically means "kill with extreme prejudice". If that can't quit it, nothing can. Process Management can be quite complex, if you have several suspended tasks running (with bg and fg and related utilities), but if you don't spend a lot of time with UNIX, those commands aren't very useful. If you're running a UNIX program that you don't know how to quit, type Control-C to quit it.

(Note: If you're wondering why I used "ps auxc" and not "ps aux", it's because piping ps aux into a grep statement will also list that grep command as part of the output, which is annoying. ps auxc works better in my experience if you're piping to grep. A reader pointed out that a better solution is the second one, as that does not mess with the ouptut that ps aux gives you, but rather removes any process with "grep" in it from the listing.)

12 Introduction to Text Editors: Pico and Emacs

As said before, pico is a text editor. It's the friendliest and easiest to use text editor in UNIX. Text editors become very, very powerful but they also become very, very complex. In this FAQ, we'll cover the basics of pico and emacs, as they're (in my humble opinion) the most useful editors.

12.1 The Pico Editor

pico is easily the simplest and most user-friendly of all the UNIX editors, and as a result, most users who are new to UNIX choose pico as their editor of choice. To launch pico, simply type:

```
[prompt] pico (filename)
```

The (filename) implies that providing an input file when launching pico is optional; if you just launch pico normally it'll open a new document. The basic commands are Control-O for save, Control-X for quit, arrow keys for navigation, Control-W for find, and Control-G for the quick and excellent built-in help system. The help will walk the user through all the other commands available in pico (cut/copy/paste, etc), but for most simple documents, those few commands should be enough for basic literacy.

12.2 The Emacs Environment

Emacs is an amazingly large subject, and for a complete reference the GNU Emacs manual is an excellent tutorial at *http://www.gnu.org/manual/emacs/*. However, learning the few basic commands in Emacs is not very difficult, and can serve you well when Pico is not up to the task and the GNU manual is overkill.

Firstly, a quick briefing of the Emacs "GUI" is necessary. The command-line version of Emacs basically has buffers and windows, and you use the minibuffer to communicate with the editor and issue various commands. When you have "opened" a file in Emacs, what the editor has actually done is load the contents of the file into a temporary buffer (until it is saved to disk), so the actual file is not being edited. You have one buffer per file, but you can split the display so that there are two windows looking at different parts of the same buffer. Alternatively, you can have two windows up looking at different buffers, of course. You can have an infinite number of windows, although more than four or five gets extremely unwieldy. Having a dozen buffers open, however, is often rather convenient, as jumping between them is fast and easy.

Also, some explanation of Emacs' feature set is necessary. Emacs is organized by "modes." Basically, for different kinds of editing, there are different modes you use. There are major modes and minor modes. The main difference is that you can only have one major mode enabled at a time, while you can have several minor modes enabled. On the whole, major modes don't overlap, while minor modes provide functionality useful in many different contexts. For example, for editing text, you use Text Mode. For writing Java code, you use Java Mode. For writing C++ code, you use C++ Mode. And so on and so forth. There are Lisp modes, C modes, and modes for almost any significant language. In different modes, Emacs automatically changes various settings to make coding in that specific language more streamlined. However, there are also Calendar modes for viewing and organizing your calendar, Diary mode for using the built-in organizer, debuggers, makefiles, and so many others that it would be impossible to cover them all.

However, since Emacs has so many features, and the mouse is unsupported, there are a ridiculous number of key bindings. There are so many key bindings, in fact, that some of them use Control keys (C-h means Control-H in Emacs syntax), while others use the Meta key (M-x means Meta key-X in Emacs). In Mac OS X, the meta key is simply the Escape key (however, in Mac OS X 10.1, there's an option to set the option key as the meta key; for now, though, stick with Esc). Also, even this number of key bindings weren't enough, so they had to have **multiple key bindings**. For example, typing C-x C-c means hold down Control, then type X, then C (holding down control the entire time). That quits Emacs. However, this also implies that C-x C-b is different from C-x b. In the former, Control is held down while you press B, while in the latter, you release control after pressing X but before pressing B. These commands do completely different things, also. C-x C-b lists all the buffers Emacs has at that time, while C-x b lets you switch buffers (the minibuffer will prompt you to type in the name of the buffer, although you can use Tab to autocomplete). Meta keys work the same way, on the whole. However, there are a large number of commands in Emacs that aren't even assigned key bindings, so you have to type out rather long commands, in some cases. For example, M-x compile is the command you can use to compile something within Emacs; M-x gdb is what you use to load GUD and run gdb on a binary within Emacs; M-x copy-region-as-kill will copy a region of text; and so on and so forth.

Some useful commands are C-x C-s for save, C-x C-f to visit (open) a file, C-x o to switch windows, C-x 1 to close all other windows, C-x 2 to split a window horizontally, C-x] for end of file, C-x [for beginning of file, C-a for beginning of line, C-e for end of line, C-k for kill line (kill is like cut), C-y to yank (paste) whatever you last killed, C-space to mark the beginning of the region, M-x copy-region-as-kill to copy an entire region (from the mark to the current location of the cursor), C-x C-b to list all buffers, C-x b to switch buffers, C-x k to kill the current window, C-x C-z to suspend Emacs, and C-x C-c to quit. Some commands for switching major modes are M-x text-mode, M-x c-mode, M-x calendar, M-x diary, and so forth. C-h stars the built-in (and extremely extensive) help system, and C-h a is just one of the options in it (apropos, which is the most convenient). For example, if you want to find out how to launch an interactive shell within Emacs, you could type C-h a shell, and this would tell you that M-x shell opens up a shell.

While this is anything but an in-depth look at Emacs, it should give you the fundamentals that lets you navigate around the environment (because it's so much more than an editor, really) and be on your way to learning your way around Emacs. For any additional information, the GNU Manual should be all you need (it has in-depth coverage of everything from saving and quitting to using the powerful built-in versional control system).

13 Introduction to X Windows / X11

Q: What is the X Windows System?
A: The X Windows System is a GUI framework. It is nothing like the Mac OS. Xfree86 is by far the most common and most popular. Neither Darwin nor Mac OS X include an X11 distribution (like Xfree86, which is free, as the name implies). You can basically run any of numerous "window managers" for X Windows; a window manager is basically a complete, self-contained GUI. Different GUIs have different styles, different menu management and are more than just themes for the same interface – rather, they're separate UIs altogether.

Q: Why do I need X11?
A: Any UNIX apps with a GUI (such as the GIMP) require X11. If you're not interested in running any of these programs, then you don't need X11.

Q: What are these "window managers" I hear about?
A: Window Managers run on top of the X Windows System, providing a much more robust and usable GUI. Some are more complex than others. The more famous window managers are AfterStep, WindowMaker, and Enlightenment. Gnome and KDE are desktop environments and run on top of (underneath?) a window manager. The window manager handles how windows are drawn on screen - how the title bar looks, etc. There's a lot more than that in Gnome and KDE.

Reader Esme Cowles offers the following explanation: "Desktop environments run on top of the window manager, in that they usually handle the task bar, program menu, etc. They also usually run services in the background, too.

"For window managers, I think it's important to say that the inside of the window is controlled by the program, but the borders and placement is handled by the window manager. Most people

who aren't familiar with UNIX are really surprised by this, so I usually explain that it's so an application can be running on one machine, with it's display on another machine, but still have the window borders and stuff fit in on the display machine."

Q: How do I get an X Windows System installed on Mac OS X?
A: There are two ways: Download and install the XFree86 binaries from www.xfree86.org or buy a prebuilt X Windows from Tenon (www.tenon.com), Xtools. Xtools is extremely overpriced, but lets you run X windows side-by-side with aqua windows - they behave like normal Aqua windows and you can have, say, the GIMP running next to OmniWeb. XFree86 with the patch from *http://mrcla.com/XonX* lets you run XFree86 side-by-side with Aqua, and now, rootless patches are available so you can run X Windows applications and Mac OS X applications side-by-side. These patches should be rolled into Xfree86 4.1 fairly soon, and once these rootless patches mature, running X11 programs should become more and more convenient in Mac OS X.

More information on X11:
http://www.xfree86.org
http://www.mrcla.com/XonX
http://xwinman.org
http://fink.sourceforge.net

14 Conclusion

That's all for now; I hope you found this useful. I'd love to hear about your experiences with the FAQ, or anything you'd like to see improved or added.

This Terminal Basics FAQ will be updated via user comments and will be kept up to date at the following location:

http://homepage.mac.com/rgriff/TerminalBasics.pdf (preferred)
http://www.oddballnews.com/misc/TerminalBasics.pdf