Getting started with a terminal emulator for SEO programming

While it’s true that you can access abundant resources through your operating system’s GUI, a terminal emulator window will provide you access to command line utilities including notably important network-centric ones for technical SEO and programming. There’s a lot of power and much more to see when you’re working “under the hood” of your computer.

In this installment of our series on programming for SEO, we’re going to open a terminal, navigate and explore your local computer, discover what resources you have, and determine what you are permitted to do in cases where your workstation is managed by someone else.

Opening a terminal

If you’re working on a Mac, simply use Spotlight (command-spacebar) to search “terminal” and the top hit should be: terminal.app. Launch it. When you want to close the window, it is wise (but not critical) to type “exit” so that bash and other child processes close before the GUI window closes. The “bash” process is a program launched by default for the Mac terminal application and uses your keyboard input to run utilities and programs on your computer in the same sense as when people in the 1970s worked at terminal stations operating huge mainframe computers.

If you are working with Windows you’ll want to install the Windows Subsystem for Linux and use a Linux terminal

application. It’s neat to see which Linux distributions Microsoft has chosen (including Kali!). You should know that as a major part of the open source community, Linux has been customized several times by groups of like-minded people into literally thousands of versions. For example, Kali Linux is highly optimized for security penetration testing teams (security hackers). Ubuntu is the most commonly used Linux distribution and we recommend you stick with Ubuntu for SEO.

Recursively: Gnu is not Unix

As mentioned, Linux is a huge part of the open source community. It’s widely used not only for its device drivers but also because its operating system is powered by Gnu programs, to some extent. Gnu is not Unix (a clever, recursive name) so there are differences between the MacOS (Unix) programs and their correlating Linux ones. You’ll be relieved to know that any differences present minimal issues. We’ll highlight how to accommodate them as we go about learning SEO Programming.

You are here

Now with a terminal window open, let’s orient ourselves. A dollar sign ($) denotes a shell (bash) command line prompt, which is what you should see in your terminal window. For the purposes of this article, a single line of text after the dollar sign will denote a command with that text and hitting <enter>. Sometimes these “single-line” commands can get pretty long and wrap, but we’ll start with very short ones. The first command we’ll run is going to be about finding where we currently are in the file system. We’re going to ask bash to “Print the Working Directory” with “pwd <enter>” as follows:

```
~ $ pwd
/Users/username
```

```
~ $
```

“/Users/username” is the result of the command, and another prompt appears beneath it.

Home directory

If you’re wondering what the tilde (~) means, it is a Unix/Linux symbol for the user home directory. Therefore, the computer’s answer to $ pwd <enter> was to print back to our screen the home folder location from the root of the file system. In this case, it is the MacOS “Users” directory followed by the logged in username which is the default starting location when you open terminal. The Users directory is how MacOS organizes Users. In Linux, it is typically named “Home” instead of Users.

Navigating your system with your terminal means you’ll be changing locations using commands. The idea is analogous to clicking or double-clicking folders using MacOS Finder or Windows File Explorer. The difference is that there isn’t as much visual feedback. Now we’re going to issue the “change directory” (cd <directory name>) command:

```
~ $ pwd
/Users/username
~ $ cd /random/directory
/random/directory $
```

Notice the command prompt now appears after the current directory, rather than after the home directory designated by the tilde (~). In case you ever feel lost you can always use cd with the tilde symbol ($ cd ~ <enter>) as a shortcut for returning to your home directory:

```
/random/directory $ cd ~
~ $
```

As you can see, commands are shortcuts and abbreviations of

common tasks. For example, to “list” what’s in the current directory (ls), “change directories” (cd), “make a new directory” (mkdir), “copy” a file (cp), “move” a file (mv), the associated program commands are meant to become intuitive.

Most of these programs require additional information from you in order to work properly. The change directory and make new directory (mkdir) commands require that you input, or “pass” a name for the new directory as in $ mkdir <directory name>. Both copy (cp) and move (mv) require additional input, as well. You need to name the original file and the resulting new filename for these programs or they will return with an error. Program input consists of “arguments” separated by spaces. When spaces appear in one such argument, use quotes to encapsulate it.

Here is another example of entering the same command to return to the home directory we entered earlier, but with a comment:

```
$ cd ~ # let's head back home
```

The command above has a tilde (~) as the argument for change directory and a hash (#) in the text which is meant to allow us space for descriptive “comment” text that won’t get processed by the shell interpreter. Comments in code are an integral part of programming. It’s important to learn how to write them for other programmers and especially for yourself where you might return to working on old code at a later date and could use helpful reminders about what code accomplishes what.

Error messages are your friends

When you try something and encounter an error, such as required information is missing or incorrect, the program will quit early and respond with an error message. Use error message text to search for answers when you don’t immediately know what is going wrong for you. Error messages are your key

to unlocking what you're trying to accomplish whenever you're stuck. Make friends with the feeling you get with error messages and use your search capability to get unstuck.

Let's back up a step. Trying to change to a directory name that doesn't exist will produce an error message and this can happen fairly frequently:

```
~ $ pwd
/Users/username
~ $ cd /random/directory # change directory
-bash: cd: /random/directory: No such file or directory
~ $
```

What's happening here is that your bash session is presenting the error text "no such file or directory" when the system lookup failed to find "/random/directory" via the change directory program (cd). Use this information by reading it from right to left to debug what happened. In this case, it should be relatively easy in that you should recognize the computer has no such directory "/random/directory." Look at the spelling of your input to programs to look for what might have gone wrong.

Table of contents

Now let's ask bash to tell us what subdirectories we have above us and what files are located in the current directory. We use the list program (ls) for this. The default output is not the nicest representation of folder contents, but it's accurate.

In a future installment of this series, we'll configure list to be more friendly for our purposes and reveal hidden information. For now, in your home directory you should see familiar looking subdirectories like Applications, Desktop, Documents etc. These are the same directories you will see in Finder or Windows Explorer. Using the list command can help in

many situations, for example if you forget the correct spelling of a directory and get the “No such file or directory” error message.

Super-user permissions

When you are a system administrator (administrator user) of your Mac or Linux computer, then you are able to perform things as the “root” user, which has far more privilege to do a wide range of things including changing configuration settings. When misconfigured, your computer can lock you out, so we’re going to caution that to avoid problems take extra care when acting as a super-user, which is going to be relatively rare for our purposes.

Super-user do

The command to run a command with super-user privilege is $ sudo (followed by a command). Sudo is shorthand for “super-user do.” When you try list with sudo you will be prompted for your password:

```
~ $ sudo ls
Password:
```

If you list the contents of your current directory with sudo then you should get a list of more files than you do without sudo. You should see “hidden files” which have a name preceded by a “dot” (.) character. Dot files are not hidden for any special reason except to hide them from users who are not administrators. That is because dot files are sometimes used for configuring programs which is a system administrator’s task. For example, the “.bashrc” text file is used to run commands when bash starts up. The “rc” in the filename .bashrc stands for “run commands.”

```
~ $ sudo ls
```

```
Password:
.CFUserTextEncoding
.DS_Store
.
.
.
.bashrc
.
.
. etc.
~ $
```

If after you enter your password you get a “permission denied” error message, then the current user is not configured as an administrator with root user privilege. If this is the case for you, you can try Login Options on your Mac to give yourself administrator capabilities, or ask your IT department to do so. If you are out of luck and have to operate as a normal user then don’t worry, you will still be able to do just about everything we’ll be covering and you will still have the ability to use sudo commands in the cloud.

Commands to know

There are a handful of commands listed below that should be committed to memory:

```
$ sudo (super user do)
$ ls (list contents)
$ cd (change directories)
$ touch (makes a blank file)
$ mkdir (makes an empty directory)
$ cp (copy files with options to copy directories)
$ mv (moves or renames files with options for directories)
$  rm (destructively removes files with options for directories)
$ head / tail (prints the first few, or last few lines of a
```

file)
$ cat / tac (prints a file forwards or backwards)
$ grep (globally search a regular expression and print)

What’s next?

In the next installment of this series, we’ll be exploring these commands along with how to piece commands together and construct more complex computing tasks using pipes (|). We will navigate the Unix user manual, available at your fingertips, by way of the command line interface so that you can find commands on your own to solve specific problems.

Without some guidance, the highly detailed information is often difficult for new users to comprehend, so it is our intention to provide you with the resources to use the manual effectively. We will also introduce you to a programmer’s text editor known as vi, a crucial program you will need to know, going forward.

More development tips for SEOs

About The Author

Detlef Johnson is Editor at Large for Third Door Media. He writes a column for Search Engine Land entitled “Technical SEO for Developers.” Detlef is one of the original group of pioneering webmasters who established the professional SEO field more than 20 years ago. Since then he has worked for major search engine technology providers, managed programming and marketing teams for Chicago Tribune, and consulted for

numerous entities including Fortune 500 companies. Detlef has a strong understanding of Technical SEO and a passion for Web programming. As a noted technology moderator at our SMX conference series, Detlef will continue to promote SEO excellence combined with marketing-programmer features and webmaster tips.

http://feeds.searchengineland.com/~r/searchengineland/~3/mVv64qpSduA/getting-started-with-a-terminal-emulator-for-seo-programming-322072