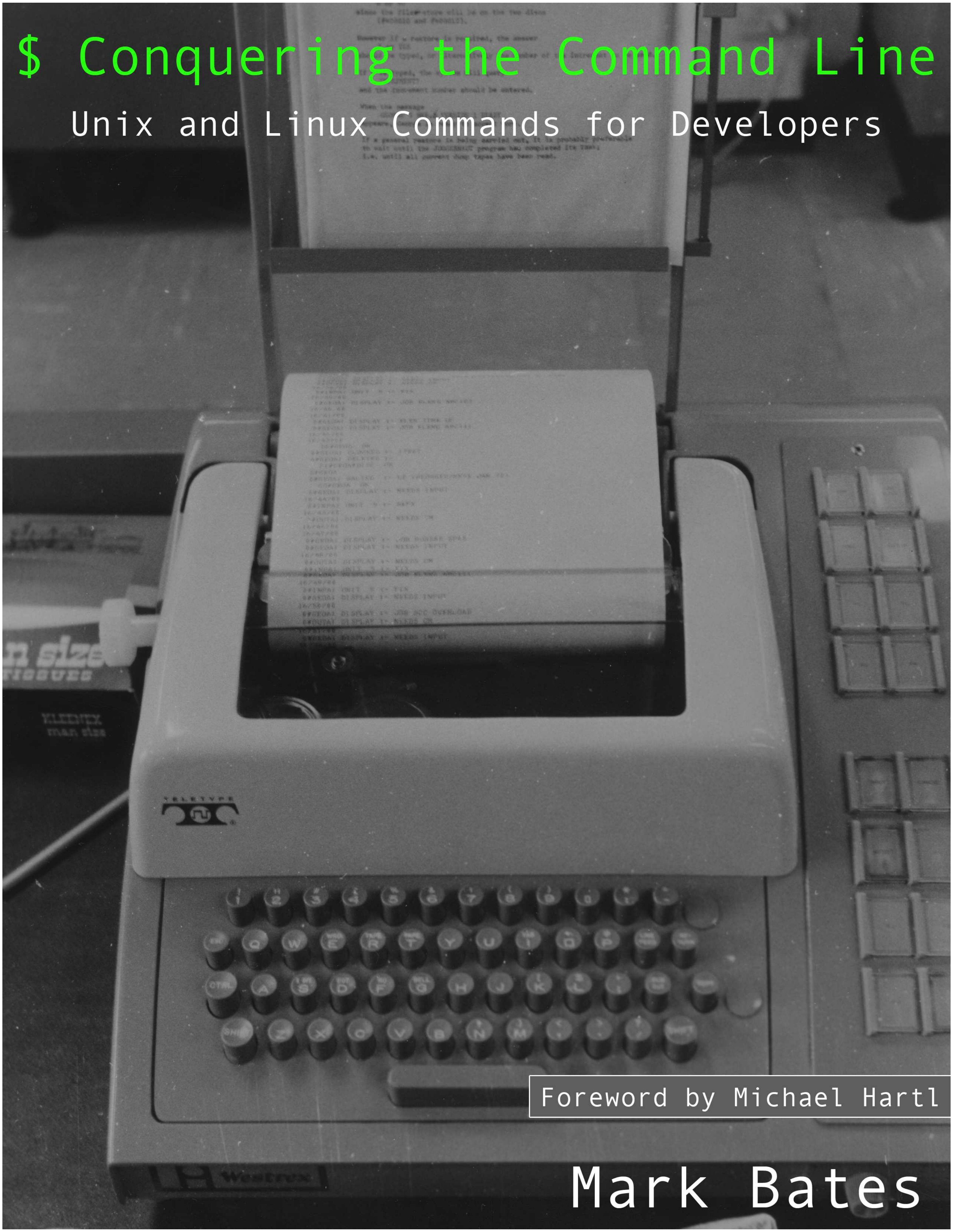
$ Conquering the Command Line

Unix and Linux Commands for Developers



Foreword by Michael Hartl

Mark Bates

Conquering the Command Line

Unix and Linux Commands for Developers

Mark Bates

Contents

Foreword

I distinctly remember, as a freshman in college, asking a local guru how to rename a file in Unix. "It's **mv**," he said "for 'move'." To me, he was a Unix god, and here I was not even knowing **mv**. Hey, we all have to start somewhere.

No matter where *you're* starting, Mark Bates's *Conquering the Command Line* is for you. Mark is an expert developer, lucid writer, and acclaimed speaker, and I can't think of a better guide to the ins and outs of the Unix command line from the perspective of a *working developer* rather than a Unix sysadmin. Mark's approach is not to give you an exhaustive and overwhelming catalog of each command and everything it can do, but rather to carefully select the most important commands and the most useful options. In other words, this is a *curated* introduction to the Unix command line.

Conquering the Command Line starts with the essential basics—things like **pwd**, **ls**, and **cd**—and then takes you on a tour of the more exotic animals in the Unix zoo. The result is that you'll quickly move past **mv** to master commands like **curl**, **grep**, **find**, and **ack**—not to mention **ps**, **tar**, and **sed**. Because of the ubiquity of Unix, the knowledge you gain will serve you well whether you use OS X, Linux, or *BSD. (And if you're stuck running Windows—well, you can always install a Linux virtual machine.)

I certainly don't consider myself a Unix god, or indeed any other kind of OS deity. For all I know, neither did that local guru. But we can all aspire at least to *demi*god status, and *Conquering the Command Line* will help get you there. Please remember, though, to use your new-found Unix powers for good, not for evil—with great **sudo** comes great responsibility.

Michael Hartl
Author, *The Ruby on Rails Tutorial*

Preface

Who is This Book For?

This book is for new developers, experienced developers, and everyone in between who wants to master Unix and Linux commands. This book was designed to showcase some of the most useful commands that a developer can know to help them in their daily tasks.

This book is not an exhaustive manual for each of these commands; in fact, it is just the opposite. Instead of just collecting up MAN pages, this book pulls out the most useful flags, options, and arguments for each of the commands in the book.

By learning and understanding the subset of flags, options, and arguments for each command, you'll be able to more efficiently use each one in your daily development workflow - while understanding where to look to find the more esoteric options you'll only need occasionally.

Code Examples

For most of the examples in this book we will use the Rails source code as a way to demonstrate a lot of the commands we will look at.

The source code for Rails can easily be downloaded from GitHub using Git.

```
$ git clone git@github.com:rails/rails.git
$ cd rails
```

If you don't have Git installed, which you should, you can download the zip of the Rails source code at https://github.com/rails/rails/archive/master.zip.

Because Rails is an ever changing code base, your results may vary from those in the book as the source code changes.

Acknowledgments

Books are not written in a vacuum, and this book was no exception. It is easy to assume that since there is only one name on the cover of this book, that there is only person responsible for bringing it all to fruition - when in fact, this couldn't be any further from the truth.

I have a lot of people to thank for helping to get this book in your hands, yourself included. If I didn't think people such as yourself would want to read this book, I wouldn't have written it, so thank you so much.

I would like to thank Michael Hartl, and the entire team at Softcover, for developing a great set of publishing tools meant for people such as myself, the technical writer. I have written two books prior to this one, and I have to say that I have never used tools as good as these for writing a book before. The platform just got out of my way and let me focus on writing, and not on constantly trying to figure out how to format things, why my PDF wasn't generating correctly, etc...

Next I would like to thank my fantastic team of technical reviewers. The tech reviewers on this book were Pat Shaughnessy, Michael Denomy, Pete Brown, Johnny Boursiqout, and Dan Pickett. They tirelessly waded through each chapter as I wrote it, and they submitted some amazing pull requests to improve both the quality of the content as well as the prose itself. Their questions, suggestions, subtractions, and additions, made this book significantly better than I could have ever written on my own. I'm eternally grateful to each of them for donating their time, and mindshare, so freely. If you see them at a conference, user group, or in the street, give them a big hug and say thanks.

Lastly, the most important thank you I can give is to my family. I have to be the luckiest man on the planet to have such an amazing, and understanding wife. She supports me, she pushes me, she believes in me, and most importantly, she makes my life better. I can't express how much I owe to her for all she has done and given to me, including our two beautiful boys. Thank you Rachel, as always, this book is dedicated to you.

Well, enough gushing - I just hope you enjoy this book, and that it helps you as much as I hope it will. Enjoy.

Introduction

The story of how this book came to be started in April of 2013 at RailsConf in Portland, Oregon. At RailsConf I was hanging out with Michael Hartl, author of the Rails Tutorial. Michael and I had known each other for a few years through conferences, as well as both being authors for Addison-Wesley.

Over dinner one night, Michael started telling me that he was thinking of building a self-publishing platform for technical writers, and was pumping me for information about what kind of a platform I would like to write books on. I left RailsConf not thinking too much more about Michael's idea, as I had vowed to never write another book.

Fast forward a couple of months to June, 2013. I was in Los Angeles for a friend's wedding. I drove up to Pasadena one night to meet Michael for dinner, and the strongest Margaritas I have ever had. During dinner Michael, as he's prone to do, pulled out his laptop and started to demonstrate his publishing platform to me. I have to admit, I thought it looked really nice. He proceeded to spend the rest of the night trying to convince me that I should be a guinea pig for him and write another book, this time using his platform.

After I left Los Angeles, I headed back to Boston, packed quickly, and boarded a plane to Stockholm with my wife and kids, to spend the entire month of July working for a client I had there.

While at my client's office, I had several conversations with some developers who had no idea how to do even those most basic of commands on their systems. They were petrified of the command line. They could do simple stuff like change a directory, move a file, etc., but they couldn't un-`tar` a file, or find runaway processes. I found this incredibly frustrating as most developers spend their whole day in a terminal window, and here was a whole group of people that weren't getting the most they could from the powerful tools at their disposal.

This lack of command line knowledge bothered me for several days. I was obsessed with how these developers could get this far, without this knowledge. I started to ask myself questions such as how had I acquired these skills? The answer was I was lucky enough to have great mentors over the years who passed on tiny snippets of commands that were incredibly useful. I also thought about what would keep people from learning how to use these commands.

I came to the conclusion that one of the biggest obstacles for developers, particularly new developers, is the system that is supposed to help developers understand these commands: `man` pages. `Man` pages (`man` being short for manual) are supposed to be manuals that explain these commands in great detail - including every last option and flag. They are daunting, overwhelming, and confusing beasts that bewilder even the most "senior" developer.

This realization came to me while I was standing in line to buy tickets to the Vasa Museet. I realized what was needed was a book that picked out the most useful commands a developer needs every day. On top of that, the most useful flags and options need to be explained in simple terms, with clear examples, that can be of immediate use for a working developer.

As I stood in line, I shot off an email to Michael explaining my idea and to get his thoughts on the subject matter and whether or not he thought it would be a good fit for his Softcover platform. It was seconds before he responded with an overwhelming "YES!", and that was how I became the first author to publish on Softcover.

I loved writing this book. It wasn't always easy, no book ever is, but it was incredibly educational. I found new tools, new flags, and new options that have radically changed the way I work everyday. I have found myself referring to this book, while I was writing it. I have incorporate all of these tools into my daily work flow, and I hope you do too.

So with that said, let's dive right in.

Mark Bates
January, 2014

Chapter 1

Basics and Navigation

It's important to have a firm grasp on the basics of how to use the Unix (or Linux) command line. This includes navigating around directories, copying files, creating files, etc.

For the more experienced developer, this chapter may be common knowledge to you. If that's the case, you can skip it, although you might pick up one or two little things you didn't already know.

For newer developers, this chapter contains information that will become muscle memory after a while. You *must* know everything in this chapter just to be able to do the most basic of things.

The rest of this book will assume you understand what is in this chapter.

1.1 Home Directory (`~`)

On Unix and Linux systems, every user has a "home" directory. This directory is yours and yours alone. The location of your home directory varies depending on which operating system you're on. For example, on Mac OS X it's **`/Users/<username>`**, but on Ubuntu systems, it's **`/home/<username>`**.

Finding your home directory is incredibly easy. When you first log into your system you will automatically be placed in your home directory by default. With this knowledge, you can use the **pwd** command (Section 1.2) to get the path of your home directory.

Listing 1.1: `$ pwd`

```
/Users/markbates
```

Your home directory will be where your settings files get stored and where your desktop is kept. The home directory (or subdirectories under your home directory) is also where you will save documents and files that are yours and not to be shared with others.

By default, your home directory will be unreadable and inaccessible to all other users apart from yourself (and root). You can change these permissions, if you want, but I would recommend against making such a change.

It is quite common to see a tilde (**~**) in place of the full path for a home directory. The **~** will expand to be the path of your home directory when you use commands such as **cd** (Section 1.5).

1.2 Present Working Directory (pwd)

Understanding where you are at any given time in your file system is important. This information can help you build paths to other files or help you if you get lost. To find out where you are you can use the **pwd** command.

```
$ pwd
```

Listing 1.2

```
/Users/markbates/development/unix_commands
```

In Listing 1.2 you can see the directory I'm currently working in. For me, it's the directory that I'm writing this book in.

By default, `pwd` will list the "logical" path of your current directory. This means it will treat symlinked paths (see Section 1.4.3) as if they were the actual paths. For example, my `development` directory doesn't exist under my home folder, `/Users/markbates`; instead it is symlinked to a folder inside of `/Users/markbates/Dropbox`.

If we want to see the actual physical path, with all of the symlinks resolved, we can use the `-P` flag.

```
$ pwd -P
```

Listing 1.3

```
/Users/markbates/Dropbox/development/unix_commands
```

Listing 1.3 shows the actual full path to my current working directory.

1.3 List Files and Directories (`ls`)

Often it is useful to be able to see what files or directories are in your current directory, or another directory. To accomplish this we can use the `ls` command.

```
$ ls
```

Listing 1.4

```
Applications    Calibre Library Desktop         Documents       Downloads
Dropbox         Library         Movies          Music           Pictures
Public          Sites           development     screencasts     scripts
tmp
```

Listing 1.4 shows the list of folders and files in my current directory, which for me would be my home directory, `/Users/markbates`. I will be using my home directory for all of the examples in this section, unless otherwise noted.

1.3.1 Listing a Different Directory

By default, `ls` will operate in the current directory that you're in. However, you may want to find out what files are in another directory, without first leaving your current directory. The `ls` command will let you pass it a path to work on.

```
$ ls /usr/local/
```

Listing 1.5

```
CONTRIBUTING.md Cellar          Frameworks      Library         README.md
SUPPORTERS.md   bin             etc             foreman         go
heroku          include         lib             opt             sbin
share           texlive         var
```

Listing 1.5 shows the files and directories inside of the `/usr/local` directory, without having to leave the current directory that I am in.

1.3.2 Listing All Files (`-a`)

So far we have seen all of the "visible" files and directories listed, however, it is common on Unix and Linux based machines to have "invisible", or "hidden", files that are prefixed with a "`.`" (dot).

Using the `-a` flag we can see both "visible" and "hidden" files and directories.

```
$ ls -a
```

Listing 1.6

```
.                    .Trash               .bashrc              .ctags
.el4r                .erlang.cookie       .go                  .mplayer
.psql_history        .rvm                 .swt                 .zshrc
Dropbox              Sites                ..                   .Xauthority
.cache               .cups                .el4rrc.rb           .gem
.guard_history       .netrc               .railsrc             .rvmrc
.vim                 Applications         Library              development
.CFUserTextEncoding  .adobe               .cisco               .dbshell
.emacs               .gemrc               .gvimrc              .node-gyp
.redis-commander     .scala_history       .viminfo             Calibre Library
Movies               screencasts          .DS_Store            .anyconnect
.cloud66             .dropbox             .emacs.d             .git-completion.sh
.heroku              .npm                 .rediscli_history    .softcover
.vimrc               Desktop              Music                scripts
.MakeMKV             .bash_history        .codeintel           .dvdcss
.ember               .gitconfig           .inkscape-etc        .pry_history
.rnd                 .ssh                 .z                   Documents
Pictures             tmp                  .NERDTreeBookmarks   .bash_profile
.config              .editedhistory       .ember-data          .gnome2
.irb-history         .pryrc               .rspec               .subversion
.zlogin              Downloads            Public
```

Listing 1.6 now shows significantly more files and directories than Listing 1.4 because the `-a` flag is including all of the "hidden" files, as well as the "visible" ones.

1.3.3 Long Form Listing (`-l`)

So far in the section, whenever we have used the `ls` command, we have been given back a list of names, arranged in a column. One of the most useful flags for `ls` is the `-l` flag that will list out the names of the files and directories as well as give more detailed information about them.

```
$ ls -l
```

Listing 1.7

```
drwxr-xr-x   16 markbates  staff   544 Aug 19 13:58 Applications
drwxr-xr-x    5 markbates  staff   170 Dec  6 16:20 Calibre Library
lrwxr-xr-x    1 markbates  staff    33 Jul 26  2012 Desktop ->
  /Users/markbates/Dropbox/Desktop/
lrwxr-xr-x    1 markbates  staff    35 Jul 26  2012 Documents ->
  /Users/markbates/Dropbox/Documents/
drwx------+ 140 markbates  staff  4760 Dec 16 13:44 Downloads
drwx------@  24 markbates  staff   816 Dec 15 17:24 Dropbox
drwx------@  66 markbates  staff  2244 Oct 23 08:29 Library
drwx------+   5 markbates  staff   170 Sep 15 18:06 Movies
drwx------+   6 markbates  staff   204 Mar  5  2013 Music
drwx------+  10 markbates  staff   340 Jul 31 18:20 Pictures
drwxr-xr-x+   4 markbates  staff   136 Mar  5  2013 Public
drwxr-xr-x+   3 markbates  staff   102 Mar  5  2013 Sites
lrwxr-xr-x    1 markbates  staff    37 Jul 26  2012 development ->
  /Users/markbates/Dropbox/development/
lrwxr-xr-x    1 markbates  staff    37 Jun  1  2013 screencasts ->
  /Users/markbates/Dropbox/screencasts/
lrwxr-xr-x    1 markbates  staff    33 Jul 26  2012 scripts ->
  /Users/markbates/Dropbox/scripts/
drwxr-xr-x   19 markbates  staff   646 Dec  6 16:07 tmp
```

In Listing 1.7 we see a significant amount of information being presented to us for each file and directory.

If the `-l` option is given, the following information is displayed for each file: file mode, number of links, owner name, group name, number of bytes in the file, abbreviated month, day-of-month file was last modified, hour file last modified, minute file last modified, and the pathname. We can also see that some paths are followed by `->` and another path. These paths are symlinked to the directories that follow the `->`. See Section 1.4.3 for more information on symbolic links.

1.3.4 Human Readable Sizes (`-h`)

In Section 1.3.3 we used the `-l` flag to give us a more detailed view of files and directories in our current directory, including the number of bytes in the file.

While knowing the number of bytes can be useful, I find it more useful to see the size of the file in human readable terms, such as `1.7K` or `35M`. Thankfully `ls` will give us this information if we use the `-h` flag.

```
$ ls -lh
```

Listing 1.8

```
drwxr-xr-x   16 markbates  staff   544B Aug 19 13:58 Applications
drwxr-xr-x    5 markbates  staff   170B Dec  6 16:20 Calibre Library
lrwxr-xr-x    1 markbates  staff    33B Jul 26  2012 Desktop ->
  /Users/markbates/Dropbox/Desktop/
lrwxr-xr-x    1 markbates  staff    35B Jul 26  2012 Documents ->
  /Users/markbates/Dropbox/Documents/
drwx------+ 140 markbates  staff   4.6K Dec 16 13:44 Downloads
drwx------@  24 markbates  staff   816B Dec 15 17:24 Dropbox
drwx------@  66 markbates  staff   2.2K Oct 23 08:29 Library
drwx------+   5 markbates  staff   170B Sep 15 18:06 Movies
drwx------+   6 markbates  staff   204B Mar  5  2013 Music
drwx------+  10 markbates  staff   340B Jul 31 18:20 Pictures
drwxr-xr-x+   4 markbates  staff   136B Mar  5  2013 Public
drwxr-xr-x+   3 markbates  staff   102B Mar  5  2013 Sites
lrwxr-xr-x    1 markbates  staff    37B Jul 26  2012 development ->
  /Users/markbates/Dropbox/development/
lrwxr-xr-x    1 markbates  staff    37B Jun  1  2013 screencasts ->
  /Users/markbates/Dropbox/screencasts/
lrwxr-xr-x    1 markbates  staff    33B Jul 26  2012 scripts ->
  /Users/markbates/Dropbox/scripts/
drwxr-xr-x   19 markbates  staff   646B Dec  6 16:07 tmp
```

In Listing 1.8 we see we now have an easier to understand description of the size of the files in the directory.

1.3.5 Sorting by Size (`-S`)

Another useful flag we can use with `ls` is `-S`, which will sort the results of `ls` by file size, instead of the default sorting by name.

```
$ ls -lhS
```

Listing 1.9

```
drwx------+ 140 markbates  staff   4.6K Dec 16 13:44 Downloads
drwx------@  66 markbates  staff   2.2K Oct 23 08:29 Library
drwx------@  24 markbates  staff   816B Dec 15 17:24 Dropbox
drwxr-xr-x   19 markbates  staff   646B Dec  6 16:07 tmp
drwxr-xr-x   16 markbates  staff   544B Aug 19 13:58 Applications
drwx------+  10 markbates  staff   340B Jul 31 18:20 Pictures
drwx------+   6 markbates  staff   204B Mar  5  2013 Music
drwxr-xr-x    5 markbates  staff   170B Dec  6 16:20 Calibre Library
drwx------+   5 markbates  staff   170B Sep 15 18:06 Movies
drwxr-xr-x+   4 markbates  staff   136B Mar  5  2013 Public
drwxr-xr-x+   3 markbates  staff   102B Mar  5  2013 Sites
lrwxr-xr-x    1 markbates  staff    37B Jul 26  2012 development ->
  /Users/markbates/Dropbox/development/
lrwxr-xr-x    1 markbates  staff    37B Jun  1  2013 screencasts ->
  /Users/markbates/Dropbox/screencasts/
lrwxr-xr-x    1 markbates  staff    35B Jul 26  2012 Documents ->
  /Users/markbates/Dropbox/Documents/
lrwxr-xr-x    1 markbates  staff    33B Jul 26  2012 Desktop ->
  /Users/markbates/Dropbox/Desktop/
lrwxr-xr-x    1 markbates  staff    33B Jul 26  2012 scripts ->
  /Users/markbates/Dropbox/scripts/
```

In Listing 1.9 the results of `ls` are displayed with the largest files at the top, and the smallest files at the bottom.

1.3.6 Sorting by Last Modified Time (`-t`)

It can often be useful to be able to sort your `ls` results by the last time the files were modified. To do this we can use the `-t` flag.

```
$ ls -lt
```

Listing 1.10

```
drwx------+ 140 markbates  staff  4760 Dec 16 13:44 Downloads
drwx------@  24 markbates  staff   816 Dec 15 17:24 Dropbox
drwxr-xr-x    5 markbates  staff   170 Dec  6 16:20 Calibre Library
drwxr-xr-x   19 markbates  staff   646 Dec  6 16:07 tmp
drwx------@  66 markbates  staff  2244 Oct 23 08:29 Library
drwx------+   5 markbates  staff   170 Sep 15 18:06 Movies
drwxr-xr-x   16 markbates  staff   544 Aug 19 13:58 Applications
drwx------+  10 markbates  staff   340 Jul 31 18:20 Pictures
lrwxr-xr-x    1 markbates  staff    37 Jun  1  2013 screencasts ->
  /Users/markbates/Dropbox/screencasts/
drwx------+   6 markbates  staff   204 Mar  5  2013 Music
drwxr-xr-x+   4 markbates  staff   136 Mar  5  2013 Public
drwxr-xr-x+   3 markbates  staff   102 Mar  5  2013 Sites
lrwxr-xr-x    1 markbates  staff    33 Jul 26  2012 scripts ->
  /Users/markbates/Dropbox/scripts/
lrwxr-xr-x    1 markbates  staff    35 Jul 26  2012 Documents ->
  /Users/markbates/Dropbox/Documents/
lrwxr-xr-x    1 markbates  staff    33 Jul 26  2012 Desktop ->
  /Users/markbates/Dropbox/Desktop/
lrwxr-xr-x    1 markbates  staff    37 Jul 26  2012 development ->
  /Users/markbates/Dropbox/development/
```

1.3.7 Reverse Sort (`-r`)

When you are listing a directory that contains many files, sorting with **`ls`** is a great way to help quickly find the files or directories you are looking for. By default, **`ls`** sorts all of its results alphabetically. We also learned how to sort results by size (Section 1.3.5) and by last modified time (Section 1.3.6).

Using the **`-r`** flag we are able to reverse the results of **`ls`**.

```
$ ls -lr
```

Listing 1.11

```
drwxr-xr-x   19 markbates  staff   646 Dec  6 16:07 tmp
lrwxr-xr-x    1 markbates  staff    33 Jul 26  2012 scripts ->
  /Users/markbates/Dropbox/scripts/
```

```
lrwxr-xr-x    1 markbates  staff    37 Jun  1  2013 screencasts ->
  /Users/markbates/Dropbox/screencasts/
lrwxr-xr-x    1 markbates  staff    37 Jul 26  2012 development ->
  /Users/markbates/Dropbox/development/
drwxr-xr-x+   3 markbates  staff   102 Mar  5  2013 Sites
drwxr-xr-x+   4 markbates  staff   136 Mar  5  2013 Public
drwx------+  10 markbates  staff   340 Jul 31 18:20 Pictures
drwx------+   6 markbates  staff   204 Mar  5  2013 Music
drwx------+   5 markbates  staff   170 Sep 15 18:06 Movies
drwx------@  66 markbates  staff  2244 Oct 23 08:29 Library
drwx------@  24 markbates  staff   816 Dec 15 17:24 Dropbox
drwx------+ 140 markbates  staff  4760 Dec 16 13:44 Downloads
lrwxr-xr-x    1 markbates  staff    35 Jul 26  2012 Documents ->
  /Users/markbates/Dropbox/Documents/
lrwxr-xr-x    1 markbates  staff    33 Jul 26  2012 Desktop ->
  /Users/markbates/Dropbox/Desktop/
drwxr-xr-x    5 markbates  staff   170 Dec  6 16:20 Calibre Library
drwxr-xr-x   16 markbates  staff   544 Aug 19 13:58 Applications
```

In Listing 1.11 we can see that default sort of `ls`, alphabetically, has now been reversed with `tmp` at the top of the list, and `Applications` at the bottom.

We can also use the `-r` flag with other options to reverse their sort order. For example, `ls -lSr` will list the files by size with smaller files listed first.

1.4 Links (`ln`)

Links allow us to create an association from one file or directory to another. A link allows us to "point" one file or directory to another. This can be quite useful for maintaining a sane file system, or for having multiple versions of a file or directory, and not wanting to use disk space to store copies of those files.

An example of where links can come in handy is when deploying a web application. Let's say when you deploy your application you create a new directory whose name is the current time, and you put your application code into that directory. After your code is there, you would have to edit your web server configuration to tell it to point to the new directory where your latest code is. However, using links you can create a link called "current" that your web server points at, and you can change where the "current" link points to - in this case the last timestamped folder you created for your code.

Links can either be "hard" or "symbolic" as we'll see in the following sections.

1.4.1 Hard Links

We can use the `ln` command to create a link between two files. By default, the `ln` command will create a hard link between these files.

Hard links create an identical copy of the linked file on disk, that gets updated automatically as the source file gets updated. That means if the content of the source is changed, so will the target file.

To create a link to a file, we call the `ln` command - first giving it the name of the file we want to link to (our source file), followed by the name of the linked file we want to create (the target).

```
$ ln a.txt b.txt
```

Listing 1.12: `$ ls -l`

```
-rw-r--r--  2 markbates  staff  12 Dec 17 16:46 a.txt
-rw-r--r--  2 markbates  staff  12 Dec 17 16:46 b.txt
```

Listing 1.12 shows the newly created `b.txt` has all of the same properties as the `a.txt` file.

An important thing to note about hard links is that they only work on the current file system. You can not create a hard link to a file on a different file system. To do that you need to use symbolic links, Section 1.4.3.

While the content of the source and target files are linked, if the source file gets deleted, the target file will continue to live as an independent file, despite the severed link.

```
$ rm a.txt
```

Listing 1.13: `$ ls -l`

```
-rw-r--r--  1 markbates  staff  12 Dec 17 17:01 b.txt
```

In Listing 1.13 we can see that the `b.txt` file still exists, despite the target file having been deleted.

If we were to use the `cat` tool from Chapter 9 Section 9.2, we can see that the contents of `b.txt` are still intact.

```
$ cat b.txt
```

Listing 1.14

```
Hello World
```

1.4.2 Forcing a Link (`-f`)

If the source file already exists you will get an error, like that in Listing 1.15.

```
$ ln a.txt b.txt
```

Listing 1.15

```
ln: b.txt: File exists
```

To fix the error in Listing 1.15 you can pass the `-f` flag to force the link.

```
$ ln -f a.txt b.txt
```

1.4.3 Symbolic Links (-s)

We have seen that the default type of link that gets created when using `ln` is that of a hard link. Unfortunately hard links do not work for directories. To create a link to a directory, we can use the `-s` flag to create a symbolic link. This can also be used for linking to files as well, not just directories.

Symbolic links can also link to files or directories on other file systems. This makes symbolic links (symlinks) more powerful, and more common than the default hard links.

To create a symbolic link we can use the `-s` flag.

```
$ ln -s a.txt b.txt
```

Listing 1.16: `$ ls -l`

```
-rw-r--r--  1 markbates  staff  12 Dec 17 17:01 a.txt
lrwxr-xr-x  1 markbates  staff   5 Dec 17 17:12 b.txt -> a.txt
```

As Listing 1.16 shows we have created a symbolic link from `a.txt` to `b.txt`. See Section 1.3.3 for an explanation of how to read Listing 1.16.

Listing 1.17 shows the results of creating a hard link, and Listing 1.18 shows the results of a symbolic link.

Listing 1.17: `$ ln a.txt b.txt`

```
-rw-r--r--  2 markbates  staff  12 Dec 17 16:46 a.txt
-rw-r--r--  2 markbates  staff  12 Dec 17 16:46 b.txt
```

Listing 1.18: `$ ln -s a.txt b.txt`

```
-rw-r--r--  1 markbates  staff  12 Dec 17 17:01 a.txt
lrwxr-xr-x  1 markbates  staff   5 Dec 17 17:12 b.txt -> a.txt
```

Notice that the file timestamps and size are identical in Listing 1.17 with the hard link, but they are different in Listing 1.18 for the symbolic link. The reason for the difference is that with the symbolic link the operating system creates a new, small file that points to the target file or directory.

1.5 Change Directories (`cd`)

While staying in the same directory all of the time might seem like fun, eventually you'll need to change, or navigate, into another directory. To do this we can use the very simple `cd` command.

`cd` takes one argument - the path to the directory you wish to navigate to.

```
$ cd ~/Documents
```

Listing 1.19: `$ pwd`

```
/Users/markbates/Documents
```

In Listing 1.19 I have successfully navigated to my `Documents` directory, and we can use the `pwd` command (Section 1.2) to prove it.

1.5.1 Navigating Up (`..`)

Once inside of a directory, it can be useful to be able to navigate back up a directory. One way to do this would be to call `cd` giving it the full path to the directory you want to go. Another way to accomplish the task of moving up to the parent directory is to use the special path `..`.

```
$ cd ..
```

Telling `cd` to navigate to `..` will cause it to navigate to the parent directory of the current directory.

1.5.2 Navigating to the Home Directory

When you are in a directory and want to navigate back to your home directory, you can call `cd` without any arguments.

```
$ cd
```

Listing 1.20: `$ pwd`

```
$ /Users/markbates
```

1.6 Creating Directories (`mkdir`)

Directories are a great place to store and organize files, and creating them is easy. To create a directory we can use the `mkdir` command followed by the name of the directory we wish to create.

```
$ mkdir foo
```

Listing 1.21: `$ ls -l`

```
drwxr-xr-x  2 markbates  staff  68 Dec 18 13:20 foo
```

1.6.1 Create Intermediate Directories (`-p`)

The `mkdir` command will allow us to create nested directories using the `-p` flag.

```
$ mkdir -p a/b/c
```

Listing 1.22: `$ ls -la a/b/c`

```
drwxr-xr-x  2 markbates  staff   68 Dec 18 13:23 .
drwxr-xr-x  3 markbates  staff  102 Dec 18 13:23 ..
```

In Listing 1.22 we can use the `ls` command (Section 1.3) to see that the `c` directory exists and is currently empty.

1.6.2 Verbose Output (`-v`)

The `mkdir` command supports a verbose mode flag, `-v`. Adding the `-v` flag will print the results of `mkdir` to the console.

```
$ mkdir -v a
```

Listing 1.23

```
mkdir: created directory 'a'
```

Using the **-v** flag can provide useful reporting when writing scripts that will execute a lot of commands for you.

1.7 Copying Files (cp)

Copying files and directories can be accomplished using the **cp** command.

1.7.1 Copying a Single File

To copy a single file we can call the **cp** command with two arguments. The first argument is the file we want to copy, the source file. The second argument, or target, is the location we want to copy the source file to. This target can either be the name of the file you wish to copy to, or the directory you wish to copy to. If you specify just a directory then the original file name of the source file will be used to create the new file in the target directory.

```
$ cp a.txt b.txt
```

Listing 1.24: `$ ls -l`

```
-rw-r--r--  1 markbates  staff  0 Dec 18 13:34 a.txt
-rw-r--r--  1 markbates  staff  0 Dec 18 13:41 b.txt
```

1.7.2 Copying Multiple Files

cp will allows you to list several files you would like to copy. When you do this, however, the last argument *must* be a directory, and the original file names of the source files will be used as the names of the new files in the target directory.

```
$ cp a.txt b.txt foo
```

Listing 1.25: `$ ls -l foo`

```
-rw-r--r--  1 markbates  staff  0 Dec 18 13:47 a.txt
-rw-r--r--  1 markbates  staff  0 Dec 18 13:47 b.txt
```

We can also pass simple patterns to `cp` to achieve the same results as Listing 1.25.

```
$ cp *.txt foo
```

Listing 1.26: `$ ls -l foo`

```
-rw-r--r--  1 markbates  staff  0 Dec 18 13:47 a.txt
-rw-r--r--  1 markbates  staff  0 Dec 18 13:47 b.txt
```

1.7.3 Verbose Output (`-v`)

The `cp` command will print verbose output regarding the files that were copied using the `-v` flag.

```
$ cp -v a.txt b.txt
```

Listing 1.27

```
a.txt -> b.txt
```

1.7.4 Copying Directories (`-R`)

By default `cp` expects the source to be copied to be a file. If we try to copy a directory to another directory we will get the error we see in Listing 1.28.

```
$ cp foo bar
```

Listing 1.28

```
cp: foo is a directory (not copied).
```

To fix the error in Listing 1.28, we can use the `-R` flag to recursively copy the directory's contents to the new directory.

```
$ cp -Rv foo bar
```

Listing 1.29

```
foo -> bar
foo/a.txt -> bar/a.txt
foo/b.txt -> bar/b.txt
foo/c.txt -> bar/c.txt
```

Using the verbose flag, `-v`, we can see in Listing 1.29 all of the files from the source directory, `foo`, have been copied to the target directory, `bar`.

1.7.5 Force Overwriting of a File (`-f`)

In Listing 1.30 we can see that we have two files, `a.txt` and `b.txt`.

```
$ ls -l
```

Listing 1.30

```
-rw-r--r--  1 markbates  staff  0 Dec 18 13:58 a.txt
-rw-r--r--  1 root       staff  6 Dec 18 14:55 b.txt
```

When looking at Listing 1.30 we can see that the two files listed are owned by two different users. The first file, `a.txt`, is owned by user `markbates`, while the second file is owned by `root`.

What would happen if we tried to copy `a.txt` to `b.txt`?

```
$ cp a.txt b.txt
```

Listing 1.31

```
cp: b.txt: Permission denied
```

As we can see in Listing 1.31 we get an error when trying to overwrite `b.txt` with `a.txt`. We don't have permission to do that. When errors such as this occur we can use the `-f` flag to force the copying of the source file to the target file.

```
$ cp -f a.txt b.txt
```

Listing 1.32: `$ ls -l`

```
-rw-r--r--  1 markbates  staff  0 Dec 18 13:58 a.txt
-rw-r--r--  1 markbates  staff  0 Dec 18 14:57 b.txt
```

Listing 1.32 shows that by using the `-f` flag we were able to successfully overwrite the target file, `b.txt`, with the contents of the source file, `a.txt`.

1.7.6 Confirm Overwriting of a File (`-i`)

When you are about to copy over several files at once, and some of the target files already exist, it may be beneficial to confirm that you do in fact want to copy the source file and replace the target file.

`cp` will happily prompt you to confirm that you are about to overwrite another file if you use the `-i` flag.

```
$ cp -i a.txt b.txt
```

Listing 1.33

```
overwrite b.txt? (y/n [n])
```

In Listing 1.33 we are presented with a prompt asking us to confirm, `y` or `n`, whether to overwrite the target file. A response of `y` will overwrite the file. A response of `n` will skip the file and move on to the next copy, if there is one.

1.8 Deleting Files (`rm`)

The `rm` command is used to delete files and folders. It supports the same flags and arguments as the `cp` command that we learned about in Section 1.7.

```
$ rm -v a.txt
```

Listing 1.34

```
a.txt
```

Listing 1.34 uses the `-v` flag to list files that were deleted using the `rm` command.

1.9 Moving Files (`mv`)

The process for moving files is almost identical to that of copying files, which we learned about in Section 1.7.

`mv` supports the same flags as `cp`. In reality the `mv` command is really just a combination of `cp` and `rm` to achieve the desired outcome of moving a file from one location to another.

```
$ mv -v a.txt b.txt
```

Listing 1.35

```
a.txt -> b.txt
```

We can achieve the same effect from Listing 1.35 by using the `cp` and `rm` commands. This can be useful when trying to move a file across file systems, as the `mv` command doesn't support that.

```
$ cp a.txt b.txt
$ rm a.txt
```

Listing 1.36: `$ ls -l`

```
-rw-r--r--  1 markbates  staff  0 Dec 18 16:20 b.txt
```

Listing 1.36 shows that we were able to replicate the behavior of `mv` using the `cp` and `rm` commands.

1.10 Input/Output (`|`, `>`)

The Unix philosophy, which you'll hear repeated often, is "do one thing, and do it well". This theme is evident as we move through this book. Each command covered does one thing, and does it well.

1.10.1 Redirecting Output (|)

With these seemingly small individual commands, we can build some pretty impressive work flows by redirecting the output of one command to the input to another command. This is made possible by using the the "pipe" operator, `|`.

In the following example, we "pipe" the output of the `ls` command to the input of the `grep` command to find all the files in my home directory that contain an underscore, `_`.

```
$ ls -a ~ | grep _
```

Listing 1.37

```
.DS_Store
.bash_history
.bash_profile
.guard_history
.pry_history
.psql_history
.rediscli_history
.scala_history
```

When using the `|` operator we can chain together any number of commands. For example, we can take the output of Listing 1.37 and pass it to the `sed` command from Chapter 7 and change all of the underscores to dashes.

```
$ ls -a ~ | grep _ | sed "s/_/-/g"
```

Listing 1.38

```
.DS-Store
.bash-history
.bash-profile
.guard-history
.pry-history
.psql-history
.rediscli-history
.scala-history
```

Listing 1.38 shows the result of chaining together the `ls`, `grep`, and `sed` commands using the `|` operator.

1.10.2 Writing to a File (`>`)

In addition to redirecting the output from one process and sending it to another process, we can also write that output to a file using the `>` operator.

```
$ ls -a ~ | grep _ > underscores.txt
```

Listing 1.39: `$ cat underscores.txt`

```
.DS_Store
.bash_history
.bash_profile
.guard_history
.pry_history
.psql_history
.rediscli_history
.scala_history
```

Listing 1.39 shows the contents of the `underscores.txt` file (using `cat` from Chapter 9 Section 9.2), which contains the results of our search for files in the home directory that contain underscores.

1.10.3 Reading from a File (`<`)

In Section 1.10.2 we saw that we can use `>` to write data to a file. If we instead wanted to read data from a file, we can use `<`.

In Chapter 9, Section 9.6, we see a great example of this using the `pbpaste` command.

1.11 Conclusion

In this chapter we covered the basics of the command line, from how to navigate between directories, to how to move, delete, or copy files, and more.

Hopefully this chapter has provided you with the solid footing that you'll need to conquer the command line. The commands in this chapter will become second nature to you over time, as you'll use them continuously each day.