

O'REILLY®

Ten Steps to Linux Survival

Essentials for Navigating the Bash Jungle



James Lehmer

Ten Steps to Linux Survival

*Essentials for Navigating
the Bash Jungle*

James Lehmer

Beijing • Boston • Farnham • Sebastopol • Tokyo

O'REILLY®

Ten Steps to Linux Survival

by James Lehmer

Copyright © 2016 O'Reilly Media, Inc. All rights reserved.

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (<http://safaribooksonline.com>). For more information, contact our corporate/institutional sales department: 800-998-9938 or corporate@oreilly.com.

Editor: Dawn Schanafelt

Acquisitions Editor: Susan Conant

Production Editor: Shiny Kalapurakkal

Copyeditor: Sharon Wilkey

Proofreader: Molly Ives Brower

Interior Designer: David Futato

Cover Designer: Randy Comer

Illustrator: Rebecca Panzer

June 2016:

First Edition

Revision History for the First Edition

2016-05-27: First Release

The O'Reilly logo is a registered trademark of O'Reilly Media, Inc. *Ten Steps to Linux Survival*, the cover image, and related trade dress are trademarks of O'Reilly Media, Inc.

While the publisher and the author have used good faith efforts to ensure that the information and instructions contained in this work are accurate, the publisher and the author disclaim all responsibility for errors or omissions, including without limitation responsibility for damages resulting from the use of or reliance on this work. Use of the information and instructions contained in this work is at your own risk. If any code samples or other technology this work contains or describes is subject to open source licenses or the intellectual property rights of others, it is your responsibility to ensure that your use thereof complies with such licenses and/or rights.

978-1-491-95918-3

[LSI]

Table of Contents

| | |
|---|-----------|
| Introduction..... | v |
| 0. Step 0: Don't Panic..... | 1 |
| 1. Step 1: Getting In..... | 3 |
| “sudo make me a sandwich” | 5 |
| 2. Step 2: Getting Around..... | 7 |
| Where Am I? | 7 |
| Listing Files | 7 |
| Changing Directories | 9 |
| Be Lazy | 10 |
| 3. Step 3: Peeking at Files..... | 13 |
| Cool cat | 13 |
| less Is More | 14 |
| tail Wind | 15 |
| 4. Step 4: Finding Files..... | 17 |
| find Files Fast | 17 |
| Location, Location, Location | 20 |
| 5. Step 5: Search Me..... | 23 |
| Getting a grep | 23 |

Introduction

And you may ask yourself, “Well, how did I get here?”

—Talking Heads, “Once in a Lifetime”



This is an excerpt of Ten Steps to Linux Survival. For access to the complete report, please see <https://www.oreilly.com/learning/ten-steps-to-linux-survival>

Why Are We Here?

This report grew out of a series of “lunch-and-learns” on Linux that I compiled for work. During that process, I ended up **writing an ebook**, and then condensing it into a one-hour presentation that focuses on the essentials needed for quick problem-solving on a Linux system. I turned that presentation into **an O’Reilly webcast**, and this report provides more details on those original 10 essentials.

Even in formerly “pure Windows” shops, Linux use is growing. Linux systems are everywhere! They may appear as *appliances* (machines) or, more likely, virtual machine (VM) images dropped in by a vendor.

Common examples of Linux systems that may appear in your shop as VMs or in the cloud include the following:

Web servers

Apache, Nginx, Node.js

Database servers

MongoDB, PostgreSQL

Mobile device management

Various MDM solutions, such as MobileIron

Security and monitoring systems

Security information and event management (SIEM) systems, network sniffers

Source-code control systems

Git or Mercurial

As Linux use continues to grow, you need to know the basics. One day you might be the only one in the office when things go south, and you'll have to fix them—fast. This guide will help.

In this report, I focus on diagnosing problems and getting a system back up. I *don't* cover these topics:

- Modifying the system, other than restarting
- Forensics, other than looking at logs
- Shell scripting
- Distro differences—for example, Ubuntu versus CentOS
- Anything in depth, as this is just to get your feet wet

Who Is This For?

The intended audience of this book is *not* seasoned Linux administrators, or anyone with a passing knowledge of the Bash shell. Instead, it is for people who are working in small Windows shops, where everyone has to wear various hats. It is for Windows administrators, network admins, developers, and the like who have no knowledge of Linux but may still have to jump in during a problem. Imagine your boss rushing into your office and saying this:

The main www site is down, and all the people who know about it are out. It's running on some sort of Linux, I think, and the credentials and IP address are scrawled on this sticky note. Can you get in, poke around, and see if you can figure it out?

In this report, you'll learn the basic steps to finding vital information that can help you quickly get the site back up. By reading this guide before disaster strikes, you will be better able to survive the preceding scenario.

How to Prepare

In small shops, sometimes things just *fall on you* because no one else is available. There is often no room for “It’s not my job” when production is down and the one person who knows about it is back-packing in Colorado. So you need to be prepared as the use of Linux becomes more prevalent, turning “pure Microsoft” shops more and more into hybrids. Linux is coming, whether you like it or not. Be prepared.

First, pay *close attention* whenever you hear the word *appliance* used in terms of a system. Perhaps it will be mentioned in passing in a vendor presentation. Dig in and find out what the appliance image is running.

Second, note that *even Microsoft* is supporting Linux, and increasing that support daily. First, it started with making Linux systems first-class citizens on Azure. Now Microsoft is partnering with Docker and Ubuntu and others, and that coordination looks like it is only going to grow.

So now is the time to *start studying*. This report is a quick-help guide to prepare you for limited diagnostic and recovery tasks, and to get you used to how Linux commands work. But you should dig further.

One place to turn next is [my ebook](#). It helps you take the next steps of understanding how to change Linux systems in basic ways. I’ve also included some useful references at the end of this report. Past that, obviously, [O’Reilly has many good resources for learning Linux](#). And the Internet is just sitting there, waiting for you.

Play with It!

The best way to learn Linux is to stand up an environment where you can explore without fear of the consequences if you mess something up. One way is to create a Linux VM; even a moderately provisioned modern laptop will comfortably run a Linux VM. You can also create one in the cloud, and many vendors make that easy, including DigitalOcean, Linode, Amazon Elastic Compute Cloud (EC2), Microsoft Azure, and Google Compute Engine. Many of these even offer a free level, perfect for playing!

Documentation and Instrumentation

To protect yourself in case you are thrown into the scenario outlined at the beginning of this report, you should make sure the following are in place at your shop:

The Linux systems are documented.

This should include their purpose, as-built documentation outlining the distro, virtual or physical hardware specs, packages installed, and so on.

These systems are being actively monitored.

Are they tied in to Paessler Router Traffic Grapher (PRTG), SIEM, and other monitoring and alerting systems? Make sure you have access to those alerts and monitoring dashboards, as they can be a great source of troubleshooting information.

You have access to the system credentials.

Ideally, your department uses secure vault software to store and share system credentials. Do you have access to the appropriate credentials if needed? You should make sure before the need arises.

Conventions

If a command, filename, or other computer code is shown inline in a sentence, it appears in a fixed-width font:

```
ls --recursive *.txt
```

If a command and its output is shown on a terminal session, it appears as shown in [Figure P-1](#).

```
myuser@ubuntu-512mb-nyc3-01:~$ cat /etc/mtab
/dev/vda1 / ext4 rw,errors=remount-ro 0 0
proc /proc proc rw,noexec,nosuid,nodev 0 0
sysfs /sys sysfs rw,noexec,nosuid,nodev 0 0
none /sys/fs/cgroup tmpfs rw 0 0
none /sys/fs/fuse/connections fusectl rw 0 0
none /sys/kernel/debug debugfs rw 0 0
none /sys/kernel/security securityfs rw 0 0
udev /dev devtmpfs rw,mode=0755 0 0
devpts /dev/pts devpts rw,noexec,nosuid,gid=5,mode=0620 0 0
tmpfs /run tmpfs rw,noexec,nosuid,size=10%,mode=0755 0 0
none /run/lock tmpfs rw,noexec,nosuid,nodev,size=5242880 0 0
none /run/shm tmpfs rw,nosuid,nodev 0 0
none /run/user tmpfs rw,noexec,nosuid,nodev,size=104857600,mode=0755 0 0
none /sys/fs/pstore pstore rw 0 0
systemd /sys/fs/cgroup/systemd cgroup rw,noexec,nosuid,nodev,none,name=systemd 0
0
myuser@ubuntu-512mb-nyc3-01:~$
```

Figure P-1. *cat* command

All such blocks have been normalized to show a maximum of only 80 x 24 characters. This is intentional. Although most modern Linux systems and terminal windows such as *ssh* can handle any geometry, some systems and situations still give you the same terminal size that your grandfather would've used. It is best to learn how to deal with these by using *less*, redirection, and the like. In addition, screenshots are shown from a variety of systems, to get you used to the ways that command output and terminal settings can differ, *much* more than under the default Windows Command Prompt.

The examples in this book typically show something like `myuser@ubuntu-512mb-nyc3-01:~ $` before the command (as in the previous example). In other systems, you may simply see `~ #` (when logged in as root) or `%` (when running under *csh*). These command prompts are not meant to be typed in as part of the command. Although they may seem confusing in the samples, you need to get used to looking at a terminal and “parsing” what is being displayed. And in our scenarios, you won't have control over the command prompt format. Get used to it.

Typically, the screenshots are set up with the command entered at the prompt at the top of the screen, the command output immediately following, and in most cases a new command prompt waiting for another command at the end, as in the preceding example.

In the few places, where a Linux command is shown in comparison to a DOS command run under Windows Command Prompt, the

latter is shown in all uppercase to help distinguish it from the Linux equivalent, even though Windows Command Prompt is case-insensitive. In other words, `cd temp` is shown for bash, and `CD TEMP` for CMD.EXE.



This element signifies a tip or suggestion.



This element signifies a general note.



This element indicates a warning or caution.

Step 0: Don't Panic

The first, essential step is to stay calm. If you are dragged into trying to diagnose a Linux system and it isn't your area of expertise, you can only do so much. We're going to be careful to keep from changing system configurations, and we're going to restart services or the system only as a last resort.

So just try to relax, like Merv the dog (Figure 0-1). No one should expect miracles from you. And if you *do* figure out the problem, you'll be a hero!

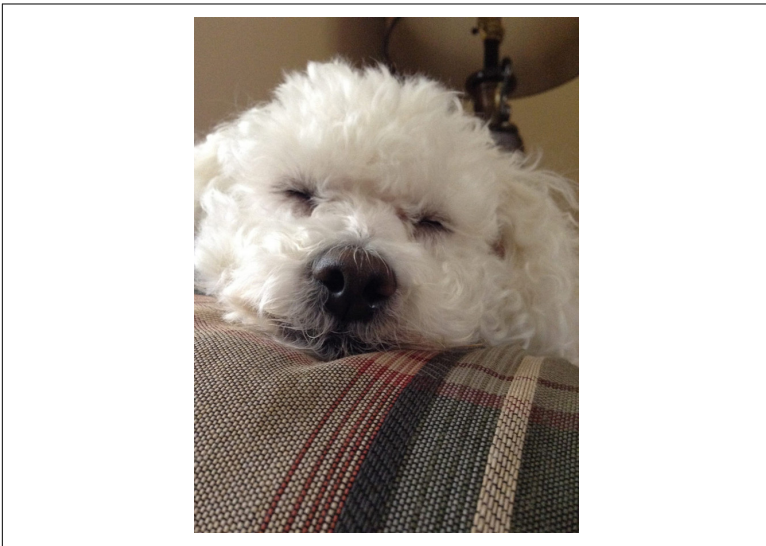


Figure 0-1. Merv the dog sez, Don't panic

Step 1: Getting In

Before I get too far, let's talk about how to connect to a Linux system in the first place. If you have an actual physical machine, you can use the console. In today's day and age, this isn't likely. If you are running VMs, you can use the VM software's console mechanism.

But most Linux systems run **OpenSSH**, a Secure Shell service, which creates an encrypted terminal connection via TCP/IP, typically to port 22. So, obviously, if you are connecting to an off-premise system, the appropriate firewall holes have to be in place on both sides. This allows you to connect from anywhere you want to work.

On Windows, you generally use **PuTTY** to establish SSH sessions with Linux systems. You typically need credentials as well, either from that sticky note your boss found, or preferably via your company's secure credentials management system.



You also could connect using public/private key pairs, but that is beyond the scope of this report.

When you start PuTTY, it looks like **Figure 1-1**.

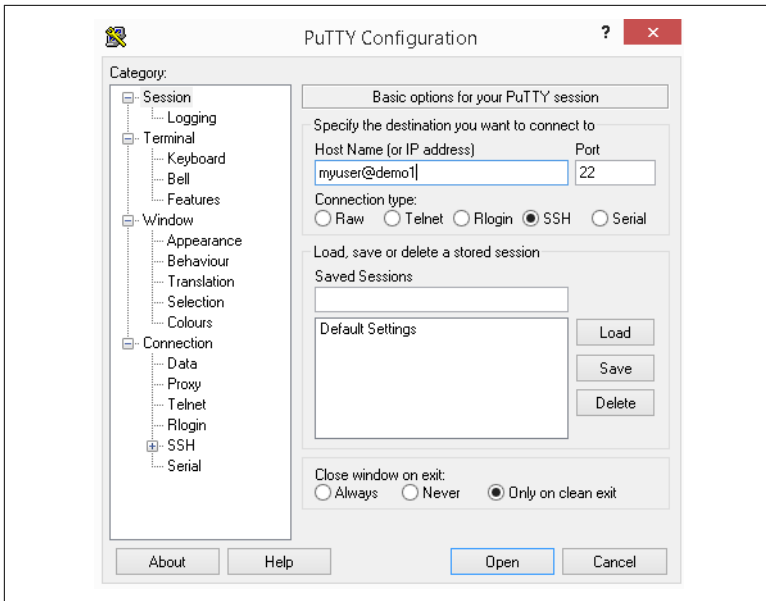


Figure 1-1. PuTTY prompt

You typically type in a user ID (in this example, **myuser**), followed by the at sign, **@**, and then the system's domain name or IP address (in this example, **demo1**).

When you click the Open button, if this is the first time you are connecting via SSH to a remote system, you will receive a warning similar to the one in [Figure 1-2](#).

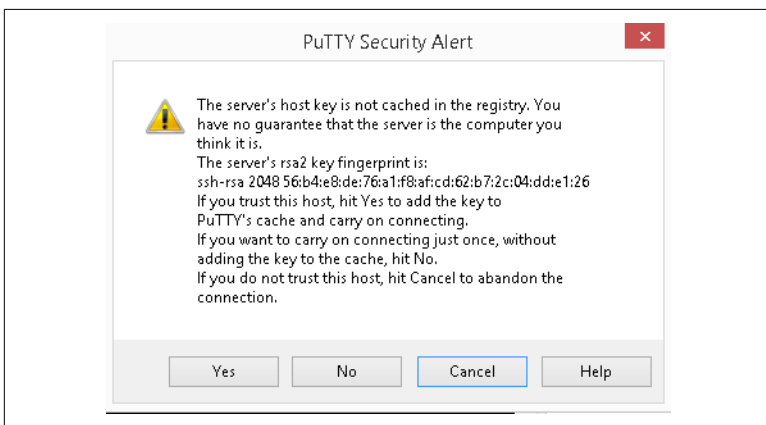


Figure 1-2. PuTTY alert

Simply click Yes, and the remote host’s key fingerprint will be stored so you don’t have to deal with this warning again. However, if you’ve already answered that prompt when connecting from your computer and you see it again *for the same remote system*, that means the remote machine’s IP address or other configuration has changed. That is often OK—changing the hosting provider for your public web server will trigger the warning for sure. However, if you know of no such changes, it may be indication of a system compromise, and you should abort the login and ask around.

You will then be presented with a password prompt, as shown in [Figure 1-3](#).

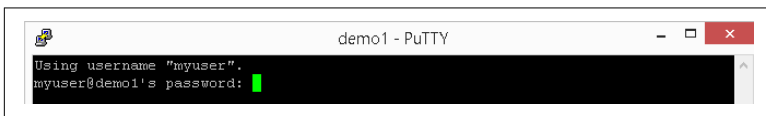


Figure 1-3. PuTTY password

Type in the password and hit Enter, and you should see something similar to [Figure 1-4](#).

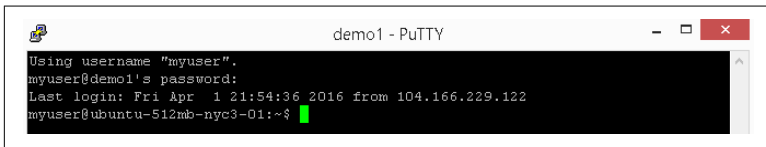


Figure 1-4. Successful login

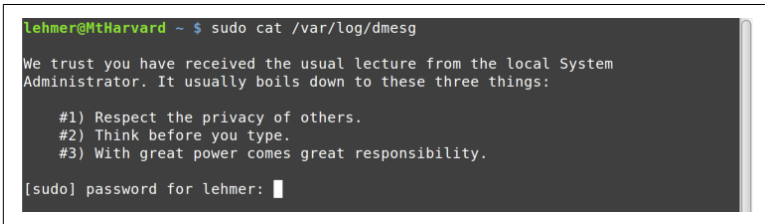
You’re in! Congratulations (or condolences, depending on how you feel about this assignment).

“sudo make me a sandwich”

I’m going to take a brief intermission to discuss the `sudo` command. It stands for *super-user do*. If a user is in the `sudo` user group, that user is allowed to execute privileged commands. It is similar to doing a `RUNAS` command in the Windows Command Prompt to run a command under an elevated account.

Logging in remotely as `root` (system administrator) is frowned upon, and in fact often forbidden for security purposes. Hence, you’ll need to use `sudo` to run admin commands that you will see later.

When you try to run a command and get an Access Denied message, you can then try it with `sudo`—for example, `sudo cat /var/log/dmesg`. The first time you run `sudo`, you will get the lecture shown in [Figure 1-5](#), which contains good words to live by anytime you are running as an administrator on any system!

A terminal window showing the output of the command 'sudo cat /var/log/dmesg'. The prompt is 'Lehmer@MtHarvard ~ \$'. The output is a lecture from the local System Administrator. It says: 'We trust you have received the usual lecture from the local System Administrator. It usually boils down to these three things: #1) Respect the privacy of others. #2) Think before you type. #3) With great power comes great responsibility.' Below this, it asks for the password: '[sudo] password for lehmer:'.

```
Lehmer@MtHarvard ~ $ sudo cat /var/log/dmesg
We trust you have received the usual lecture from the local System
Administrator. It usually boils down to these three things:

#1) Respect the privacy of others.
#2) Think before you type.
#3) With great power comes great responsibility.

[sudo] password for lehmer: █
```

Figure 1-5. sudo lecture

Note that you have to enter your password when you invoke `sudo`. Be clear, this is *your* user ID's password, not root's. This is to ensure that a human being is in control and that someone else isn't trying to hijack your terminal session while you're getting another cup of coffee.

Now that you know about `sudo`, you should get the punchline to [this comic](#), and hence the title of this section.

Step 2: Getting Around

Now that you're logged in, the first thing you'll want to do is inspect what is going on and how the system is configured. To do that, you need to list files and directories, and move around within the filesystem. This chapter covers these basics.

Where Am I?

Some command prompts are set to show the current directory path. Others are not, and it can be tough to remember where you are in the filesystem. The `pwd` (print working directory) command shows you:

```
bash-4.2$ pwd
/etc/init.d
```



Unlike in Windows, which is case-insensitive (but case-aware), in Bash and in Linux in general, *case matters*. By convention, most Linux commands are lowercase. If you try to type in an uppercase `PWD`, you will get a Command Not Found error.

Listing Files

In Bash, the `ls` (list) command is used to show directories and files. It is similar to the `DIR` command in Windows Command Prompt.

Figure 2-1 shows a simple sample of an `ls` command.

```
myuser@ubuntu-512mb-nyc3-01:~$ ls
CorporateSecrets.pdf  MyResume.docx  mysql.php  mysvc  Passwords.xlsx
myuser@ubuntu-512mb-nyc3-01:~$
```

Figure 2-1. `ls` command



Some ssh sessions use color highlighting, as shown in these screenshots (in this case, green means the file is executable). Some do not. So don't be surprised if you see colors!

To see a more detailed listing of the files and directories, you can use the `ls -l` command, as shown in [Figure 2-2](#).

```
myuser@ubuntu-512mb-nyc3-01:~$ ls -l
total 32
-rw-r--r-- 1 myuser myuser 9982 Apr 1 20:15 CorporateSecrets.pdf
-rw-r--r-- 1 myuser myuser 4027 Apr 1 20:15 MyResume.docx
-rw-r--r-- 1 myuser myuser 2627 Apr 1 20:15 mysql.php
-rwxrwx-- 1 myuser myuser 58 Apr 1 20:15 mysvc
-rw-r--r-- 1 myuser myuser 4723 Apr 1 20:15 Passwords.xlsx
myuser@ubuntu-512mb-nyc3-01:~$
```

Figure 2-2. `ls -l` command

From left to right, you see file permissions, owner, group, size, last modified date, and finally the file or directory name. File permissions are beyond the scope of this report, but if you continue your Linux education after reading this, you can learn more about them in my ebook.

In Windows, a file is hidden by setting a file attribute (metadata) on the file. In Linux, a file is hidden if its name starts with a period, or dot. To show these dot files, you use the `ls -a` command shown in [Figure 2-3](#).

```
myuser@ubuntu-512mb-nyc3-01:~$ ls -a
.      .bash_history  MyResume.docx  mysvc  .ssh
..     CorporateSecrets.pdf  mysql.php      Passwords.xlsx
myuser@ubuntu-512mb-nyc3-01:~$
```

Figure 2-3. `ls -a` command

On the left you see `.` and `..`, which mean *current directory* and *parent directory*, respectively, just as in Windows. You also see previously hidden files such as `.bash_history` and the `.ssh` directory (in this example, blue denotes a directory).

Finally, you can combine parameters. If you want to see a detailed listing (-l) of all files (-a), recursively descending into every child directory (-R), you simply combine them all (ls -aR), as shown in Figure 2-4.

```
myuser@ubuntu-512mb-nyc3-01:~$ ls -aR
.:
total 48
drwxr-xr-x 3 myuser myuser 4096 Apr  1 20:15 .
drwxr-xr-x 3 root   root   4096 Mar 27 11:58 ..
-rw----- 1 myuser myuser   93 Apr  1 20:17 .bash_history
-rw-r--r-- 1 myuser myuser 9982 Apr  1 20:15 CorporateSecrets.pdf
-rw-r--r-- 1 myuser myuser 4027 Apr  1 20:15 MyResume.docx
-rw-r--r-- 1 myuser myuser 2627 Apr  1 20:15 mysql.php
-rwxrwx--- 1 myuser myuser   58 Apr  1 20:15 mysvc
-rw-r--r-- 1 myuser myuser 4723 Apr  1 20:15 Passwords.xlsx
drwx----- 2 myuser myuser 4096 Apr  1 20:08 .ssh

./ssh:
total 12
drwx----- 2 myuser myuser 4096 Apr  1 20:08 .
drwxr-xr-x 3 myuser myuser 4096 Apr  1 20:15 ..
-rw----- 1 myuser myuser  395 Apr  1 20:08 authorized_keys
myuser@ubuntu-512mb-nyc3-01:~$
```

Figure 2-4. ls -aR command

Note the d in the far left column for ., .., and .ssh. This tells you they are directories, and in terminal sessions that do not use color highlighting, this d will be the only way you know which entries are files and which are directories.

Changing Directories

To change to a different directory, use the cd (change directory) command.



Linux uses the / character as the path delimiter, unlike Windows, which uses \. This will trip you up the first few times, especially because \ has a different meaning in Bash (it is an escape character).

Linux doesn't use drive letters. Instead, all devices are mounted in a single hierarchical namespace starting at the root (/) directory. You will see examples of this later in this report.

On login, you are usually in the *home directory*, which is represented by `~`. It is similar to the user directories under `C:\Users` on Windows. Hence, you will probably need to go elsewhere. Here's a list of common directories on Linux systems that are of interest:

`/etc`

System configuration files (often pronounced *slash-et-see* if someone is instructing you what to do over the phone)

`/var`

Installed software

`/var/log`

Log files

`/proc`

Real-time system information—similar to Windows Management Instrumentation (WMI), but easier!

`/tmp`

Temp files, cleared on reboots



Remember, case matters! And use `/`, not `\`!

Changing to another directory with `cd` is simple, as you can see in [Figure 2-5](#).

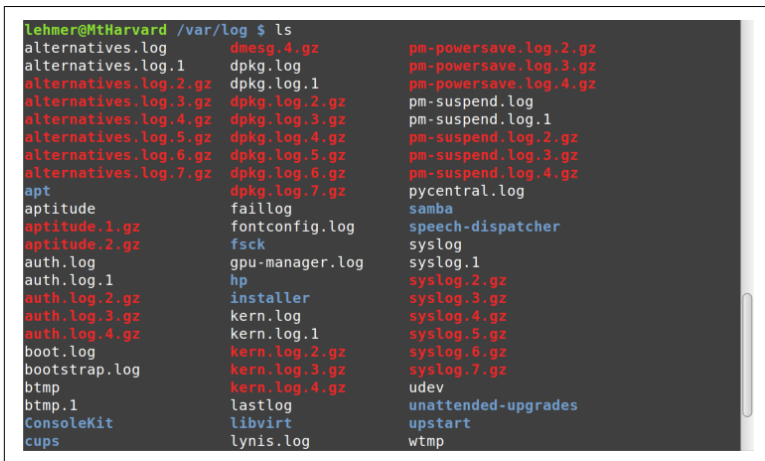
```
myuser@ubuntu-512mb-nyc3-01:~$ cd /etc
myuser@ubuntu-512mb-nyc3-01:/etc$ pwd
/etc
myuser@ubuntu-512mb-nyc3-01:/etc$
```

Figure 2-5. `cd /etc` command

Be Lazy

Most modern interactive shells like Bash and Windows Command Prompt allow for tab expansion and command history, at least for the current session of the shell. This is a good thing in a crisis situation, because it saves you typing, and thus, time.

Tab expansion is like autocomplete for the command prompt. Let's say you have some files in a directory, as shown in [Figure 2-6](#).



```
lehmer@MtHarvard /var/log $ ls
alternatives.log      dmesg.4.gz          pm-powersave.log.2.gz
alternatives.log.1   dpkg.log            pm-powersave.log.3.gz
alternatives.log.2.gz dpkg.log.1         pm-powersave.log.4.gz
alternatives.log.3.gz dpkg.log.2.gz      pm-suspend.log
alternatives.log.4.gz dpkg.log.3.gz      pm-suspend.log.1
alternatives.log.5.gz dpkg.log.4.gz      pm-suspend.log.2.gz
alternatives.log.6.gz dpkg.log.5.gz      pm-suspend.log.3.gz
alternatives.log.7.gz dpkg.log.6.gz      pm-suspend.log.4.gz
apt                  dpkg.log.7.gz      pycentral.log
aptitude            faillog             samba
aptitude.1.gz       fontconfig.log     speech-dispatcher
aptitude.2.gz       fsck                syslog
auth.log            gpu-manager.log    syslog.1
auth.log.1          hp                  syslog.2.gz
auth.log.2.gz       installer           syslog.3.gz
auth.log.3.gz       kern.log            syslog.4.gz
auth.log.4.gz       kern.log.1         syslog.5.gz
boot.log            kern.log.2.gz      syslog.6.gz
bootstrap.log       kern.log.3.gz      syslog.7.gz
btmp                kern.log.4.gz      udev
btmp.1              lastlog             unattended-upgrades
ConsoleKit          libvirt             upstart
cups                lynis.log           wtmp
```

Figure 2-6. `ls /var/log` command

Without tab expansion, typing out something like this is slow and error-prone:

```
cd unattended-upgrades
```

But with tab expansion, you can simply type `cd un[Tab]`, where [Tab] represents hitting the Tab key, and because only one directory starts with `un`, tab expansion will fill in the rest of the directory name for you.

One way that tab completion in Bash is different than in Windows Command Prompt is that in Bash, if you hit Tab and there are multiple candidates, Bash will expand as far as it can and then show you a list of files that match up to that point. You can then type in more characters and hit Tab again to complete it.

For example, in the previous example, if you wanted to list the details of the `pm-powersave.log.2.gz` file, instead of typing out `ls -l pm-powersave.log.2.gz` (27 keystrokes to type and possibly get wrong), you could use tab expansion to get it in two simple steps:

1. Type `ls -l pm-p[Tab]`. This would expand to `ls -l pm-powersave.log.`, because only the files named *pm-powersave.log* begin with *pm-p*. In this case, I specified just enough characters to distinguish between *pm-powersave.log* files and those beginning with *pm-suspend.log*.
2. Type `2[Tab]`. This would complete the rest, `.gz`, because only one *pm-powersave.log* file has a `2` in the next character location.

Thus, a total of 13 keystrokes, with two tab characters, saved typing 14 more!

Tab expansion is your friend, and you should use it as often as possible. It gives at least three benefits:

- Saves you typing.
- Helps eliminate misspellings in long file and directory names.
- Acts as an error checker—if the tab doesn't expand, chances are you are specifying the beginning part of the name wrong.

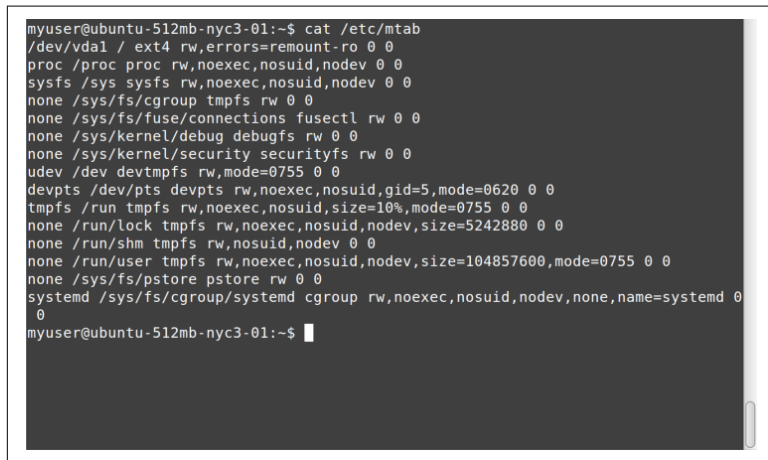
Another thing to remember about the interactive shell is command history. Both Windows Command Prompt and Bash give you command history, but Bash supports a rich interactive environment for searching for, editing, and saving command history. However, the biggest thing you need to remember in an emergency is simply that the up and down arrows work in the command prompt and bring back your recent commands so you can update them and re-execute them. This saves typing and reduces errors—use it!

Step 3: Peeking at Files

Now that you know how to move around in the filesystem, it is time to learn about how to inspect the content of files. In this chapter, I show a few commands that allow you to look inside files safely, without changing them.

Cool cat

The `cat` (concatenate) command dumps a file to the console, as shown in [Figure 3-1](#).

A terminal window showing the output of the `cat /etc/mtab` command. The output lists various filesystems and their mount options. The terminal prompt is `myuser@ubuntu-512mb-nyc3-01:~$`.

```
myuser@ubuntu-512mb-nyc3-01:~$ cat /etc/mtab
/dev/vda1 / ext4 rw,errors=remount-ro 0 0
proc /proc proc rw,noexec,nosuid,nodev 0 0
sysfs /sys sysfs rw,noexec,nosuid,nodev 0 0
none /sys/fs/cgroup tmpfs rw 0 0
none /sys/fs/fuse/connections fusectl rw 0 0
none /sys/kernel/debug debugfs rw 0 0
none /sys/kernel/security securityfs rw 0 0
udev /dev devtmpfs rw,mode=0755 0 0
devpts /dev/pts devpts rw,noexec,nosuid,gid=5,mode=0620 0 0
tmpfs /run tmpfs rw,noexec,nosuid,size=10%,mode=0755 0 0
none /run/lock tmpfs rw,noexec,nosuid,nodev,size=5242880 0 0
none /run/shm tmpfs rw,nosuid,nodev 0 0
none /run/user tmpfs rw,noexec,nosuid,nodev,size=104857600,mode=0755 0 0
none /sys/fs/pstore pstore rw 0 0
systemd /sys/fs/cgroup/systemd cgroup rw,noexec,nosuid,nodev,none,name=systemd 0
0
myuser@ubuntu-512mb-nyc3-01:~$
```

Figure 3-1. `cat` command

We will be using `cat` a lot in the rest of this report. Because most Linux configuration and log files are text, this command is handy for examining files, knowing that we can't change them by accident. The `CMD.EXE` equivalent is the `TYPE` command.

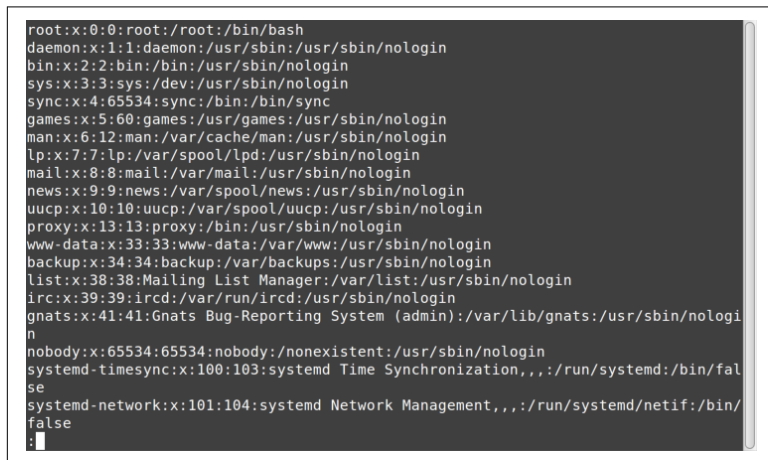
less Is More

The `less` command paginates files or output, with each “page” based on the size of the console window.

In Bash, as in Windows Command Prompt, the output from one command can be redirected, or piped, to another command by using the `|` character. In Linux, where each command “does one thing, well,” it is common practice to combine multiple commands, piping the output from one command to the next to accomplish a series of tasks in sequence. For example, later in this report you will see how to use the `ps` command to produce a list of running processes and then pipe that output to the `grep` command to search for a specific process by name. To demonstrate, although `less` can be passed a filename directly, here's how to pipe command output from `cat` to `less`:

```
~ $ cat /etc/passwd | less
```

The output from `less` clears the screen, and then shows the first page, as you can see in [Figure 3-2](#).



```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-timesync:x:100:103:systemd Time Synchronization,,,:/run/systemd:/bin/false
systemd-network:x:101:104:systemd Network Management,,,:/run/systemd/netif:/bin/false
:
```

Figure 3-2. less output

The colon at the bottom of the screen indicates that less is waiting for a command. After less displays its output, you have various navigation options:

- *Space, Page Down, or the down arrow* scrolls down.
- *Page Up or the up arrow* scrolls up.
- */* finds text searching forward (down) from the current cursor position, until the end of the file is reached; for example, */error*.
- *?* finds text searching backward (up) from the current cursor position, until the beginning of the file is reached; for example, *?error*.
- *n* finds next instance of the text you're searching for (note that the meaning of this is reversed when using *?*).
- *p* finds previous instance of the text you're searching for (note that the meaning of this is reversed when using *?*).
- *q* quits the less command and returns you to the prior view of the console.

tail Wind

The `tail` command shows the last lines in a file. It is useful when you're looking at large log files and want to see just the last lines—for example, right after an error has occurred. By default, `tail` will show the last 10 lines, but you can adjust the number of lines displayed with the `-n` parameter. For example, [Figure 3-3](#) shows how to display just the last five lines.

```
root@ubuntu-512mb-nyc3-01:/var/log/apache2# tail -n 5 access.log
54.186.16.79 - - [01/Apr/2016:18:54:52 -0400] "GET / HTTP/1.1" 200 543 "-" "Mozilla/5.0 (Windows NT 10.0; WOW64; rv:44.0) Gecko/20100101 Firefox/44.0"
54.186.16.79 - - [01/Apr/2016:18:54:57 -0400] "GET /CHANGELOG.txt HTTP/1.1" 404
470 "-" "Mozilla/5.0 (Windows NT 10.0; WOW64; rv:44.0) Gecko/20100101 Firefox/44.0"
54.186.16.79 - - [01/Apr/2016:18:55:02 -0400] "GET / HTTP/1.1" 200 543 "-" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/48.0.2564.103 Safari/537.36"
54.186.16.79 - - [01/Apr/2016:18:55:09 -0400] "GET /readme.html HTTP/1.1" 404 468 "-" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/48.0.2564.103 Safari/537.36"
185.56.82.99 - - [01/Apr/2016:21:24:55 -0400] "GET / HTTP/1.0" 200 609 "-" "masscan/1.0 (https://github.com/robertdavidgraham/masscan)"
root@ubuntu-512mb-nyc3-01:/var/log/apache2# █
```

Figure 3-3. `tail` command

The `tail` command can also “follow” a file, remaining running and showing new lines on the console as they are written to the file. This is useful when you’re watching a log file for a new instance of an error message, perhaps as you are testing to see if you can trigger the condition by visiting a web page on the site that is throwing an error. [Figure 3-4](#) shows an example using the `-f` parameter to follow a log file.

```
root@ubuntu-512mb-nyc3-01:/var/log/apache2# tail -n 5 -f access.log
54.186.16.79 - - [01/Apr/2016:18:54:52 -0400] "GET / HTTP/1.1" 200 543 "-" "Mozilla/5.0 (Windows NT 10.0; WOW64; rv:44.0) Gecko/20100101 Firefox/44.0"
54.186.16.79 - - [01/Apr/2016:18:54:57 -0400] "GET /CHANGELOG.txt HTTP/1.1" 404
470 "-" "Mozilla/5.0 (Windows NT 10.0; WOW64; rv:44.0) Gecko/20100101 Firefox/44.0"
54.186.16.79 - - [01/Apr/2016:18:55:02 -0400] "GET / HTTP/1.1" 200 543 "-" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/48.0.2564.103 Safari/537.36"
54.186.16.79 - - [01/Apr/2016:18:55:09 -0400] "GET /readme.html HTTP/1.1" 404 468 "-" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/48.0.2564.103 Safari/537.36"
185.56.82.99 - - [01/Apr/2016:21:24:55 -0400] "GET / HTTP/1.0" 200 609 "-" "masscan/1.0 (https://github.com/robertdavidgraham/masscan)"
```

Figure 3-4. tail -f command

Step 4: Finding Files

In the preceding chapter, you learned how to look inside files without changing them. But how do you know which files to look at? In this chapter, I cover searching for files, which can help narrow the scope for your troubleshooting.

find Files Fast

The `find` command is one of the most useful commands in Linux. The command works like this:

- Starting at location *x*
- Recursively find entries that *match* condition(s)
- *Do something* to each match

As a simple example, let's say you're in the `/var/log` directory, and you want to find all files that end in `.log`. Because there may be a lot of them, you will pipe the output to `less` so you can page through it. Here is the command:

```
/var/log# find . -name \*.log -print | less
```



Remember that I said the `\` has a different meaning in Bash, that it is an escape character? Notice its use in this example, where it is preventing the Bash shell from expanding the wildcard character (`*`) into all matching files in the current directory. Instead, by escaping it, the `\` character is telling `find` that wildcard in the current directory and all of its children.

Figure 4-1 shows the first page of the output I got from that command, awaiting our navigation via `less`.

```
./ufw.log
./apache2/other_vhosts_access.log
./apache2/access.log
./apache2/error.log
./boot.log
./mysql.log
./cloud-init-output.log
./dpkg.log
./unattended-upgrades/unattended-upgrades-shutdown.log
./upstart/network-interface-security-network-interface_eth0.log
./upstart/procps.log
./upstart/network-interface-eth0.log
./upstart/network-interface-lo.log
./upstart/systemd-logind.log
./upstart/network-interface-security-networking.log
./upstart/ureadahead.log
./upstart/network-interface-security-network-interface_lo.log
./alternatives.log
./auth.log
./cloud-init.log
./bootstrap.log
./apt/term.log
./apt/history.log
:
```

Figure 4-1. `find` results

The `find` command has a lot more power than this simple example! You can find files and directories based on creation and modification dates, file sizes, types, and much more. You can execute any variety of actions on each one as you find them, including Bash commands and shell scripts.

Figure 4-2 shows another example, where I am looking for all log files in `/var/log` and its child directories that were modified in the last hour, using the `-mmin` (modified *minutes*) parameter set to `-60` minutes. In this example no action parameter is given, so `-print` is implied.

```
myuser@ubuntu-512mb-nyc3-01:/var/log$ find . -mmin -60
./uFW.log
./lastlog
./btmpt
./upstart/systemd-logind.log
./wtmpt
./syslog
./auth.log
./kern.log
myuser@ubuntu-512mb-nyc3-01:/var/log$ █
```

Figure 4-2. *find -mmin*

You can also combine multiple search conditions and multiple actions. For example, if you want to find all log files in */var/log* that were modified in the last minute (*-mmin -1*), and then print its path (*-print*) and display the last two lines of each log file found (using *tail -n 2*), you use the following:

```
sudo find . -mmin -1 -print -exec tail -n 2 {} \;
```

I will pick that apart for you. From left to right:

sudo

Because some of the log files are protected unless you are root.

find

Search for some files.

•

Starting in the current directory (in this example, that's */var/log*).

-mmin -1

Find files that were modified in the last minute (*-1*).

-print

Print its full path.

-exec

For each file found, execute a command.

-tail -n 2

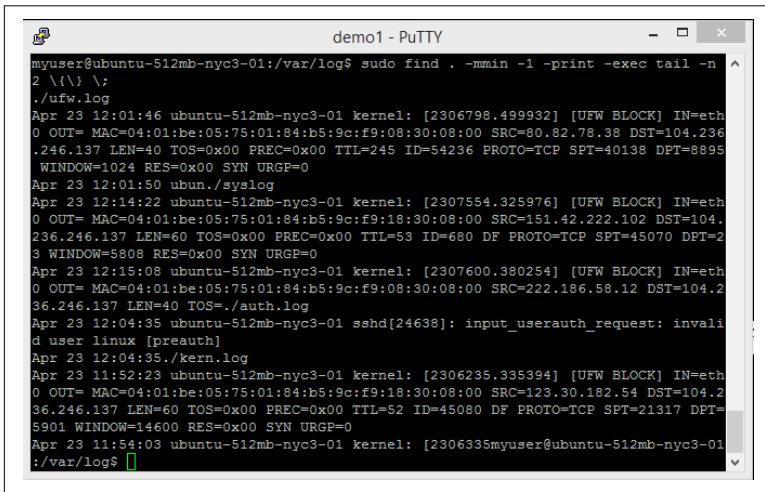
As you learned in the preceding chapter, *tail* shows you the final lines of a file; by default, it shows the last 10 lines, but here I have specified that it should show only the last 2 lines.

\{\} \;

Passing in the full path of the filename found to the *tail* command.

That last little bit of magic is important, and you will do well to memorize it for using `-exec` with the `find` command. The `\{\}` is the syntax for “pass in the path of the file that was found” (it is actually `{}`), but the `\` characters are escaping the brackets because they have special meaning to the Bash shell). The `;` is terminating the `-exec` parameter, so that other action parameters could follow on the `find` command. It is similarly escaped by `\` because the semicolon also has special meaning to Bash. The intervening space between `\{\}` and `;` is *required!*

Figure 4-3 shows it in action.



```
demo1 - PuTTY
myuser@ubuntu-512mb-nyc3-01:/var/log$ sudo find . -mmin -1 -print -exec tail -n 2 \{\} \;
./ufw.log
Apr 23 12:01:46 ubuntu-512mb-nyc3-01 kernel: [2306798.499932] [UFW BLOCK] IN=eth
0 OUT= MAC=04:01:be:05:75:01:84:b5:9c:f9:08:30:08:00 SRC=80.82.78.38 DST=104.236
.246.137 LEN=40 TOS=0x00 PREC=0x00 TTL=245 ID=54236 PROTO=TCP SPT=40138 DPT=8895
WINDOW=1024 RES=0x00 SYN URGP=0
Apr 23 12:01:50 ubun./syslog
Apr 23 12:14:22 ubuntu-512mb-nyc3-01 kernel: [2307554.325976] [UFW BLOCK] IN=eth
0 OUT= MAC=04:01:be:05:75:01:84:b5:9c:f9:18:30:08:00 SRC=151.42.222.102 DST=104.
236.246.137 LEN=60 TOS=0x00 PREC=0x00 TTL=53 ID=680 DF PROTO=TCP SPT=45070 DPT=2
3 WINDOW=5808 RES=0x00 SYN URGP=0
Apr 23 12:15:08 ubuntu-512mb-nyc3-01 kernel: [2307600.380254] [UFW BLOCK] IN=eth
0 OUT= MAC=04:01:be:05:75:01:84:b5:9c:f9:08:30:08:00 SRC=222.186.58.12 DST=104.2
36.246.137 LEN=40 TOS=0x00 PREC=0x00 TTL=52 ID=54236 PROTO=TCP SPT=40138 DPT=8895
WINDOW=1024 RES=0x00 SYN URGP=0
Apr 23 12:04:35 ubuntu-512mb-nyc3-01 sshd[24638]: input_userauth_request: invali
d user linux [preauth]
Apr 23 12:04:35 ./kernel.log
Apr 23 11:52:23 ubuntu-512mb-nyc3-01 kernel: [2306235.335394] [UFW BLOCK] IN=eth
0 OUT= MAC=04:01:be:05:75:01:84:b5:9c:f9:18:30:08:00 SRC=123.30.182.54 DST=104.2
36.246.137 LEN=60 TOS=0x00 PREC=0x00 TTL=52 ID=45080 DF PROTO=TCP SPT=21317 DPT=
5901 WINDOW=14600 RES=0x00 SYN URGP=0
Apr 23 11:54:03 ubuntu-512mb-nyc3-01 kernel: [2306335myuser@ubuntu-512mb-nyc3-01
:/var/log$
```

Figure 4-3. `find tail`



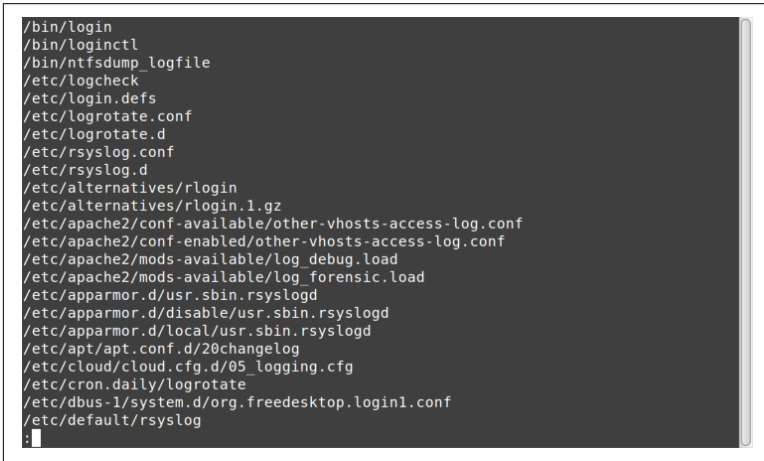
Because of the usefulness of the `find` command, I recommend you study it and play with it if you get a chance.

Location, Location, Location

The `locate` command searches a list of all the filenames on the system. The filenames are gathered periodically by a service, so it does not update in real time, but usually close enough. If you know the name of a file you are looking for, perhaps the Apache `access.log` file (which can change location depending on the Linux distro), you can use the `locate` command to quickly find it. Because `locate` searches

a pre-built list, it is much quicker for finding files by name than using `find -name`.

The `locate` command isn't "smart." It is simply looking for any file or directory with the string you pass it somewhere in the path. For example, if you execute `locate log | less` in the root (`/`) directory, you'll see something like [Figure 4-4](#).



```
/bin/login
/bin/loginctl
/bin/ntfsdump_logfile
/etc/logcheck
/etc/login.defs
/etc/logrotate.conf
/etc/logrotate.d
/etc/rsyslog.conf
/etc/rsyslog.d
/etc/alternatives/rlogin
/etc/alternatives/rlogin.1.gz
/etc/apache2/conf-available/other-vhosts-access-log.conf
/etc/apache2/conf-enabled/other-vhosts-access-log.conf
/etc/apache2/mods-available/log_debug.load
/etc/apache2/mods-available/log_forensic.load
/etc/apparmor.d/usr.sbin.rsyslogd
/etc/apparmor.d/disable/usr.sbin.rsyslogd
/etc/apparmor.d/local/usr.sbin.rsyslogd
/etc/apt/apt.conf.d/20changelog
/etc/cloud/cloud.cfg.d/05_logging.cfg
/etc/cron.daily/logrotate
/etc/dbus-1/system.d/org.freedesktop.login1.conf
/etc/default/rsyslog
:
```

Figure 4-4. locate results

Note that `log` appears somewhere in each path, but doesn't necessarily lead to `log files`.

Step 5: Search Me

In the preceding chapter, you learned to search for files by their attributes, such as name, last modified time, and the like. In this chapter, I show how to search *inside* a file, perhaps to find a specific error message.

Getting a grep

The `grep` command (whose name comes from globally search a regular expression and *print*) searches within files. It uses regular expressions (regex) to match patterns inside the files. It can be used to search within binary files, but is most useful for finding things inside text files. There are lots of uses for this command in our crisis scenario, such as searching for certain error messages within log files, or finding every mention of a certain resource inside the source files for an entire website.

There is an old joke by Jamie Zawinski:

Some people, when confronted with a problem, think, “I know, I’ll use regular expressions.” Now they have two problems.

Some regular expressions are simple—for example, `*`, which you should recognize as a valid wildcard in Windows Command Prompt. Others can be mind-blowingly complex. For example:

```
^\(*\d{3}\)*(-)*\d{3}(-)*\d{4}$
```

This regular expression is an (incomplete) approach to matching US phone numbers.

Because regexes are so inscrutable, sometimes I write a regex in a program or a script, come back to it six months later, and have no idea what it is doing. (Now I have two problems.) In this chapter, you're just going to look at a few simple examples.

Here are some samples of using regular expressions with `grep`. You will look at the output of some of them in the following screenshots.

```
grep 500 access.log
```

Find any occurrence of *500* in *access.log*

```
grep '\s500\s' access.log
```

Find *500* surrounded by whitespace (space, tab)

```
grep '^159.203' access.log
```

Find *159.203* at *beginning* of lines (^)

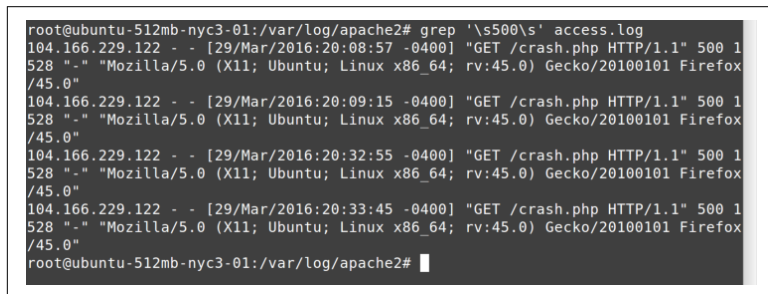
```
grep 'bash$' /etc/passwd
```

Find *bash* at *end* of lines (\$)

```
grep -i -r error /var/log
```

Find all case-insensitive (-i) instances of *error* in the */var/log* directory and its children (-r)

For that first example, you know that if a web program throws a server-side error, by convention it will send an HTTP status code of 500 to the client (browser). Most web servers also write that to their logs. So let's look for 500 in Apache's web log, as shown in [Figure 5-1](#).



```
root@ubuntu-512mb-nyc3-01:/var/log/apache2# grep '\s500\s' access.log
104.166.229.122 - - [29/Mar/2016:20:08:57 -0400] "GET /crash.php HTTP/1.1" 500 1
528 "-" "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:45.0) Gecko/20100101 Firefox
/45.0"
104.166.229.122 - - [29/Mar/2016:20:09:15 -0400] "GET /crash.php HTTP/1.1" 500 1
528 "-" "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:45.0) Gecko/20100101 Firefox
/45.0"
104.166.229.122 - - [29/Mar/2016:20:32:55 -0400] "GET /crash.php HTTP/1.1" 500 1
528 "-" "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:45.0) Gecko/20100101 Firefox
/45.0"
104.166.229.122 - - [29/Mar/2016:20:33:45 -0400] "GET /crash.php HTTP/1.1" 500 1
528 "-" "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:45.0) Gecko/20100101 Firefox
/45.0"
root@ubuntu-512mb-nyc3-01:/var/log/apache2#
```

Figure 5-1. grep command

I use the `'\s500\s'` regular expression in this command to make sure that only instances of 500 surrounded by spaces (or tabs) are found. Web logs tend to put the HTTP status code in its own col-

umn, and I don't want to see extraneous 500s that are part of response sizes, time-zone offsets, or whatnot.

Perhaps you're being attacked by a block of IP addresses, maybe a bunch of botnets running on some cable modems. The IP block attacking you is 159.203, so let's find all log lines that start with that client address, as shown in [Figure 5-2](#).

```
root@ubuntu-512mb-nyc3-01:/var/log/apache2# grep '^159.203' access.log
159.203.76.169 - - [30/Mar/2016:18:57:57 -0400] "GET /muieblackcat HTTP/1.1" 404
469 "-" "-"
159.203.76.169 - - [30/Mar/2016:18:57:57 -0400] "GET //phpMyAdmin/scripts/setup.
php HTTP/1.1" 404 485 "-" "-"
159.203.76.169 - - [30/Mar/2016:18:57:57 -0400] "GET //phpmyadmin/scripts/setup.
php HTTP/1.1" 404 485 "-" "-"
159.203.76.169 - - [30/Mar/2016:18:57:57 -0400] "GET //pma/scripts/setup.php HTT
P/1.1" 404 478 "-" "-"
159.203.76.169 - - [30/Mar/2016:18:57:57 -0400] "GET //myadmin/scripts/setup.php
HTTP/1.1" 404 482 "-" "-"
159.203.76.169 - - [30/Mar/2016:18:57:57 -0400] "GET //MyAdmin/scripts/setup.php
HTTP/1.1" 404 482 "-" "-"
root@ubuntu-512mb-nyc3-01:/var/log/apache2#
```

Figure 5-2. grep 159.203 command

In this case, note that the regular expression starts with `^`, which means to look for the following pattern only at the beginning of each line in the log file.

Similarly, you can look for patterns at the end of each line as well. The `/etc/passwd` file holds every user ID on a Linux system. (Don't worry, it no longer holds the password, but once upon a time, it did!) Each user is defined by a line in the file, and the last entry on each line indicates the "shell" in which they run. Some user IDs are defined to not be allowed to have interactive logins, and so they might have something like `/bin/false` or `/usr/sbin/nologin` as their shell.

But user IDs that can log in will have `bash` or `csh` or similar. So if you want to find all user IDs that can log in interactively, you could use the command in [Figure 5-3](#), which looks for `bash` at the end of the line by specifying the `$` in the regular expression.

```
root@ubuntu-512mb-nyc3-01:~# grep 'bash$' /etc/passwd
root:x:0:0:root:/root:/bin/bash
myuser:x:1000:1000:My User,,,:/home/myuser:/bin/bash
root@ubuntu-512mb-nyc3-01:~#
```

Figure 5-3. grep bash command

You then see that `root` and `myuser` are the only IDs allowed an interactive login on this system.

Finally, because you're trying to find out what is wrong with the Linux system you've been thrown into, perhaps you want to see each instance of the word *exception* in the log files. You could do that with something like this:

```
grep -i -r 'exception' /var/log | less
```

Here's what each part of that command does:

`grep`

Searches through files

`-i`

Ignores case (makes the search string case-insensitive)

`-r`

Recursively searches through all directories

`'exception'`

Looks for the string *exception*

`/var/log`

Starts in the */var/log* directory

`| less`

Pipes the output through `less` so you can look at it one "page" at a time

Figure 5-4 shows the first page of the output.

```
/var/log/auth.log:Mar 27 15:56:12 ubuntu-512mb-nyc3-01 sshd[1927]: error: Received disconnect from 162.255.86.31: 3: com.jcraft.jsch.JSchException: Auth fail [preauth]
/var/log/auth.log:Mar 27 22:23:53 ubuntu-512mb-nyc3-01 sshd[1650]: error: Received disconnect from 162.255.86.31: 3: com.jcraft.jsch.JSchException: Auth fail [preauth]
/var/log/auth.log:Mar 27 23:15:31 ubuntu-512mb-nyc3-01 sshd[1694]: error: Received disconnect from 195.154.52.9: 3: com.jcraft.jsch.JSchException: Auth fail [preauth]
/var/log/auth.log:Mar 28 03:09:29 ubuntu-512mb-nyc3-01 sshd[1939]: error: Received disconnect from 162.255.86.31: 3: com.jcraft.jsch.JSchException: Auth fail [preauth]
/var/log/auth.log:Mar 28 09:59:29 ubuntu-512mb-nyc3-01 sshd[2971]: error: Received disconnect from 162.255.86.31: 3: com.jcraft.jsch.JSchException: Auth fail [preauth]
/var/log/auth.log:Mar 28 10:03:25 ubuntu-512mb-nyc3-01 sshd[2992]: error: Received disconnect from 125.212.232.94: 3: com.jcraft.jsch.JSchException: Auth fail [preauth]
/var/log/auth.log:Apr 1 03:11:00 ubuntu-512mb-nyc3-01 sshd[12787]: error: Received disconnect from 42.114.202.229: 3: com.jcraft.jsch.JSchException: Auth fail [preauth]
/var/log/auth.log:Apr 1 03:11:12 ubuntu-512mb-nyc3-01 sshd[12789]: error: Received disconnect from 42.114.202.229: 3: com.jcraft.jsch.JSchException: Auth fail [preauth]
```

Figure 5-4. *grep* exception results

In this case, you see a bunch of authorization failures in the first page of output from the `/var/auth` log. If the problem you are chasing includes an authentication error, perhaps on your website, this would show a good path to keep continuing down. Many times you have to change your search phrases multiple times and use your “tech intuition” to decide which errors are worth following further. Troubleshooting is often more of an art than a science, so “Use the Force, Luke.”

About the Author

Jim Lehmer has been “in computers” for over three decades. He has held various software development roles, including programmer, systems programmer, software engineer, team lead, and architect.

Besides bragging about his wife, Leslie, his five children, and four grandchildren, his hobbies include reading, writing, running, hiking, and climbing.

Acknowledgments

Thanks to my coworkers, who inspired and attended the lunch-and-learn sessions from which my ebook, webcast, and this report grew—especially Aaron Vandegriff and Rob Harvey. I received excellent advice and promotion from Professor Allen Downey, for which I am grateful. I am thankful to my editor at O’Reilly, Dawn Schanafelt, with her eye for detail and helpful suggestions. Finally, I owe more than I can repay (as usual) to my wife, Leslie, who deserves shared credit for putting up with me during the nights and weekends I obsessed over this project.