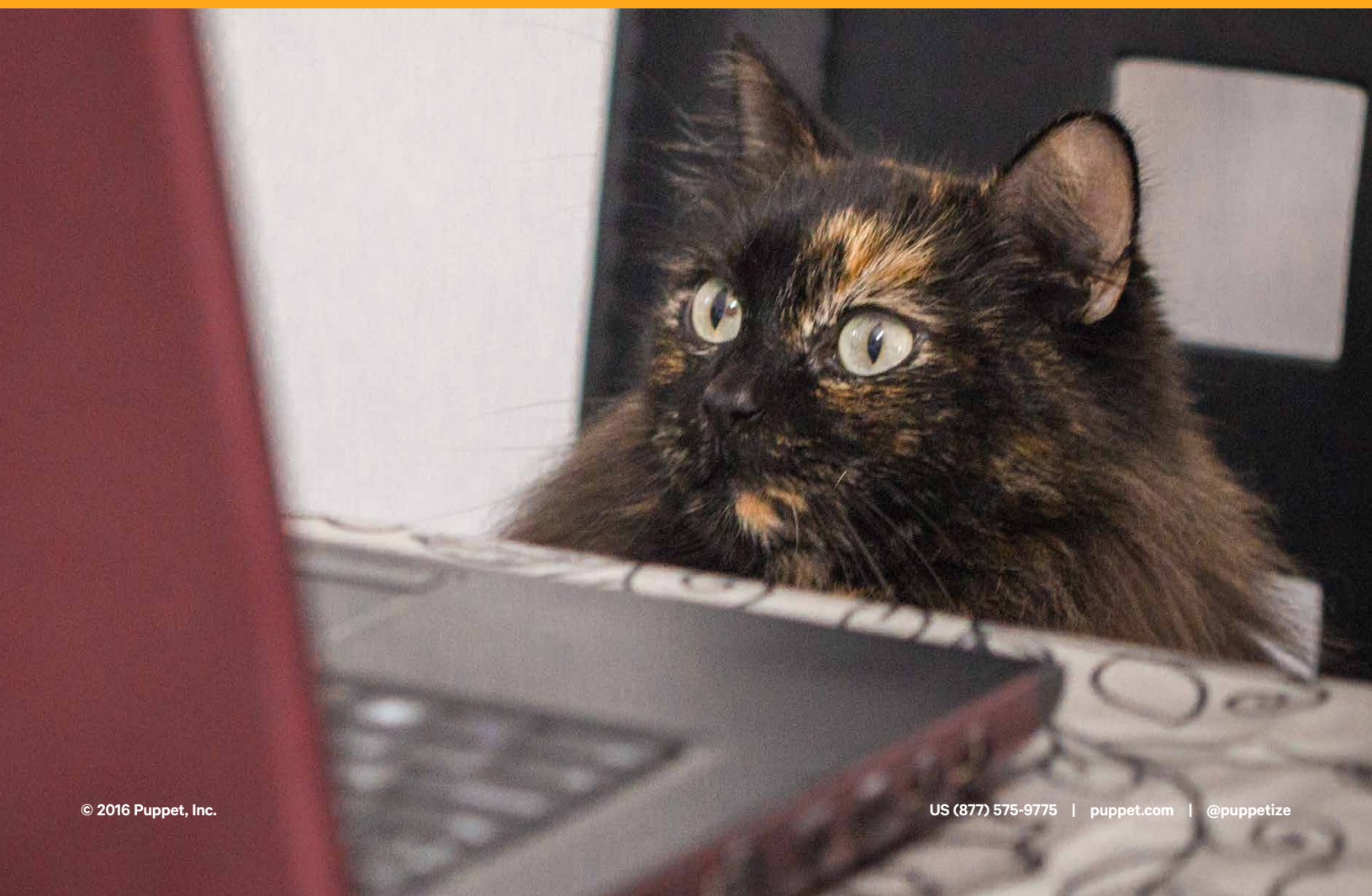


The Tools for Learning Puppet

A guide to getting started with command line, Vim and Git





Introduction

So you want to learn how to use Puppet?

Perfect! This ebook gives you descriptions of tools that Puppet users work with on a routine basis, and that are used in **Puppet Fundamentals** courses. We'll take you through a tutorial to help you learn each of these tools: command line interface (CLI), Vim and Git.

All of these are fundamental tools for system administrators, software developers and others working in IT and related fields. They're powerful tools that give you a great deal of flexibility and a strong basis for learning more.

Cover image:
Special thanks to **Iker Cortabarría** for original image on Flickr.



Thanks to [Anna Hanks](#) for original image on Flickr.

Before you dive in, it's good to know you can complete a Puppet Fundamentals course without working extensively with Vim and the CLI — if you prefer, you can get away with doing little more on the command line beyond typing `puppet agent --test`.

However, most people who want to learn Puppet have found that gaining a bit of knowledge in CLI, Vim and Git makes them feel more comfortable and confident. And once you've taken your first dive into using Puppet, you'll begin to see benefits that other system administrators enjoy.

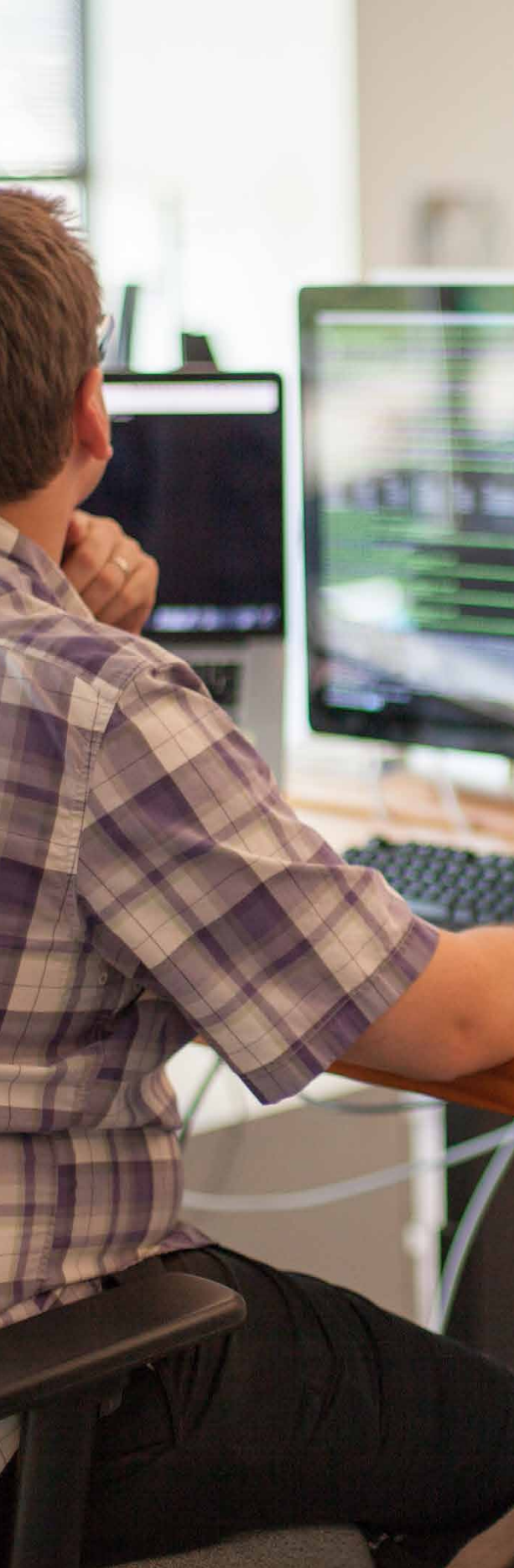
Getting your hands around any new tools can make you want to flip tables at first, especially when the instructions assume you know something you don't — for example, if the guide uses the word **directory** to refer to something you think of as a **folder**.

People who want to learn Puppet come from all kinds of backgrounds, so this ebook is designed to help you get familiar with fundamental tools commonly used in system administration and other technical fields. Once you get a handle on things like interacting through a command line interface (CLI), along with other tools like Vim and Git, you'll feel a lot more confident going through a Puppet Fundamentals course.

We like to imagine that you'll be learning all this with your trusty cat by your side (and sometimes all over your keyboard), so some of our examples in this guide have a certain *catty* quality.

The Linux/Unix-y basics needed for a Puppet Fundamentals class are a command line interface (CLI), a text editor and a version control system.

The first section of this guide includes a quick overview of what the tools do and the steps needed to install what is basically a tiny computer on your computer.



The tools for using Puppet: an overview

Command line interface (CLI)/terminal



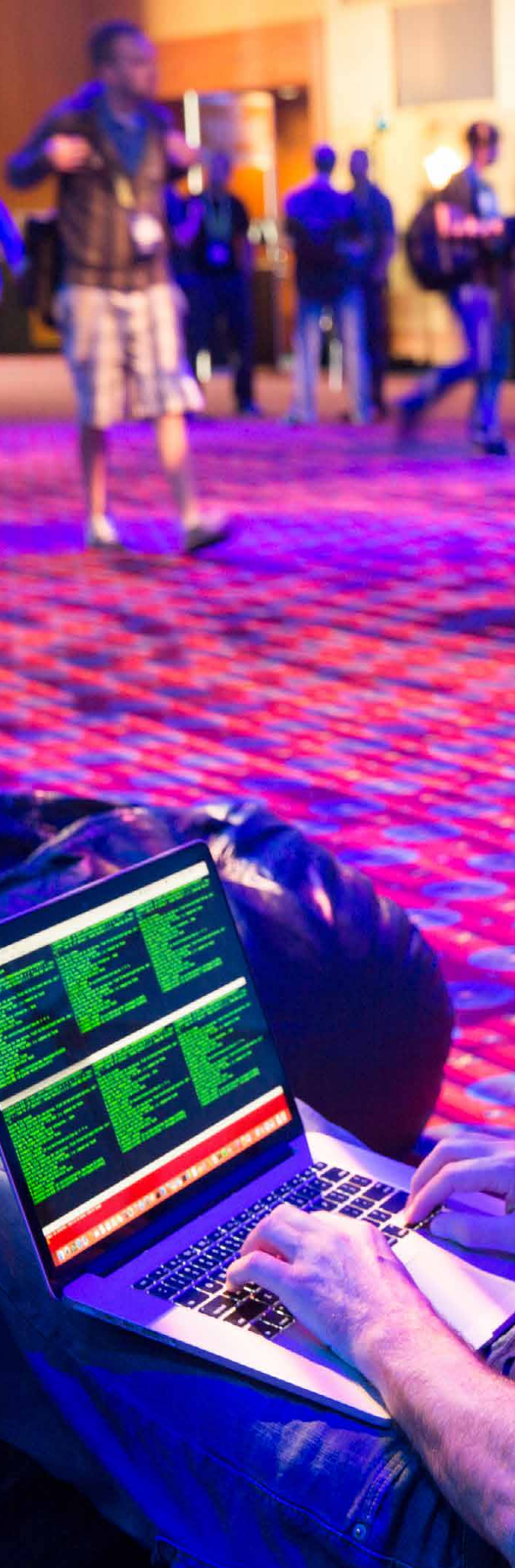
If you come from a Windows or Macintosh background, you are used to opening files by going to your desktop, selecting a folder, selecting any sub-folders if necessary and then clicking to open the file so you can get work done.

Back before folks figured out how to create the now common mouse-icon interface, called a graphical user interface (GUI), you had to talk to your computer directly with the keyboard. Command line interface (CLI), commonly referred to as **command line**, is the language used to communicate with the computer through the keyboard. Think of the command line terminal as a chat room/instant message conversation between you and your computer.

A programmer writes a program in command line and saves it in a file. Then they use the terminal to run the new file while watching the terminal to make sure the program runs as intended. Operations folks use command line to explore inside the computer to make sure everything is working according to plan.

Vim/text editor

You are going to need a text editor for any computer you plan to use for programming. Don't use Microsoft Word, or similar programs used to type up documents, because these programs actually put invisible code around your words to format text and make it look pretty. The invisible code will get jumbled with the code you are writing, and you'll have a bad time.



Technically you could use Notepad, but we are going to work with the popular open source text editor Vim. Vim looks weird at first, but it is a pro-level tool that will save you lots of time in the long run, so it's worth learning. You can use Vim to edit Puppet code in your Puppet Fundamentals course (though it's worth noting that students in this course who don't know how to use Vim can edit code without it).

Instead of a GUI (clicking icon to open menu and then selecting), Vim uses a text interface. This means you will save a lot of time by not moving and clicking the mouse constantly, but you will have to memorize (or make a cheatsheet of) keyboard shortcuts to make things happen.

Later, we will download a virtual machine tool with Vim already installed, but if you want to add Vim to your personal computer, go [here](#) for installation instructions.

Git/versioning control

We want to be able to share our code, allowing it to be used on any computer, because you never know when your laptop may mysteriously combust. To do this, we are going to use Git. Git keeps track of all changes you make to your files. You can create several different copies of the original file (the copies are called **branches**), make different changes to each of the branches, and then seamlessly merge everything back together into the original file (called a **repository** or **repo**).

GitHub is the website that runs Git, and is also the place where you can write or upload code right in the browser, and then choose to allow others to use and contribute to that code. If someone wants to make a change that will affect your original, they make what is called a **pull request**. You then review their changes and decide whether to merge their code into your repository or not. You can create your own GitHub account and start using Git at github.com.

Get the tools ready

Now that we know what our tools are, let's get ready to play. You will need a computer that uses a terminal and another that you can follow along with. Since we aren't sitting together in the same room, you will need to download a **virtual machine** (VM).

A VM is basically an entirely separate computer that lives in a small section of your computer. You could install a Windows VM on your Mac, or a Mac VM on your Unix box.

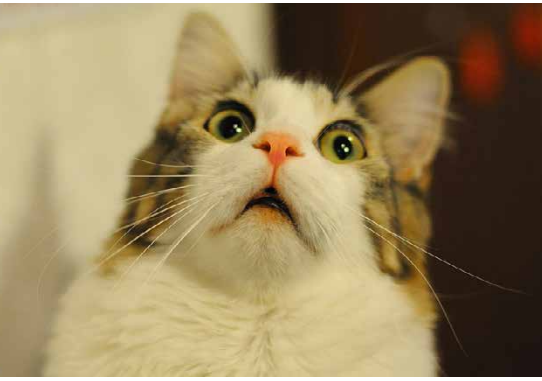
VMware provides software that allows a computer to host a VM. You can download VMware Fusion for OS X [here](#), and VMware Workstation for Windows and Linux [here](#).

VirtualBox is another popular virtualization product. Go to virtualbox.org/wiki/downloads to get the latest version for your operating system (Windows, OS X, Linux or Solaris). Open it and install when prompted.

Finished? Great! Your computer can now host VMs, so let's add one! We've got some VMs ready to roll [here](#). Be sure to check the requirements to make sure your computer will be compatible, then fill out the quick registration form and start your download. Unfortunately, downloading the VM will take a while, which is why we recommend using a wired internet connection (not Wi-Fi). Once it's finished downloading unzip the file. If you need help unzipping a file, check out [this page](#). Follow the setup instructions to get the VM up and running, but instead of continuing on with the Quest Guide that comes with the VM, return here and we'll get you started learning the CLI, Vim, and Git. (Once you're through here, of course, you may forge ahead with the rest of the Learning VM!)



OMG! WAT?!



Okay, maybe we should have warned you what is going on — particularly regarding the mouse and keyboard capture — before just diving right in there. A lot of things are happening at once, and some can be alarming.

Just let them happen and read on.



That black window that appeared out of nowhere? That's the virtual machine. All the white text in the main screen is the VM booting up. The semi-transparent text windows that are popping up are not happening in the VM, but are notifications from your computer about what is going on. You can click to expand and read these notifications, close them by clicking **X**, or click the **no speech bubbles icon** so you never see these warnings again.

When you click inside the black box that appeared, it captures your mouse — kinda like sucking it through a wormhole in your computer. You won't be able to get your mouse back or mess with your computer until you send the mouse back through the wormhole. There is a notification the first time you try to click, and an escape command (which is not the same thing as the escape key on your keyboard) will prompt you to send your mouse back. Don't worry about remembering this — you'll see there is a reminder note in the bottom right corner of the VM window in case you forget. If you're a Mac user, when VirtualBox refers to Left Command, that means you should use the Command key on the left side of the keyboard (**not** the command key plus the left arrow key).

Once the loading bar is finished, you're ready to log in. Type `root` as your username and use the password supplied on the splash screen. Congratulations, you're in!



OK, I'm in — but in where?

Image courtesy of [Marnee Pearce](#) on Flickr.
Licensed via Creative Commons.

You may be thinking, “OK, I’m in...but in where?” Once you are in, you will start in the biggest folder in what is called the **home directory**. Since you logged in as an administrator — that’s what **root** means — you can now check out any of the files or folders inside the home directory. Also, you can see all of the user logins for your computer and make changes to other users’ files. If you had logged in as a non-root/administrative user, you probably wouldn’t have those magnificent and terrifying abilities to change other users’ files.

You will now see: `root@learning:~#`

This is a prompt, and it indicates that you are logged in as `root@learning` and that you are currently in the home directory. The tilde sign `~` is shorthand for the home directory. When you see the hash symbol `#`, that means the computer is waiting for you to tell it what to do. In other terminals, you may see `>` or `$` instead of `#`.

The next steps are telling the computer to do something, learning how to maneuver and how to wreak havoc using the command line. All of this is covered in the next section — see you there!



Learning Puppet with CLI

Now that you understand the basic tools needed to run Puppet and have successfully downloaded the **Puppet Learning VM**, we are going to use the keyboard and command line to communicate with the computer, instead of using the mouse to click from place to place.

YOU'RE NO HELP AT ALL.

Image courtesy of **Marnee Pearce** on Flickr.
Licensed via Creative Commons.

Let's try command line

FossWire has a useful **cheat sheet** of commonly used command line interface (CLI) commands that you may want to print for easy reference.

First we'll try a few of the more straightforward commands.

(Note: The commands in this section are for Unix/OS X, so if you want to use CLI on Windows, there will be minor differences.)

Simply type the sample commands below that look like **this** exactly as shown and press Enter after each one. The terminal should display the right kind of information to the right of the command. **If it doesn't, ask your local engineer.**

What? You don't have one, or they don't know? Check out **StackOverflow**, a site where programmers from beginners to experts ask and answer questions. If you can't find the answer you need, just open up a new question.

Unix/Linux Command Reference		FossWire.com
File Commands		
ls - directory listing		
ls -al - formatted listing with hidden files		
cd dir - change directory to dir		
cd - change to home		
pwd - show current directory		
mkdir dir - create a directory dir		
rm file - delete file		
rm -r dir - delete directory dir		
rm -f file - force remove file		
rm -rf dir - force remove directory dir *		
cp file1 file2 - copy file1 to file2		
cp -r dir1 dir2 - copy dir1 to dir2; create dir2 if it doesn't exist		
mv file1 file2 - rename or move file1 to file2		
if file1 is an existing directory, moves file2 into directory file2		
ln -s file link - create symbolic link link to file		
touch file - create or update file		
cat > file - places standard input into file		
more file - output the contents of file		
head file - output the first 10 lines of file		
tail file - output the last 10 lines of file		
tail -f file - output the contents of file as it grows, starting with the last 10 lines		
Process Management		
ps - display your currently active processes		
top - display all running processes		
kill pid - kill process id pid		
killall proc - kill all processes named proc *		
bg - lists stopped or background jobs; resume a stopped job in the background		
fg - brings the most recent job to foreground		
File Permissions		
chmod octal file - change the permissions of file to octal, which can be found separately for user, group, and world by adding:		
● 4 - read (r)		
● 2 - write (w)		
● 1 - execute (x)		
Examples:		
chmod 777 - read, write, execute for all		
chmod 755 - rwx for owner, rx for group and world		
For more options, see man chmod		
SSH		
ssh user@host - connect to host as user		
ssh -p port user@host - connect to host on port port as user		
ssh-copy-id user@host - add your key to host for user to enable a keyless or passwordless login		
Searching		
grep pattern files - search for pattern in files		
grep -r pattern dir - search recursively for pattern in dir		
command grep pattern - search for pattern in the output of command		
locate file - find all instances of file		
System Info		
date - show the current date and time		
cal - show this month's calendar		
uptime - show current uptime		
w - display who's online		
whoami - who you are logged in as		
finger user - display information about user		
uname -a - show kernel information		
cat /proc/cpuinfo - cpu information		
cat /proc/meminfo - memory information		
man command - show the manual for command		
df - show disk usage		
du - show directory space usage		
free - show memory and swap usage		
whereis app - show possible locations of app		
which app - show which app will be run by default		
Compression		
tar cf file.tar files - create a tar named file.tar containing files		
tar xf file.tar - extract the files from file.tar		
tar czf file.tar.gz - create a tar with gzip compression		
tar xzf file.tar.gz - extract a tar using gzip		
tar xjf file.tar.bz2 - create a tar with bzip2 compression		
tar xjf file.tar.bz2 - extract a tar using bzip2		
gzip file - compresses file and renames it to file.gz		
gunzip -d file.gz - decompresses file.gz back to file		
Network		
ping host - ping host and output results		
whois domain - get whois information for domain		
dig domain - get DNS information for domain		
dig -x host - reverse lookup host		
wget file - download file		
wget -c file - continue a stopped download		
Installation		
Install from source:		
./configure		
make		
make install		
dpkg -i pkg.deb - install a package (Debian)		
rpm -ivh pkg.rpm - install a package (RPM)		
Shortcuts		
Ctrl+C - kills the current command		
Ctrl+Z - stops the current command, resume with fg in the foreground or bg in the background		
Ctrl+D - log out of current session, similar to exit		
Ctrl+W - erases one word in the current line		
Ctrl+U - erases the whole line		
!! - repeats the last command		
exit - log out of current session		
* use with extreme caution.		

Let's try these few commands first:

- Type `date` to show the current date and time.
- Type `cal` to view a monthly calendar.
- Type `whoami` to show which user you are logged in as.

BAM! You just used command line! Now let's explore and learn more useful commands.

If you ever forget where you are located while cruising through the computer, you can type `pwd` to discover your **p**resent **w**orking **d**irectory. We are currently in the root directory, which is the main directory of the root user. So in other words, we can do anything we want — mwahaha!

`ls` is the command that shows a list of all the files and folders in the directory where you are now.

`cd` is the command for changing directories that allows movement from one directory to another. To leave the root directory and go into the **Examples** folder, for example, you would type `cd examples` and press **Enter**. Note that directory names will autocomplete if you press **Tab** after typing enough of the word, but also note that directory names are case-sensitive.



`mkdir` is used to make a directory/folder in the location where you are located. Let's make a folder to store all the things your trusty cat might make by typing `mkdir kitten_code` or `kitten.code`. Note that you can name the directory whatever you want, but spaces make life more difficult than it needs to be, so we suggest using underscores (`_`) to avoid ambiguity.

(Still not sure how or what to name a file? Check out the [rules and regulations for naming folders and files](#).)

You can also make a directory outside the one you are currently in. An example of this would be `mkdir -p ~/examples/other_folder`. Remember that `~` means home directory and `-p` stands for parents, and tells the command to also create any parent directories that don't already exist.

Now change back into the directory you just made by using `cd kitten_code`.

When creating a file, be sure to add the file type (`.txt`, `.csv`, `.pdf`, etc.) after the file name, and remember to follow the same naming rules as directories.

Type `touch ball_of_yarn.txt`. You could use Vim to edit this file, but we will go over that in the next section.

Now that we are done, you can type `cd` to go back to the home directory or `cd ..` to go up just one level.

Let's look at two commands that are similar so we can keep them straight: `mv` for move and `cp` for copy.

Example syntax for the two looks like this:

```
mv current/location/of/file.txt new/location/file.txt
```

or

```
cd current/location/of/file.txt new/location/file.txt
```

Moving is essentially changing the file path name, so you can actually use `mv` to rename a file. If you would like to move `ball_of_yarn.txt` to the home directory and change its name to `wheres_my_food.txt`, you would type:

```
mv ~/examples/kitten_code/ball_of_yarn.txt ~/wheres_my_food.txt
```

If you wanted to make a copy of the file and keep the file name as it is, you would use this:

```
cp ~/examples/kitten_code/ball_of_yarn.txt ~/wheres_the_food.txt
```

Groovy! You can now rename, make and move files and folders around in command line. Ready for what's next?

Installing things

Now is a good time to remind your cat (or vice versa) about good computer hygiene. No, we don't mean washing your paws before walking on the keyboard, though that's not a bad idea. We mean things like being careful about what you download and install, and also routinely creating backups.

Backups are an essential part of being a responsible computer owner. With backups, if something unfortunate should happen, you won't have to start your computer life over from scratch.

Before we download and install things, it's time for you and your cat to have "the talk" about downloading. You will always need to exercise good judgement when downloading and installing. Check out [this great article](#) to help judge whether something is safe to download.

Image courtesy of [Kent Wang](#) on Flickr. Licensed via Creative Commons.



Installation tools allow you to install a program, and are specific to the operating system (OS) you are working on. We are going to use an installation tool called **yum** because it comes installed by default on many Linux operating systems, including CentOS, the OS that the Puppet Learning VM runs.

Your package may need other packages to run properly — these are known as **dependencies**. Luckily, yum will automatically update and handle all dependencies for you. This means instead of many commands to install dependencies, you simply need to type: `yum install subversion`. Yum figures out what the most recent version of subversion is, gets it for you and makes everything groovy.

If you're having trouble installing packages via yum, you should try the commands:

- `puppet resource yumrepo base enabled=1`
- `puppet resource yumrepo epel enabled=1`

If you are still having computer woes, try [StackOverflow](#).

Congratulations! You are now equipped to make all the things happen on your computer via command line. The next section will cover editing files in command line with Vim.

Five things to remember about command line:

- You need to press **Enter** after each command or nothing will happen.
- If you ever get lost in the terminal, don't panic. Just use `pwd` and think of it as a GPS that shows your current location, or present working directory.
- If you are completely lost, or just want to go back to the home directory, use the `cd` command.
- When creating new files, remember to add the file type at the end of the new file name, like this: `new_file.txt`.
- There are many resources for help with CLI when you are stuck or confused, including [StackOverflow](#) and FossWire's command [Cheat Sheet](#).



The World of Vim, full of terror and confusion.

From [Wikimedia Commons](#).

Learning Puppet with Vim

Next, we are going to learn to use Vim, the popular (and free) text editor.

Remember when we made the file `ball_of_yarn.txt` earlier? Now we are going to use Vim to go inside the file and fill it with text. Keep this book open, grab your towel and don't panic!

Type `vim ball_of_yarn.txt` in your terminal.

Everything went dark! But don't worry — you, your computer and your cat are not blind or dead, you are just in Vim. If you were to type now (DON'T DO IT!), Vim would act in bizarre and terrifying ways. This happens because the initial mode for Vim upon opening is called **Normal**, and is where you tell the computer to do things to the file. But we want to be able to type inside the file. In order to do so, Vim must be in **Insert** mode.

Entering **Insert** mode is easy — just type `i`. You should now see `--INSERT--` at the bottom of the window. Now when you type, it will appear at the top of the window. Go ahead and write your cat a letter, or anything else you find useful.

Once you are done writing, you must return to **Normal** mode to save the new letter as a file. You can do this by pressing the **Escape (esc)** key. Here, typing `:w` will save the file as it is. Or you can type `:w newfilename.txt` to save it as a new file.

Now that you and kitty are done, you can quit Vim by typing `:q` to quit and then `!` to say, “Yes, I’m sure I want to quit.”

You can combine these commands by typing `:wq!` to say, “Write the file, save the file, and yes I’m sure I want to quit.”

If there's a fire alarm or other emergency, you can quickly use `ZZ` to quit and save the file for you. Alternatively, if you caused the fire by what you typed or otherwise, you may want to quit without saving the changes by typing `ZQ`.

If you want to read the letter after quitting Vim, use the `cat` command. It stands for **concatenated**, and is a versatile CLI command. You can use this command to copy contents of one file to another file, or to smash several files together. Want to do more with `cat`? Check out [this great resource](#).

Now type `cat ball_of_yarn.txt`. This should pull up the text of the letter you and your cat wrote earlier. Congratulations! You wrote a thing and pulled it back up in Vim!

Since you won't always be creating a new file every time you go to type something, let's go over how to make changes to an existing file. Let's get back in with `vim ball_of_yarn.txt`.

Moving around in normal mode

You may have noticed when typing in Vim that you still cannot move the cursor with the mouse. Instead, you will have to use arrow keys on the keyboard. Man, this is going to be a pain in the butt! Luckily, there are a whole bunch of shortcuts you can use in **Normal** mode without going into **Insert** mode.

The shortcut keys are found along the main row of keys:

- `h` - moves the cursor one space left
- `j` - moves the cursor one line down
- `k` - moves the cursor one line up
- `l` - moves the cursor one space right

Using these keys, you can maneuver the cursor below any characters you want to delete, and type `X` to delete them, all without leaving **Insert** mode.

We're moving along, but are we moving fast enough?



NEVER.

Image courtesy of [Yung Luen-Lan](#) on Flickr.
Licensed via Creative Commons.

If you want to move more than one character, you only need to add a number to the directional commands listed above. For example, if we type `3h`, then we move three characters left. If we type `10j`, it will take us 10 lines down.

Sometimes you reaaaaalllly don't want to count the exact number of characters or lines that you want to move. The following commands are extremely useful for those times:

- `gg` moves to the beginning of the file.
- `G` moves to the end of the file.
- `0` moves to the beginning of the line.
- `$` moves to the end of the line.
- `w` moves to the next word.
- `b` moves back one word.

These commands also accept a number before them. For example, if you type `3b`, the cursor will move back 3 words or if you type `2$`, the cursor will move to the end of the line below the cursor.

Making all the things happen

Now we're cruising! We know how to delete bits and pieces with `X`, but let's learn how to do some real damage.

If you are serious about deleting stuff, `d` is the command that you want. Like the commands we learned earlier, there are different variations shown below.

- `dw` deletes a word.
- `d%` deletes from where the cursor is to the end of the line.
- `dd` deletes the whole line the cursor is on.
- `dgg` deletes from where cursor is all the way back to beginning of document.

If you get a bit carried away deleting, you can undo it with the `u` command. You can use it as many times as needed to restore what you deleted.

If you get too carried away undoing what you deleted, you can use the redo button with the command **Control + r**.

If you want to resurrect what you deleted, move to the new spot where you want to place the text and use `p`, which pastes the most recently deleted text.

If you want to copy some text without deleting the original, you will want to learn how to **yank** text out with the `y` command. Remember how `dd` deleted the whole line you were on? Similarly, `yy` copies the whole line you are currently on (and deletes nothing).



Silly human, I own you.

Image courtesy of [travel oriented](#) on Flickr.
License via Creative Commons

If you want to copy a whole paragraph, place the cursor in the paragraph and then use the `yap` command (which stands for **yank around paragraph**). It will yank all the text above and below the cursor until it reaches a blank line in either direction. You can then move the cursor to the place where you want the text, and use `p` to paste the section you just yanked out with `y`.

For more complex yanking, you will need to enter **Visual** mode by moving the cursor to the beginning of the section you want to yank and then typing `v`. At the bottom, you will see that you are now in `--VISUAL--` mode. At the end of the section you want to yank, type `y` to yank the selection to the clipboard. `--VISUAL--` will disappear and you can move to a new location and type `p` to paste what you yanked.

For reference:

- `v` begins your selection.
- `y` end and yanks your selection.
- `p` pastes your selection.
- `yy` yanks the line you are on.
- `yap` yanks the paragraph you are on.

If you need to go digging through the entire file for a word or phrase like “I own you,” you would type `/I own you` and hit **Enter**. Vim will then highlight the text. If there are several instances of the phrase you are looking for, you scroll through them with `n` for next (and `N` for previous).

And now for an old favorite: the find and replace command.

Any guesses for what this command does?

```
:%s/old text/new text/gc
```

It will find “old text” each time it appears in your file and ask if you would like to replace it with “new text.” This is particularly useful for open source things, since you regularly have to find dummy text like “YourIPAddress” in a wall of text to insert your own IP address.

For reference:

- `:` puts the Vim in **Command** mode, like when using it with `w` and `q` earlier.
- `%` is Vim shorthand for “all of it” — if you had replaced `%` with numbers, it would search only those numbered lines.
- `s` is for substitute, `g` is for global (meaning, “make all of the changes to the whole file”) and `c` is for check.
- `gc` is for approving each instance of the replacement with `y` for yes, `n` for no, `a` for all, and `q` for, “That’s it. I’m done. Get me out of here.”
- Remember to hit the **Escape** key to enter **Command** mode before entering these commands. Also remember to hit `i` so that you can type again in **Insert** mode.

We’ve been working with text files, since we don’t know which programming languages you and your cat are experts in, but Vim will format coding syntax for basically everything you could need.

```
puts "somebody set us up the bomb"
base = gets.chomp
puts "all your #{base} are belong_to us"
```

If you want some practice moving around Vim, you could spend some time playing the first few levels of the **Vim Adventures** video game. After the first six levels, you will be offered a six-month subscription to play the remaining levels for \$25.

Wasn’t that both pretty cool and pretty exhausting? Now that this kit and caboodle is wrapped up, we can move on to the next section about version control with Git.

Five cool things to remember about Vim

- Vim is in its **Normal** mode when it begins with an intimidating black screen. This mode is for taking action to a file. Enter **Insert** mode with `i` to type *within* the file.
- While in **Insert** mode, you can move the cursor below a character you want to delete and then use the `X` command. Remember that `dw` deletes the whole **w**ord.
- If you are unhappy with the changes you made, use command `ZQ` to exit Vim without saving the bad changes.
- You can copy text from one section and paste it to another section with the `y`, or yank, command. Other options for yanking are available, like `yy` to yank the whole line.
- Vim has a built-in tutor that you can access by typing `vimtutor -g` in the VM command line. It will give you some practice and teach you the next few steps.



Learning Puppet with Git

Git is a version control system (VCS) or, in other words, a tool to keep track of changes that were made to a file or directory. This is great for figuring out What Went Wrong and How To Fix It.

Git works extremely well for both individuals and collaborative teams. Open source people love it because it is helpful when you have herds of cats making many changes to the same files. It helps to think of Git as a system that keeps track and stores every revision in a document.

Open source contributors often use Git with **GitHub**. Companies can also use Git and GitHub to manage proprietary code. They do this because they expect to have multiple people working on any given project, and when you use GitHub, you are actually sharing changes made to the repo (or repository) with others who have access to that repo.

We are going to start off with Git alone, then work up to pulling code from GitHub. Then we'll finally commit (contribute) code back to GitHub. Did any of that make sense? No? Well then, let's try a metaphor to help digest the concept of Git: packing for a series of trips.

Imagine you are planning a series of trips. You may be going to different locations, bringing your cat, and maybe meeting up with other people, so you need to plan carefully before you pack your suitcase. You lay out toiletries, clothes, laser pointers and catnip on the bed. When you think you've found everything you need for the trip, you put it in the suitcase. You will probably want to keep a list of what you add or take out for the different trips, including different clothes for different climates; it's likely you'll be using the same toiletries.

Image courtesy of **Snipergirl** on Flickr.
License via Creative Commons



STAGED / COMMITTED

Thanks to [John Wright](#) for original image on Flickr.

We can think of your suitcase as a **Git repo**, or Git repository. To open our suitcase and begin packing, use the command `git init` to start a new repo. The bed represents the **working directory**, the clothes you set out represent files, and setting them on the bed is how you **stage** those files. Staging means you plan on using the file in your repository and you do this in Git by typing `git add FILE_NAME.txt`.

When you put everything in your suitcase, this is called a **commit**. You move everything from the bed (the staging area in our packing-for-multiple-trips example) to the suitcase by typing `git commit`. This command will take you into Vim to create a note about the changes you just made to the commit — for example, “added flipflops.” After you’ve added the message, use `:wq` to save and quit Vim, so you can reference the commit later.

(**Pro tip:** Typing `git commit -m 'packed bag for Miami'` lets you skip going into Vim, and uses what you quote after `-m` as the commit message).

Now before we do anything else, let’s do a little housekeeping. Type `git config --global user.name Duke`, replacing “Duke” with your name. Hit **Enter**, then type `git config --global user.email Duke@example.com`, using your email address in place of Duke’s. This sets your name and email for Git on a global level, meaning that these settings apply to everything you work with, within or below the home directory of your computer. You won’t have to type your name and email address for Git ever again. You will want to do this for each computer you use, including the computer with the Learning VM installed on it. Now let’s make our first commit!

Initial commit

When you type `git init`, Git turns whatever directory you are in into a magical suitcase. Let’s make a new directory for the next exercise. Type `cd` to go into your home directory, create a new directory called “magical suitcase” by typing `mkdir magic_suitcase`. You can then go into the new directory with `cd magic_suitcase`.

To initiate Git and track all of the changes to the directory, type `git init`. Next, type `git status` to see what is going on. This tells us that we are in our initial commit (without having packed yet) and that there is nothing to commit (nothing has been laid out to be packed yet). So let's start packing!

Type `vim lucky_shirt.txt` to open the file in Vim. Next go into Vim's **Insert** mode by typing `i` and then type The 'keep calm and git revert' shirt. Next hit **Escape** and type `:wq` to quit the file.

Now, when you type `git status`, it tells us that there is an untracked file: `lucky_shirt.txt`. An untracked change, like this one, is something we have changed but haven't yet staged. It's like something we have put on the floor or the dresser for packing, but haven't yet laid out on the bed to be packed. Helpfully, the `git status` command also tells us how to stage the changed file.

```
Untracked files:
(use "git add <file>..." to include in what will be committed)
```

To stage the lucky shirt and commit it later, use the command `git add lucky_shirt.txt`. It's like you've staged it on the bed so you know that you will be sure to pack it in the suitcase, and won't leave home without it. Now if we type `git status`, it will show:

```
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#       new file:   lucky_shirt.txt
```

Let's create a few more files and add them:

```
(root@learn ~/magic_suitcase)# vim hiking_boots.txt
(root@learn ~/magic_suitcase)# vim sweater.txt
(root@learn ~/magic_suitcase)# git add hiking_boots.txt
(root@learn ~/magic_suitcase)# git add sweater.txt
(root@learn ~/magic_suitcase)# git status
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#       new file:   hiking_boots.txt
#       new file:   lucky_shirt.txt
#       new file:   sweater.txt
```

Let's say you now have enough for the first trip, and we want to go ahead and pack it all in the suitcase. Type `git commit`. This takes us into Vim to make a note about what is being committed. The traditional first message is "initial commit," but we added info for a hiking trip to provide more clarification. Save and quit Vim with `:wq`. Voila! You've just used Git to commit a change! This is the bare bones of using Git to collaborate with others on software or IT projects.



It's getting crowded in here!

Thanks to [Shannon](#) for the original image on Flickr.

Making changes

Maybe you decided that you actually own a luckier shirt than the one you labeled and packed last time. You could change the `lucky_shirt.txt` file to say something like “the Kitty Coderz shirt.” If you type `git status`, you will see that now the lucky shirt file is in the changed but not modified section, instead of the area where things are added to commit. You will still have to type `git add lucky_shirt.txt` to stage any file that you’ve changed.

If you type `git diff`, you can see the exact lines where things have changed in the file. Using `git diff --color` might make it easier to see too. The lines with differences will be between @@ symbols and the committed version will be marked with - signs. Changes will be marked with + signs, like this:

```
diff --git a/lucky_shirt.txt b/lucky_shirt.txt
index 3632b52..e904011 100644
--- a/lucky_shirt.txt
+++ b/lucky_shirt.txt
@@ -1,3 +1,4 @@
-the pink ship it shirt
+the ship it shirt
 and
 the venn diagram shirt
+because a wardrobe can't be lucky enough
```

After adding the new `lucky_shirt.txt` file, you can type `git commit -m 'updated lucky shirt'`. The `-m 'message'` allows you to commit the message without going into Vim. Yup — it’s that easy! Now you have your updated shirt — the Kitty Coderz shirt — packed and ready to go.

Now let’s say you want to remove the file `hiking_boots.txt` because you are going to the beach this time, and not hitting the hiking trails. Typing `git rm --cached hiking_boots.txt` will delete the file from the repository (taking it out of the suitcase) once you commit — but that file will stay in your working directory, and the hiking boots are still available to you.



Let's say you want to remove `my_sweater.txt` from both the repository and the working directory (you never liked that sweater anyway). You leave out the `--cached` part and type `git rm sweater.txt` to remove `my_sweater.txt` everywhere, forever. (Think of `rm` as **remove**.)

For reference:

- `git init` — Only for when you start.
- `git add <filename>` — For adding new or modified files or directories.
- `git status` — Checks for everything staged that we want to have available for packing.
- `git diff` — Allows viewing of exact changes.
- `git commit` — Finishing your packing; zip up the suitcase.
- `i <insert commit message here> esc :wq` — For leaving a note (or “commit message”) about what's inside the suitcase and what is being changed.

Git log

Sometimes you might need to look at things you packed in the past. When you type `git log`, Git will list the commit ID, the author, the committed date and the message for the commit — for example, “In this commit, I updated my lucky shirt.” The commit ID is a really long list of random numbers and characters, and the author is, of course, the person who did the packing.

Borrowing from others

Up until now, we have been packing our magical suitcase that we will bring with us to the airport as we embark on our multi-destination trip. Well, now it's time to level up our capabilities. We can use an external repository, like a company server or GitHub.com, to push a copy of our magical suitcase to every airport that has internet access (or company server access).

If you want to destroy my sweater... DO IT, QUICKLY!

Image courtesy of [Found Animals Foundation](#) on Flickr.
License via Creative Commons



When the Duke loses his train of thought,
he tries to cover his tracks.

Thanks to [Andy Gregorowicz](#) for the original image on Flickr.

Here's the sequence: Set out (add) the contents; pack (commit) them in your suitcase (local repository); then push that suitcase to the internet or company network so that you, your cat and anyone else can grab a copy anywhere, anytime. Don't worry about the wrong people seeing your dirty laundry — you can set up a private repository (or multiple private repositories) if you want, so that nobody else has access to your unmentionables.

Git is useful for tracking what you have already done, but it is even more powerful when you have more than one person working on the same thing. Maybe some other people have already traveled to the places you are going to, and they have already packed pretty awesomely useful suitcases. If someone makes their suitcase available to copy, you can clone it to your own closet, make changes, and then push those changes back to the main suitcase that everyone can access. Ultimately, this is what GitHub is for: sharing changes to files with others who are working with those files.

To be really effective, you need to know about the remote repositories you want to work with, and have permission to push to them. To go with our packing-for-a-trip metaphor, you need to know that other suitcases exist, and you need to have permission to add to them, or change items that are in them. These repositories (or other suitcases) are usually in your internal network, or in a browser-based tool like GitHub.

Setting permissions for the Learning VM is beyond the scope of this section, so we are going to clone from a remote repo and save the changes locally. Or in other words, we are going to copy someone else's suitcase, make changes, but not share those changes back with the original on GitHub or the company server. In other, other words, we're taking a great suitcase, making changes so it's suitable for us, but not imposing our particular changes on others.



If you search for **suitcase** or **magic suitcase** on GitHub.com, you might stumble upon a repo called **thelongshanx/super_suitcase**. We are going to start by simply copying it, but if you want to contribute to a repository on GitHub, you will need to create a GitHub account. Directly below the header **Set Up Git**, select your operating system and follow the instructions. You will also find more instructions for setting up to work with other people later in this same resource.

To get the repo onto your computer, make sure your VM network settings are set to **Bridged**. On your computer (not the VM), open up VirtualBox, select the Learning VM, and click **Network** about halfway down on the right. Then in the drop-down menu where it shows **Connect to:** simply select **Bridged**. The **Bridged** setting tells VirtualBox to share your internet connection.

Next, type `cd` to enter a directory where you would like this repo (suitcase) to live. Next, type `git clone https://github.com/thelongshanx/super_suitcase.git`. You now have your very own copy of the suitcase stored locally, and you can edit it, add to it, remove from it and commit changes to it.

Branches: why you have a specific master

When you create a new repository or clone one, this repository is called the **origin**. Within repositories are branches, and the first branch, by default, is called the **master branch**. When you want to push your changes to the master repository, you type `git push origin master`. You can replace **origin master** with any other repository or branch.

Let's say you've decided to get a new wardrobe, but aren't sure what look you are going for. You don't want to throw away everything you have in the master branch, so you are going to create a new branch. When you type `git branch new_look`, it creates a new branch called **new_look**. This new branch is where you will start experimenting with wardrobe changes.

Five things to remember about Git

- When you need to create a new repo, simply type `git init`.
- If you're not sure what is really going on, use `git status`.
- You can learn useful information about a commit with the `git log` command. Git will show you the commit ID number, name of the author, committed date and the message written for the commit.
- The master branch is your original commit. You can then create multiple branches off the master so you can make changes with `git branch new_branch`. Now you can make changes in the new branch with `git checkout new_branch`.
- Any changes you make on your local files should be pushed to your GitHub account with `git push origin master`.


To begin adding to this branch, you need to first type `git checkout new_look`. You are now in the **new_look branch**, and the master branch will not change as you play with `new_hats.txt` and `nautical_theme_pashmina_afghans.txt`.

You can create as many branches as you like, but you can have only one accessed at a time. If you decide that the **new_look** branch is good enough to become the master, you go back to the **master branch** with `git checkout master` and then type `git merge new_look`. This pushes everything in **new_look** to the master branch.

If, alternatively, the **leopard_print_jumpsuit** branch didn't work out, you can type `git branch -d leopard_print_jumpsuit` to banish that jumpsuit forever.

Sharing with others

Quick note: Until now, we have remained pretty self-contained by making copies and keeping them to ourselves. If you want to contribute some code back to the open source world, however, you will need an account with a remote repository like GitHub. Setting up a GitHub account and browsing through it from the Learning VM is tough, so if you haven't created an account yet, use your browser to register at [GitHub.com](https://github.com).

With an account, you can use your browser to search for interesting things on [GitHub](https://github.com). If, for example, you found **thelongshanx/magic_suitcase** repo and you want to contribute to it, you should click the **Fork** button  on GitHub in the top right part of your browser to create a copy in your GitHub account that you can play with.

You can clone this copy from the browser to your computer almost the same way as we did before, by making a new directory. First type `cd` and type `git init`. However, you must add your login name to give you the power to push changes back to GitHub. Instead of typing:

```
git clone https://github.com/thelongshanx/super_suitcase.git,
```

```
you type git clone https://thelongshanx@github.com/thelongshanx/magic_suitcase.git.
```

You will then be asked for your GitHub password to make sure you're not up to any funny business before you can pull it all down into your working directory. Type `cd magic_suitcase` to get into the shared Git repository.

After making any changes and committing on the computer, you should push that new version to your GitHub repository page. You do so by typing `git push origin master`. (Remember, wherever you are when you make your copy is called **origin**.)

If you want contribute back to the repo you cloned from originally, you need to use the **pull request** button on the far right side of your browser screen when you're on GitHub. The owner of the repository will receive a notification of your request, and will decide whether or not to incorporate your changes.

On the other hand, if there is another GitHub repo besides **origin** that you want to share your changes with, you should copy the SSH URL from the bottom right side of the repository's web page (see image at right).

Next you add the remote repository by typing `git remote add duke git://github.com/the_duke/wardrobe.git`. You should replace **duke** with whatever you want to call the repository, and also copy the last part of the SSH URL that you got from GitHub. You can now push the remote repo by typing `git push duke master` (again, replacing **duke** with whatever name you chose).


SSH clone URL

git@github.com:thed



You can clone with [HTTPS](#), [SSH](#), or [Subversion](#). ?

Cheat sheet for working with GitHub

-  **Fork** in GitHub
- `mkdir <directory name>`
- `cd <directory name>`
- `git init`
- `git clone https://YourUserName@github.com User/Project.git`
- `cd <project>`
- Change, add, commit.
- `git push origin master`
- Click **submit pull request** to submit your pull request.



Congratulations!

It's been a long journey, but those are the most rewarding, right? We covered a whole lot of stuff, and you should be proud of yourself.

Once you are comfortable using the commands you have learned in this series, you should check out some of the other wonderful resources the internet has come up with. The following section includes some recommendations and additional resources for information about topics we covered this in ebook.

For more great how-to articles and technical advice, visit puppet.com/blog

You're not kitten around!

If your cat's claws are trimmed, you should give each other a high five. You did it!

Thanks to [Joachim S. Muller](#) for the original image on Flickr.

Tiffany Longworth authored *The Tools for Learning Puppet* out of her experience as someone who didn't know what she needed to know before she could learn the things she needed to learn, and empathy for anyone who who has encountered a "basics" class that wasn't so basic.

About Puppet

Puppet is driving the movement to a world of unconstrained software change. Its revolutionary platform is the industry standard for automating the delivery and operation of the software that powers everything around us. More than 30,000 companies — including more than two thirds of the Fortune 100 — use Puppet's open source and commercial solutions to achieve situational awareness and drive software change with confidence. Based in Portland, Oregon, Puppet is a privately held company with more than 400 employees around the world. For more information, visit puppet.com.

Resources

- Download [VirtualBox](#) so you can host a VM on your computer. It's free!
- Download the [Learning Puppet VM](#) — also free!
- If you're looking for enterprise-level configuration management software that's fully tested, scalable and fully supported, you can download and try out [Puppet Enterprise](#) for free.
- Once you've downloaded Puppet Enterprise, learn foundational Puppet concepts and best practices for managing your infrastructure with [Puppet Fundamentals](#).
- Learn how to manage all aspects of a Windows system configuration with Puppet Enterprise, leveraging existing Puppet modules with [Puppet Essentials for Windows](#).
- [Learn Code the Hard Way](#) — Slightly sassy dude tells you to, "Shut up and shell."
- The [Puppet blog](#) has lots of helpful technical posts.

CLI

Linux & Mac OSX

Get in a terminal

- Linux: Let Lifehacker.com help you choose a [Linux terminal emulator](#).
- Mac OSX: Go to **Finder**, then **Applications**, then **Utilities** and select **Terminal**.

[Learning the Shell](#) — A darn good resource to learn Linux, period.

Cheat sheets:

- Dave Child's [Linux command line cheat sheet](#)
- [Linux command line reference for common operations](#)
- [Linux bash shell cheat sheet](#)

Windows (Powershell)

- Get [instructions](#) for installing and getting started with Powershell.
- Check out the [Learn](#) and [Library](#) sections on [technet.microsoft.com](#).

Vim

Vim has a built-in tutor that you can access by typing `vimtutor -g` in the VM command line. It will give you some practice and teach you the next few steps.

[Vimdoc](#) — The manual written by Bram Moolenaar, the author of Vim. If you want to jump straight to the review and new stuff, go to [this page](#).

Git

[Git cheat sheet](#) — A cool interactive cheat sheet to learn the different places where you do things or put things in a Git workflow. It will show you some useful next-level commands, which you can then review more thoroughly with `git help <command name>`.

[Git Tutorial](#) — A good reference for visual learners from Atlassian (which makes Bitbucket, an alternative to GitHub).

[Pro Git](#) — The entire Pro Git book written by Scott Chacon. Scott goes into far more depth than what is covered here. Thankfully, this wonderful resource is available for free.