3: Cookbooks

Cookbooks
Organizing Recipes

©2015 Chef Software Inc.

CHEF

Slide 2

Objectives

After completing this module, you should be able to:

➢ Modify a recipe

➢ Use version control

➢ Generate a Chef cookbook

➢ Define a Chef recipe that sets up a web server

In this module you will learn how to modify a recipe, use version control, generate a Chef cookbook and define a Chef recipe that sets up a web server.

Slide 3

Answers:

1. The recipe that you put together to setup the workstation proved useful--useful enough to see if the same could be done with a webserver. It's a package, a file, and a service. Everything you've already completed. Well, almost everything.

2. Now the request to add version control and a README would definitely make it easier to share the recipes that we create. Without version control we'd have no way to build this software collaboratively or recover our work. Without a README no one would know what the recipe even was suppose to do or what it did.

3. And yes, before we start creating more recipes and cookbooks, we should choose a versioning solution.

Slide 4
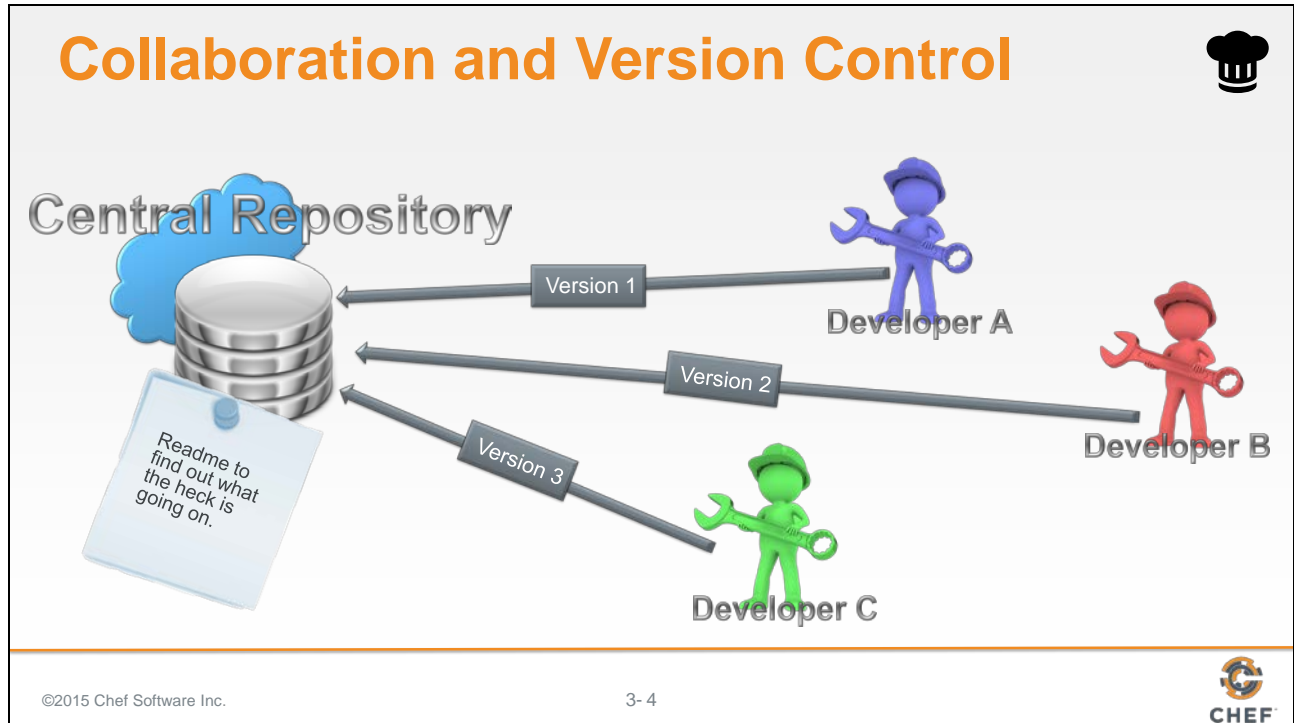
Before we answer that question, let's talk about collaboration. Usually, none of us work in a vacuum, and it's important that systems are in place to make collaboration easier. One such system is versioning. Versioning will make it easier to share the recipes that we create.

A versioning system should include:

A Central Repository into which all the developers publish their work.
Each revision should be stored as a new version.
For each change, a readme file should be added so that everyone knows what has or has not been changed

Slide 5

Versioning Pros and Cons

```
$ cp setup.rb setup.rb.bak
or
$ cp foo{,.`date +%Y%m%d%H%M`}
or
$ cp foo{,.`date +%Y%m%d%H%M`-`$USER`}
```

Saving a copy of the original file as another filename.

Lets explore this first option of renaming the file by adding a quick extension, like in the first example shown here. In this way we can keep working on the original file as we add more features. As a group lets talk about the pros and cons of using this strategy.

So obviously a single backup won't do. We need backups more often as we are going to be iterating quickly.

We could use the current date and time down to the minute like in the second example. As a group lets talk about the pros and cons of using this strategy.

Would adding the user's name to the end of the file, like in the third example, solve the problems we are facing with other choices? Again what are the pros and cons of this new approach?

Slide 6

How about we use git?

What are the pros and cons of this approach?

For the rest of this course we will be using git. This may not be the version control software you use on your teams or within your organization and that is alright. Our use of git within this course is used solely to demonstrate the use of version control when developing Chef code. When you develop with Chef you are welcome to use the version control system of your choice.

Instructor Note: It is not important that the learners understand and learn all of git during this course. It is more important that the learners understand when and where to use version control to save their work. This is about training them on making changes, testing, and then committing their work. Version control is an instrumental piece of the workflow when you adopt Infrastructure as code. There are some benefits of learning and using git because Chef uses git and GitHub to do almost all development of Chef. The majority of the Chef community uses git and GitHub.

Slide 7

Lab: Install git

❑ Add the additional policy to the "setup.rb":

The package named "git" is installed.

❑ Then apply this recipe with chef-apply.

Is git installed? Do we know if it will be installed with every new instance that is setup?

It sounds like we need the tool now to store our cookbook but we also want to define a policy that git is installed on all of our workstations. Update the setup recipe to define the new policy and apply the setup recipe again.

Instructor Note: Allow 5 minutes to complete this exercise.

Slide 8

Lab: Adding the git Package

```
~/setup.rb

package "nano"
package "vim"
package "emacs"

package "tree"
package "git"                                                          +

file "/etc/motd" do
  content "Property of ..."
end
```

We add a package resource named 'git' to the setup recipe within our workstation cookbook.

Slide 9

Lab: Re-apply the Setup Recipe

```
$ sudo chef-apply setup.rb

Recipe: (chef-apply cookbook)::(chef-apply recipe)
  * yum_package[nano] action install (up to date)
  * yum_package[vim] action install (up to date)
  * yum_package[emacs] action install (up to date)
  * yum_package[tree] action install (up to date)
  * yum_package[git] action install
    - install version 1.7.1-3.el6_4.1 of package git
  * file[/etc/motd] action create (up to date)
```

Then we use chef-apply to apply our recipe.

Slide 10

GE: Create a Cookbook

How are we going to manage this file? Does it need a README?

Objective:

- ❏ Use chef to generate a cookbook to store our setup recipe.
- ❏ Add the "workstation" cookbook to version control.

The setup recipe now installs everything we currently need on our workstation.

But before throw this recipe file into a directory with our other scripts we should look at a concept in Chef called a cookbook.

What is a cookbook? How do we create one? Let's ask 'chef'.

Slide 11

CONCEPT

What is 'chef'?

An executable program that allows you generate cookbooks and cookbook components.

3-11

In this context, 'chef' is a command, not the company.

What's the best way to learn Chef? Use Chef. We want you to literally run 'chef'.

Slide 12

What can 'chef' do?

```
$ chef --help

Usage:
    chef -h/--help
    chef -v/--version
    chef command [arguments...] [options...]


Available Commands:
    exec       Runs the command in context of the embedded ruby
    gem        Runs the `gem` command in context of the embedded ruby
    generate   Generate a new app, cookbook, or component
    shell-init Initialize your shell to use ChefDK as your primary ruby
    install    Install cookbooks from a Policyfile and generate a locked cookbook
set
    update     Updates a Policyfile.lock.json with latest run_list and cookbooks
```

©2015 Chef Software Inc.   3-12

'chef' is a command-line application that does quite a few things. The most important thing to us right now is its ability to generate cookbooks and components.

Alright. So 'chef' can generate a cookbook. But what is the purpose of a cookbook? That sounds like we should read the documentation.

Slide 13

Cookbooks

A Chef cookbook is the fundamental unit of configuration and policy distribution.

Each cookbook defines a scenario, such as everything needed to install and configure MySQL, and then it contains all of the components that are required to support that scenario.

Read the first three paragraphs here: http://docs.chef.io/cookbooks.html

It's important that you learn to read the Chef documentation. Lets look up cookbooks in Chef's documentation. Visit the docs page on cookbooks and read the first three paragraphs.

Slide 14

What Can 'chef generate' Do?

```
$ chef generate --help

Usage: chef generate GENERATOR [options]

Available generators:
  app        Generate an application repo
  cookbook   Generate a single cookbook
  recipe     Generate a new recipe
  attribute  Generate an attributes file
  template   Generate a file template
  file       Generate a cookbook file
  lwrp       Generate a lightweight resource/provider
  repo       Generate a Chef policy repository
  policyfile Generate a Policyfile for use with the install/push commands
(experimental)
```

©2015 Chef Software Inc. 3-14

A cookbook is a structure that contains recipes. It also contains a number of other things--but right now we are most interested in a finding a home for our recipes, giving them a version, and providing a README to help describe them.

Slide 15

What Can 'chef generate cookbook' Do?

```
$ chef generate cookbook --help
Usage: chef generate cookbook NAME [options]
    -C, --copyright COPYRIGHT         Name of the copyright holder - defaults
to 'The Authors'
    -m, --email EMAIL                 Email address of the author - defaults to
'you@exa...
    -a, --generator-arg KEY=VALUE     Use to set arbitrary attribute KEY to
VALUE in the...
    -I, --license LICENSE             all_rights, httpd, mit, gplv2, gplv3 -
defaults ...
    -g GENERATOR_COOKBOOK_PATH,       Use GENERATOR_COOKBOOK_PATH for the
code_generator...
        --generator-cookbook
```

CHEF

Lets examine the 'chef generate' command. We can see that the command is capable of generating a large number of different things for us. It looks like if we want to generate a cookbook we're going to need to use 'chef generate cookbook'.

Lets ask the 'chef generate cookbook' command for help to see how it is used.

Slide 16

GE: Let's Create a Cookbook

```
$ chef generate cookbook workstation
Compiling Cookbooks...
Recipe: code_generator::cookbook
* directory[/home/chef/workstation] action create
    - create new directory /home/chef/workstation
  * template[/home/chef/workstation/metadata.rb] action create_if_missing
    - create new file /home/chef/workstation/metadata.rb
    - update content in file /home/chef/workstation/metadata.rb from none to bd85d3
    (diff output suppressed by config)
  * template[/home/chef/workstation/README.md] action create_if_missing
    - create new file /home/chef/workstation/README.md
    - update content in file /home/chef/workstation/README.md from none to 44d165
    (diff output suppressed by config)
  * cookbook_file[/home/chef/workstation/chefignore] action create
```

©2015 Chef Software Inc.   3-16

To generate a cookbook, all we have to do is provide it with a name.

Naming things: There are two hard things in Computer Science and one of those is giving something a name.

We have you covered. Call the cookbook *workstation*. That's a generic enough name.

We want you to use 'chef generate' to generate a cookbook named *workstation*.

Slide 17

Aren't you curious what's inside it? Lets take a look with the help of the '`tree`' command. If we provide '`tree`' with a path we will see all the visible files in the specified directory.

So the chef cookbook generator created an outline of a cookbook with a number of default files and folders. The first one we'll focus on is the README.

Slide 18

All cookbooks that 'chef' will generate for you will include a default README file. The extension `.md` means that the file is a markdown file.

Markdown files are text documents that use various punctuation characters to provide formatting. They are meant to be easily readable by humans and can be easily be rendered as HTML or other formats by computers.

Slide 19

GE: The Cookbook Has Some Metadata

```
$ tree workstation

workstation
├── Berksfile
├── README.md
├── chefignore
├── metadata.rb
├── README.md
├── recipes
│   └── default.rb
├── spec
│   ├── spec_helper.rb
│   └── unit
│       └── recipes
10 directories, 9 files
```

The cookbook also has a metadata file.

Slide 20

metadata.rb

Every cookbook requires a small amount of metadata. Metadata is stored in a file called metadata.rb that lives at the top of each cookbook's directory.

http://docs.chef.io/config_rb_metadata.html

©2015 Chef Software Inc.   3-20

This is a ruby file that contains its own domain specific language (DSL) for describing the details about the cookbook.

Slide 21

GE: Let's Take a Look at the Metadata

```
$ cat workstation/metadata.rb
name            'workstation'
maintainer      'The Authors'
maintainer_email 'you@example.com'
license         'all_rights'
description     'Installs/Configures workstation'
long_description 'Installs/Configures workstation'
version         '0.1.0'
```

If you view the contents of your new cookbook's metadata, you'll see a number of details that help describe the cookbook:

The name of the cookbook, its maintainer, a way to reach them, how the cookbook is licensed, descriptions, and the cookbook's version number.

Slide 22

GE: The Cookbook Has a Folder for Recipes

```
$ tree workstation
workstation
├── Berksfile
├── README.md
├── chefignore
├── metadata.rb
├── README.md
├── recipes
│   └── default.rb
├── spec
│   ├── spec_helper.rb
│   └── unit
│       └── recipes
10 directories, 9 files
```

The cookbook also has a folder named *recipes*. This is where we store the recipes in our cookbook. You'll see that the generator created a default recipe in our cookbook. What does it do?

Slide 23

GE: The Cookbook Has a Default Recipe

```
$ cat workstation/recipes/default.rb

# Cookbook Name:: workstation
# Recipe:: default
#
# Copyright (c) 2015 The Authors, All Rights Reserved.
```

Looking at the contents of the default recipe you'll find it's empty except for some ruby comments.

A cookbook doesn't have to have a default recipe but most every cookbook has one. It's called *default* because when you think of a cookbook, it is probably the recipe that defines the most common configuration policy.

Slide 24

GE: Copy the Recipe into the Cookbook

```
$ mv setup.rb workstation/recipes/setup.rb
```

3-24

From the Home directory, move your setup.rb recipe to the workstation cookbook and place it alongside our default recipe.

Slide 25

Group Exercise: Version Control

This is a probably a good point to capture the initial state of our cookbook.

Objective:

- ✓ Use chef to generate a cookbook to store our setup recipe.
- ❑ Add the "workstation" cookbook to version control.

Now that we have our cookbook with its README and version number, it's time to start tracking our changes with git.

Slide 26

GE: Move into the Cookbook Directory

```
$ cd workstation
```

Change into the workstation cookbook directory.

Slide 27

We want git to start tracking the entire contents of this folder and any content in the subfolders. To do that with git, you need to execute the command 'git init' in the parent directory of the cookbook that you want to start tracking.

Instructor Note: You will notice that git will say that the repository has been 'Reinitialized'. This is because the chef cookbook generator detected that we have git installed and automatically initialized the cookbook as a git repository.

Slide 28

GE: Use 'git add' to Stage Files to be Committed

```
$ git add .
```

Now we need to tell git which files it should start tracking in source control. In our case, we want to add all the files to the repository and we can do that by executing `'git add .'` (dot).

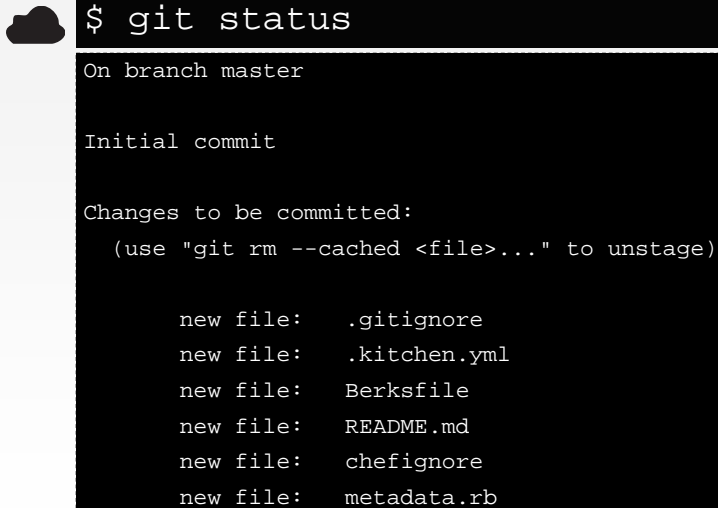This will place all the files into a staging area.

Slide 29

You can think of the staging area as a box in which to put a bunch of items -- like a care package you would send to someone.

Staging files means to put them in the box, but don't close it up because you may add a few things, and don't close it up because you may replace or remove a few things. But put the items in the box because eventually we are going to close that box when it is ready to send it off.

Slide 30

GE: Use 'git status' to View the Staged Files

```
$ git status

On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   .gitignore
        new file:   .kitchen.yml
        new file:   Berksfile
        new file:   README.md
        new file:   chefignore
        new file:   metadata.rb
```

Lets see what changes we have placed in the staging area.

Thinking about our care package example, this is like looking inside the box and taking an inventory, allowing us to figure out if we need to move more things in or remove things we accidently threw in there.

Running `git status` allows us to see in the box. Git reports back to us the changes that will be committed.

Instructor Note: Git helpfully tries to show you the command you can use to remove an item from that box. This is useful if you want to include all items excepts for one or simply manage everything before you commit.

Slide 31

GE: Use 'git commit' to Save the Staged Changes

```
$ git commit -m "Initial workstation cookbook"

master (root-commit) 9998472] Initial workstation cookbook
 Committer: ChefDK User <chef@ip-172-31-59-191.ec2.internal>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly:

    git config --global user.name "Your Name"
    git config --global user.email you@example.com

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author
```

If everything that is staged looks correct, then we are ready to commit the changes.

This is like saying we're ready to close the box up.

This is done in git with **git commit**. We can optionally provide a message on the command-line and that is done with the **-m** flag and then a string of text that describes that change.
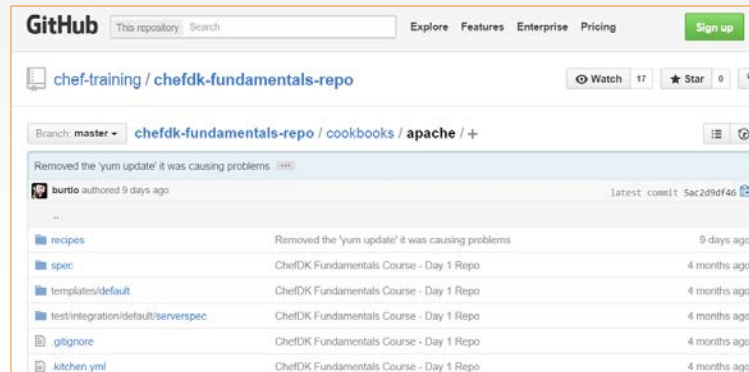
Slide 32
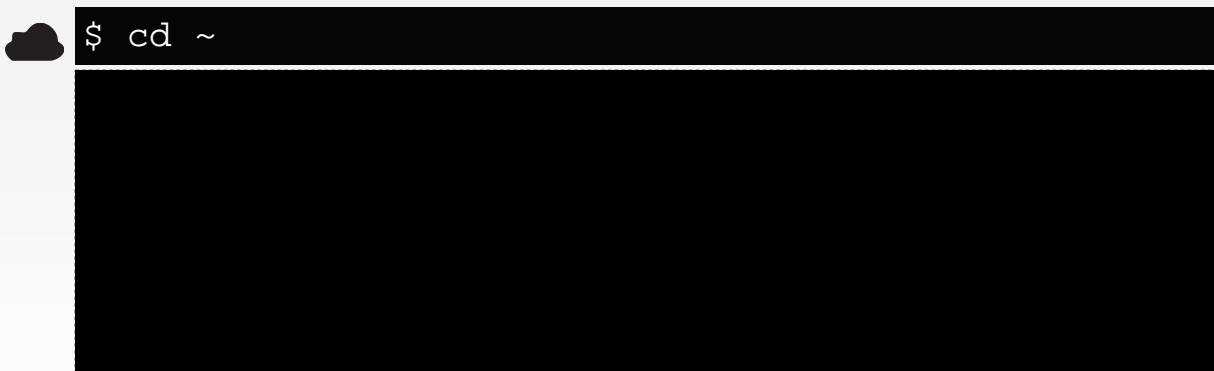
git tracks all our commits, all those closed up boxes, locally on the current system. If we wanted to share those commits with other individuals we would need to push those changes to a central repository where we could collaborate with other members of the team.

Slide 33

GE: Move out of the Workstation Cookbook

```
$ cd ~
```

©2015 Chef Software Inc.   3-33

Now that we are done adding our workstation cookbook to version control lets return to our home directory.

Slide 34

Lab: Setting up a Web Server

- ❏ Use `chef generate` to create a cookbook named "apache".
- ❏ Write and apply a recipe named `"server.rb"` with the policy:
  - The package named "httpd" is installed.
  - The file named "/var/www/html/index.html" is created with the content "<h1>Hello, world!</h1>"
  - The service named "httpd" is started and enabled.
- ❏ Apply the recipe with chef-apply
- ❏ Verify the site is available by running `curl localhost`

©2015 Chef Software Inc.   3-34

Now. Here is your latest challenge. Deploying a Web Server with Chef.

Thinking about all that we have accomplished so far that hopefully seems possible.

We need a cookbook named apache that has a server recipe. Within that server recipe we need to install the appropriate package. Write out an example HTML file, and then start and enable the service.

Then we should apply that recipe and make sure the site is up and running by running a command to visit that site.

So show me it can be done!

Instructor Note: Allow 15 minutes to complete this exercise.

Slide 35

Lab: Create a Cookbook

```
$ chef generate cookbook apache

Compiling Cookbooks...
Recipe: code_generator::cookbook
  * directory[/home/chef/apache] action create
    - create new directory /home/chef
  * template[/home/chef/apache/metadata.rb] action create_if_missing
    - create new file /home/chef/apache/metadata.rb
    - update content in file /home/chef/apache/metadata.rb from none to bd85d3
    (diff output suppressed by config)
  * template[/home/chef/apache/README.md] action create_if_missing
    - create new file /home/chef/apache/README.md
    - update content in file /home/chef/apache/README.md from none to 44d165
    (diff output suppressed by config)
  * cookbook_file[/home/chef/apache/chefignore] action create
```

©2015 Chef Software Inc.   3-35

From the Chef home directory, run the command 'chef generate cookbook apache'. This will place the apache cookbook alongside the setup cookbook.

Slide 36

Lab: Create Apache Recipe

`~/apache/recipes/server.rb`

```
package "httpd"

file "/var/www/html/index.html" do
  content "<h1>Hello, world!</h1>"
end

service "httpd" do
  action [ :enable, :start ]
end
```

The server recipe, found at ~/apache/recipes/server.rb, defines the policy:

* The package named **httpd** is installed.

* The file named "/var/www/html/index.html" is created with the content "Hello, world!"

• The service named **httpd** is started and enabled.

Instructor Note: The service action defines two actions within a Ruby array. Ruby arrays are ordered, integer-indexed collections of any object. Each element in an array is associated with and referred to by an index.

Slide 37

Lab: Apply the Server Recipe

```
$ sudo chef-apply apache/recipes/server.rb
Recipe: (chef-apply cookbook)::(chef-apply recipe)
  * yum_package[httpd] action install
    - install version 2.2.15-47.el6.centos of package httpd
  * file[/var/www/html/index.html] action create
    - create new file /var/www/html/index.html
    - update content in file /var/www/html/index.html from none to 17d291
    --- /var/www/html/index.html       2015-09-14 22:57:21.151137524 +0000
    +++ /var/www/html/.index.html20150914-2132-n4lsm6   2015-09-14
22:57:21.150137524 +0000
    @@ -1 +1,2 @@
    +<h1>Hello, world!</h1>
  * service[httpd] action enable
    - enable service service[httpd]
  * service[httpd] action start
```

©2015 Chef Software Inc.   3-37

When applying the recipe with '**chef-apply'**, you need to specify the partial path to the recipe file within the apache cookbook's recipe folder.

Slide 38

Lab: Verify That the Website is Available

```
$ curl localhost

<h1>Hello, world!</h1>
```

You already setup apache, which is a web server. So verify that the website is available and returns the content we expect to see.

Slide 39

Now, with everything working it is time to add the apache cookbook to version control.

- Move into the apache directory.
- Initialize the cookbook as a git repository.
- Add all the files within the cookbook.
- And commit all the files in the staging area.

Slide 40

Answer these questions.

With your answers, turn to another person and alternate asking each other asking these questions and sharing your answers.

Slide 41

Q & A

What questions can we answer for you?

- Cookbooks
- Versions
- Version control

©2015 Chef Software Inc.   2-41

What questions can we help you answer?

General questions or more specifically about cookbooks, versioning and version control.

Slide 42