# SCSA2611 DEVOPS LAB

1) A) Version Control to Perform Version Control on websites/ Software using different Version control tools like GIT

**Aim:**

**To install and configure the Git on Ubuntu to perform version control**

**Prerequisites**

- Access to a user account with sudo or root privileges.
- A machine running Ubuntu 20.04 or 22.04.
- Access to a command line/terminal window (**Ctrl**+**Alt**+**T**).

**Starting with Basic Git Commands on Ubuntu**

The following is a list of useful Git commands to help you get started:

- Find changed files in the working directory: **git status**
- Change to tracked files: **git diff**
- Add all changes to your next commit: **git add**
- Add selected changes into your next commit: **git add -p**
- Change the last commit: **git commit -amend**
- Commit all local changes in tracked files: **git commit -a**
- Commit previously staged changes: **git commit**
- Rename a Local branch **git branch -m new-name**
- List all currently configured remotes: **git remote -v**
- View information about a remote: **git remote show**
- Add a new remote repository: **git remote add**
- Delete a remote repository **git remote remove [remote name]**
- Download all changes from a remote repository: **git fetch**
- Download all changes from and merge into HEAD: **git pull branch**
- Create a new branch with the command: **git branch first-branch**
- To see more Git commands use: **git --help**

**To install Git**
From your shell, install Git using apt-get:

```
$sudo apt-get update or sudo apt update
```

```
$ sudo apt-get install git  or sudo apt install git
```

**To verify Git**
Verify the installation was successful and version by running:

```
$git –version
```

You will get

```
git version 2.9.2
```

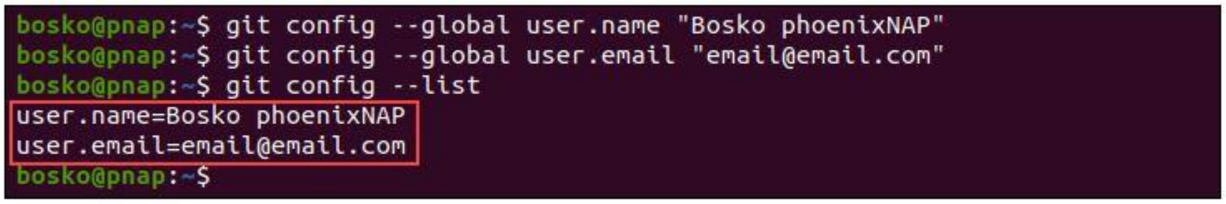
**Configuring Git on Ubuntu after Installation**
Configure your Git username and email using the following commands, replacing your_name with your own name and email address

$ git config --global user.name "your_name"

$ git config --global user.email " email@address.com "

**Verify configuration changes with the command:**
$git config –list

The system should display the name and email address you just entered.


```
bosko@pnap:~$ git config --global user.name "Bosko phoenixNAP"
bosko@pnap:~$ git config --global user.email "email@email.com"
bosko@pnap:~$ git config --list
user.name=Bosko phoenixNAP
user.email=email@email.com
bosko@pnap:~$
```

**Creating your own first Git Repository:**
$ git init project1

$ cd project1

**Create a file named file1.txt:**
$ sudo nano file1.txt

$ git add file1.txt

$ git commit –m ”My first commit”

**Cloning the existing Repository**
You clone a repository with **git clone <url>.**

For example, if you want to clone the Git linkable library called **libgit2**, you can do so like this:

$ git clone https://github.com/libgit2/libgit2

If you want to clone the repository into a directory named something other than **libgit2**, you can specify the new directory name as an additional argument:

$ git clone https://github.com/libgit2/libgit2 mylibgit

That command does the same thing as the previous one, but the target directory is called mylibgi

**Reference:**

1. https://git-scm.com/docs
2. https://git-scm.com/book/en/v2
3. https://phoenixnap.com/kb/how-to-install-git-on-ubuntu
4. https://phoenixnap.com/kb/git-commands-cheat-sheet
5. https://www.atlassian.com/git/tutorials/install-git

# *Linux: Understanding Revision Control System (RCS)*

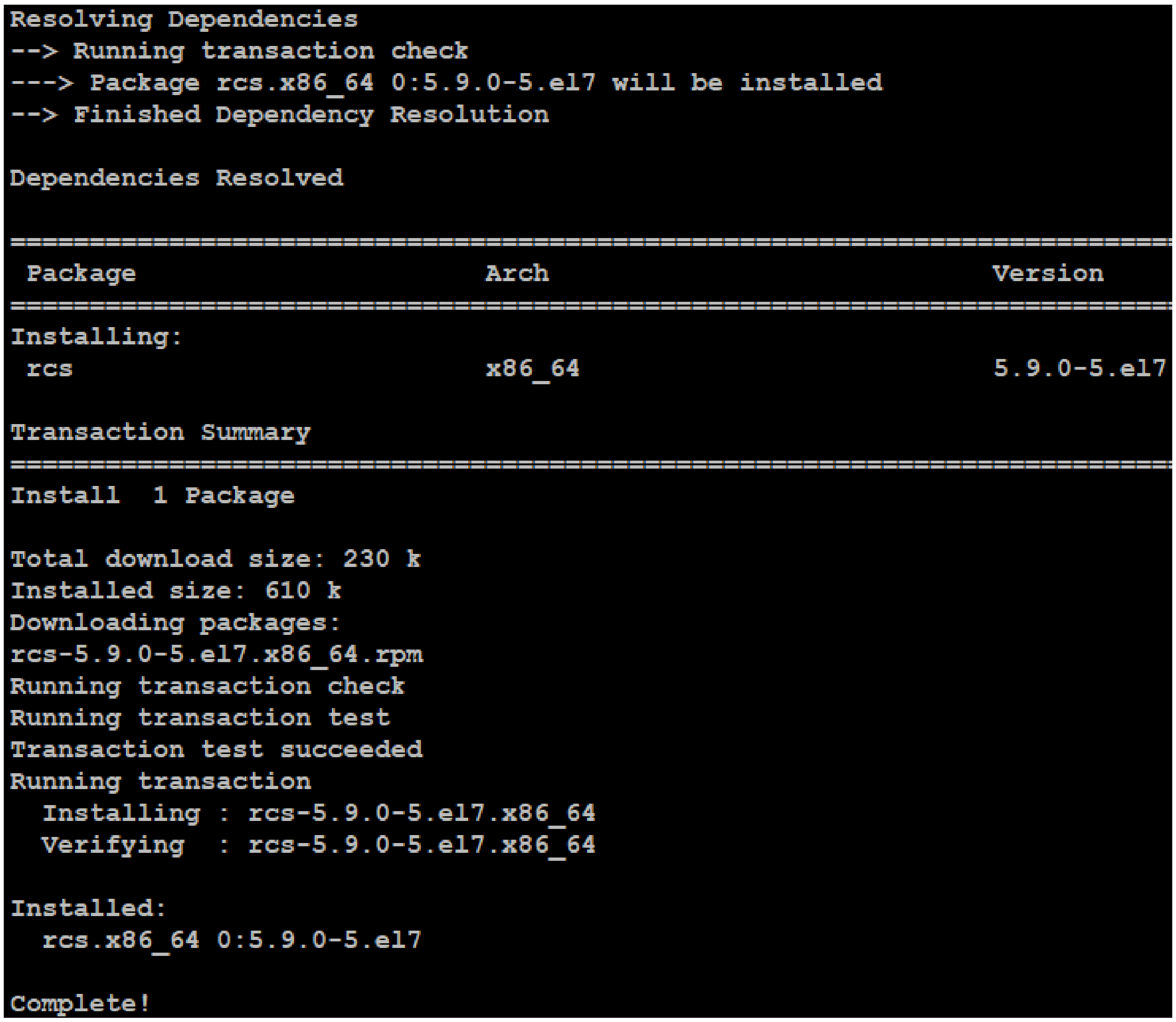
## *INTRODUCTION*

The **Revision Control System (RCS)** is a system for managing multiple versions of files. It automates the storage, retrieval, logging, identification and merging of file revisions. RCS is useful for text files that are revised frequently (for example, programs, documentation, graphics, papers and form letters).
The rcs package should be installed if you need a system for managing different versions of files.

## *INSTALLATION*

Usually the package ships with default OS repository, you just need to install it.

```
[root@serverd ~]# yum install rcs -y
```



```
Resolving Dependencies
--> Running transaction check
---> Package rcs.x86_64 0:5.9.0-5.el7 will be installed
--> Finished Dependency Resolution

Dependencies Resolved

=========================================================================
 Package                      Arch                          Version
=========================================================================
Installing:
 rcs                          x86_64                        5.9.0-5.el7

Transaction Summary
=========================================================================
Install  1 Package

Total download size: 230 k
Installed size: 610 k
Downloading packages:
rcs-5.9.0-5.el7.x86_64.rpm
Running transaction check
Running transaction test
Transaction test succeeded
Running transaction
  Installing : rcs-5.9.0-5.el7.x86_64
  Verifying  : rcs-5.9.0-5.el7.x86_64

Installed:
  rcs.x86_64 0:5.9.0-5.el7

Complete!
```

## *Fundamental operations*

It majorly work in two modes: checkin and checkout.

The “**checkin(ci)**” operation records the contents of the working file in the RCS file, assigning it a new (normally the next higher) **revision number** and recording the username, timestamp, “state” (a short symbol), and user-supplied “log message” (a textual description of the changes leading to that revision). It **uses diff** to find the differences between the tip of the default branch and the working file, thereby writing the minimal amount of information needed to be able to recreate the contents of the previous tip.

The “**checkout(co)**” operation identifies a specific revision from the RCS file and either displays the content to standard output or instantiates a working file, overwriting any current instantiation with the selected revision.

## *PLAYING WITH COMMANDS*

The basic user interface is extremely simple. The novice only needs to learn two commands: ci(1) and co(1).

ci, short for “check in”, deposits the contents of a file into an archival file called an RCS file. An RCS file contains all revisions of a particular file.

co, short for “check out”, retrieves revisions from an RCS file.

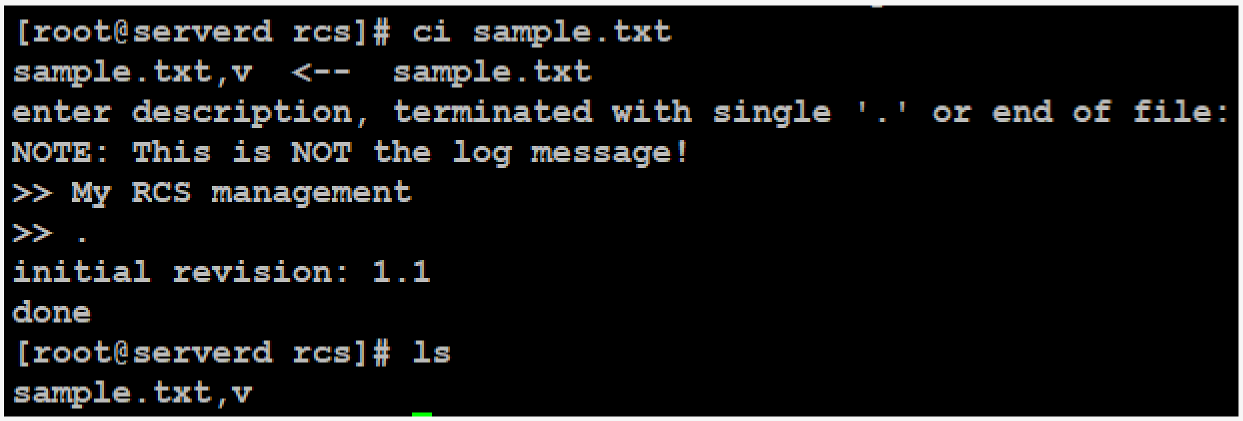
*Creating file to me managed by RCS*

```
[root@serverd ~]# mkdir rcs
[root@serverd ~]# echo "I am version 1" >> rcs/sample.txt
```

*Bringing file under RCS control*

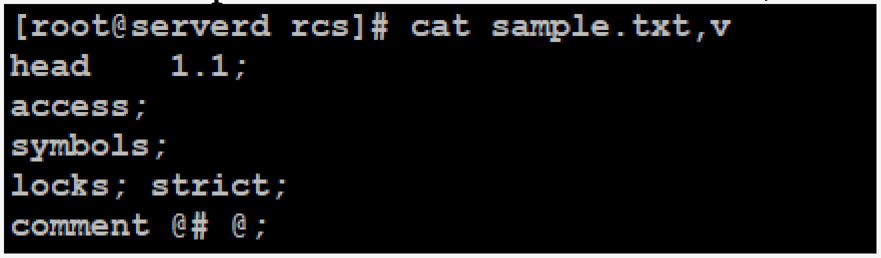
We first need to “**check the file in**” using the command “ci filename, and then add a description terminated with a “.” full stop.

```
# ci <filename>
```



```
[root@serverd rcs]# ci sample.txt
sample.txt,v  <--  sample.txt
enter description, terminated with single '.' or end of file:
NOTE: This is NOT the log message!
>> My RCS management
>> .
initial revision: 1.1
done
[root@serverd rcs]# ls
sample.txt,v
```

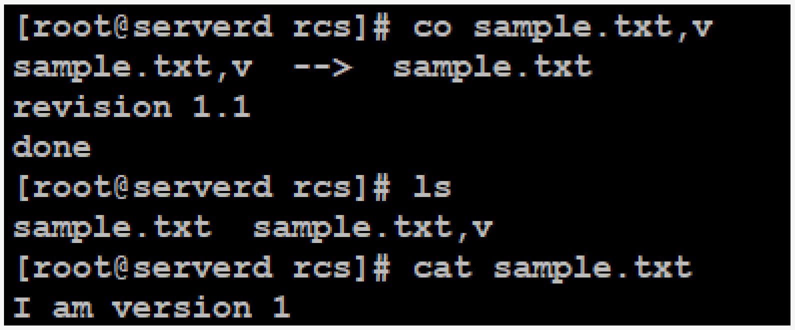
This command creates an RCS file in the RCS directory, stores sample.txt into it as revision 1.1, and deletes sample.txt



```
[root@serverd rcs]# cat sample.txt,v
head    1.1;
access;
symbols;
locks; strict;
comment @# @;
```

*Reading back the file*

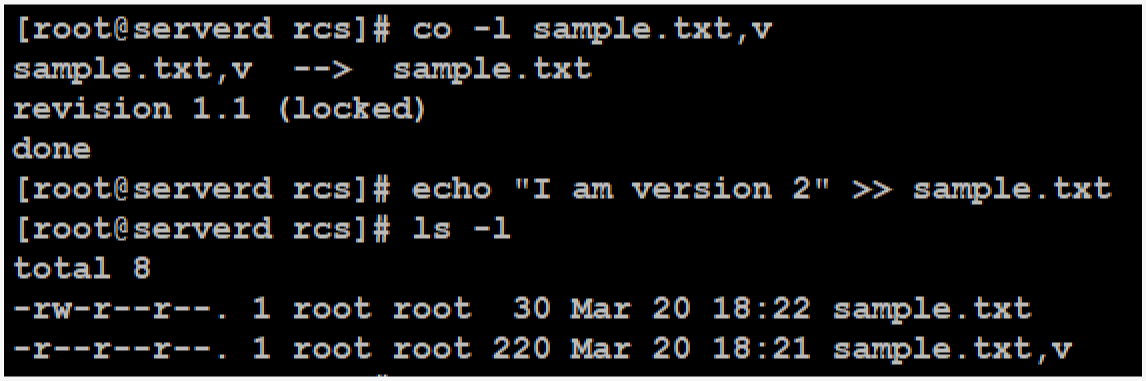
To return the file back so that it can be read, we need to "Check Out" the file: This command extracts the latest revision from the RCS file and writes it into.



```
[root@serverd rcs]# co sample.txt,v
sample.txt,v  -->  sample.txt
revision 1.1
done
[root@serverd rcs]# ls
sample.txt  sample.txt,v
[root@serverd rcs]# cat sample.txt
I am version 1
```

*Editing the file*

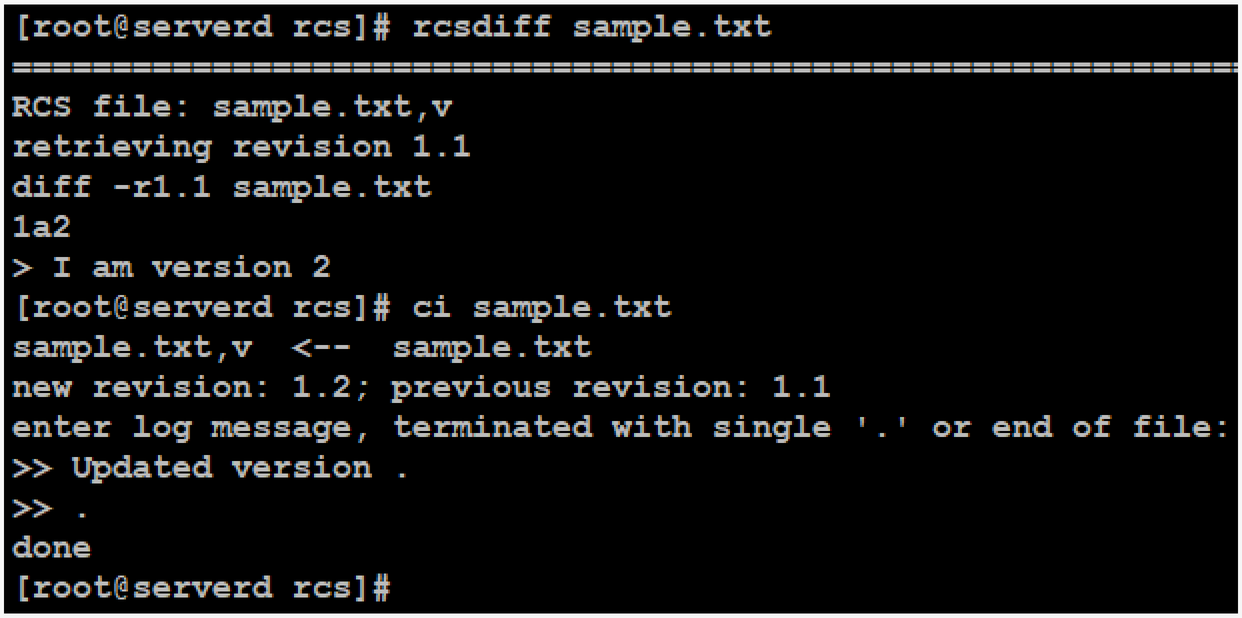
If you want to edit sample.txt, you must lock it as you check it out with the command



```
[root@serverd rcs]# co -l sample.txt,v
sample.txt,v  -->  sample.txt
revision 1.1 (locked)
done
[root@serverd rcs]# echo "I am version 2" >> sample.txt
[root@serverd rcs]# ls -l
total 8
-rw-r--r--. 1 root root  30 Mar 20 18:22 sample.txt
-r--r--r--. 1 root root 220 Mar 20 18:21 sample.txt,v
```

*Difference after editing file*

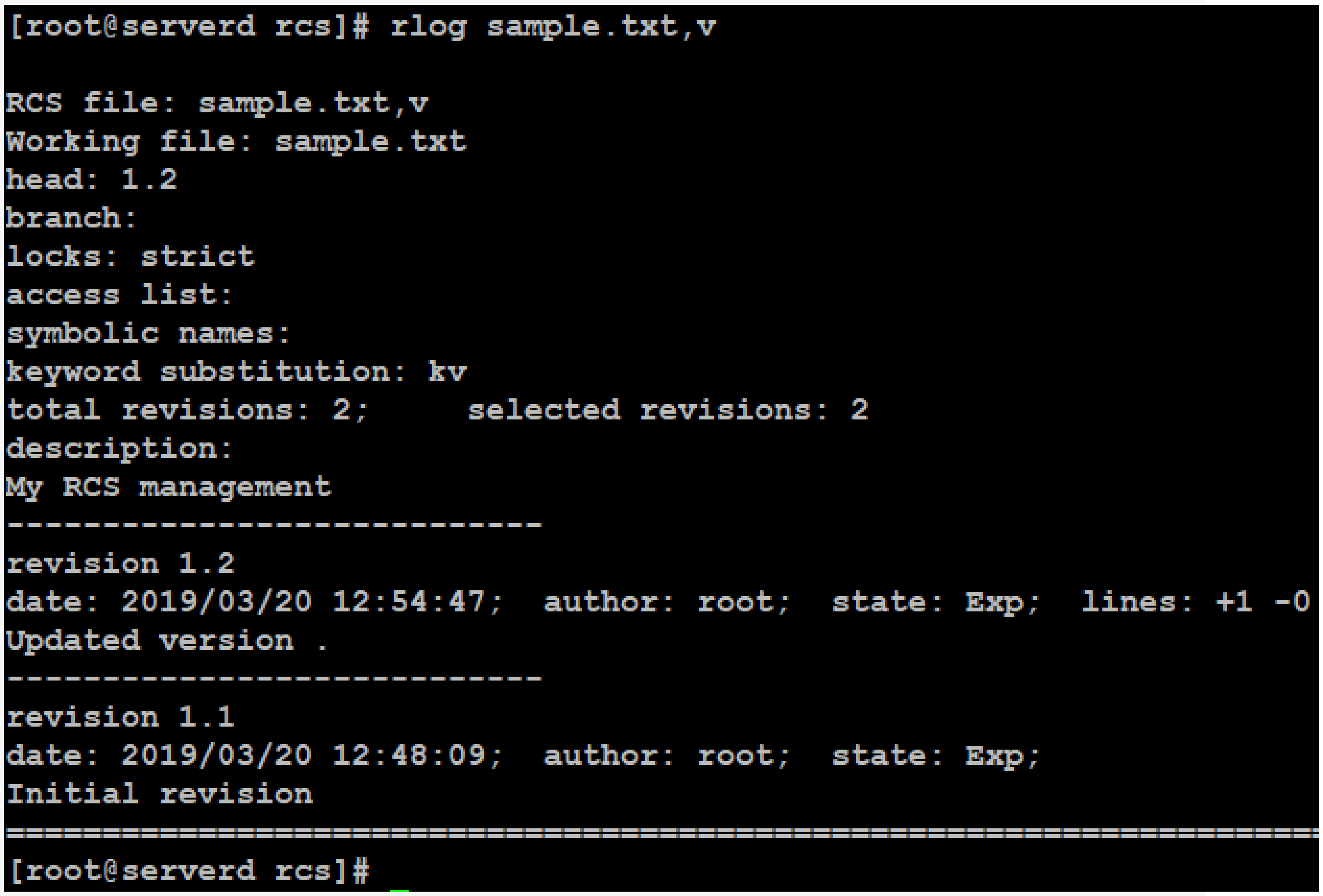
Suppose after some editing you want to know what changes that you have made. The command



```
[root@serverd rcs]# rcsdiff sample.txt
==============================================================
RCS file: sample.txt,v
retrieving revision 1.1
diff -r1.1 sample.txt
1a2
> I am version 2
[root@serverd rcs]# ci sample.txt
sample.txt,v  <--  sample.txt
new revision: 1.2; previous revision: 1.1
enter log message, terminated with single '.' or end of file:
>> Updated version .
>> .
done
[root@serverd rcs]#
```

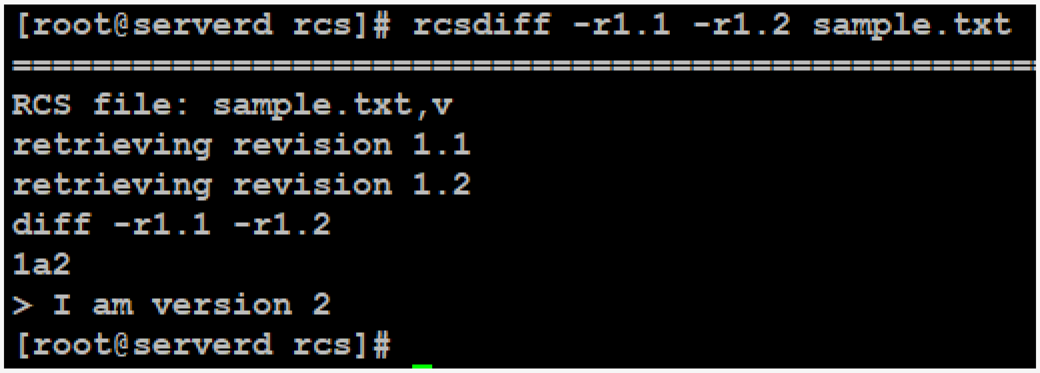
*Displaying info of RCS file*

To display information about a RCS owned file you can use the command “**rlog**“.

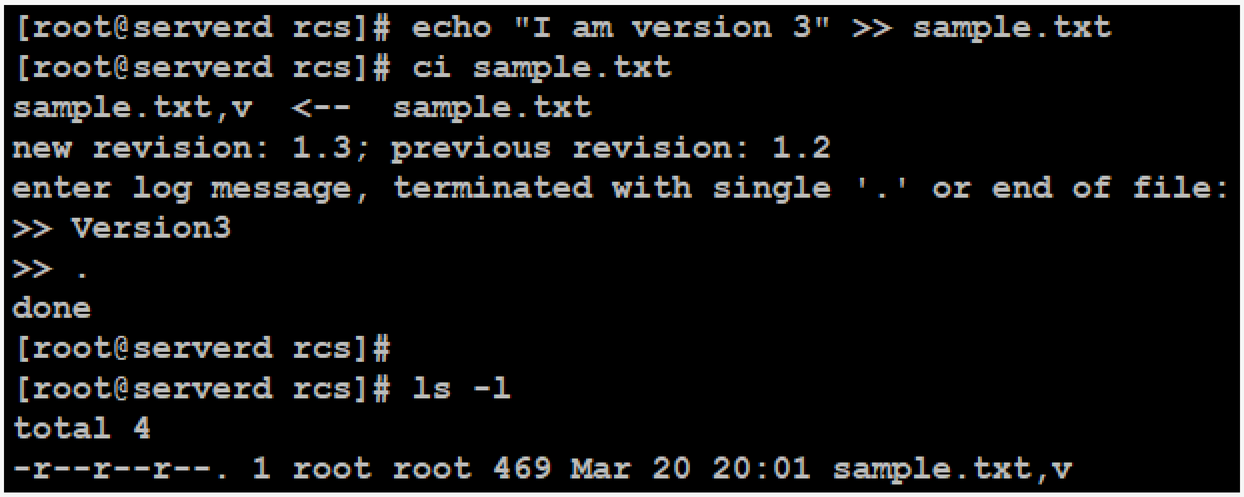

```
[root@serverd rcs]# rlog sample.txt,v

RCS file: sample.txt,v
Working file: sample.txt
head: 1.2
branch:
locks: strict
access list:
symbolic names:
keyword substitution: kv
total revisions: 2;     selected revisions: 2
description:
My RCS management
----------------------------
revision 1.2
date: 2019/03/20 12:54:47;  author: root;  state: Exp;  lines: +1 -0
Updated version .
----------------------------
revision 1.1
date: 2019/03/20 12:48:09;  author: root;  state: Exp;
Initial revision
=====================================================================
[root@serverd rcs]#
```

*Checking difference between different versions*

Now checking difference between various versions


```
[root@serverd rcs]# rcsdiff -r1.1 -r1.2 sample.txt
===================================================
RCS file: sample.txt,v
retrieving revision 1.1
retrieving revision 1.2
diff -r1.1 -r1.2
1a2
> I am version 2
[root@serverd rcs]#
```

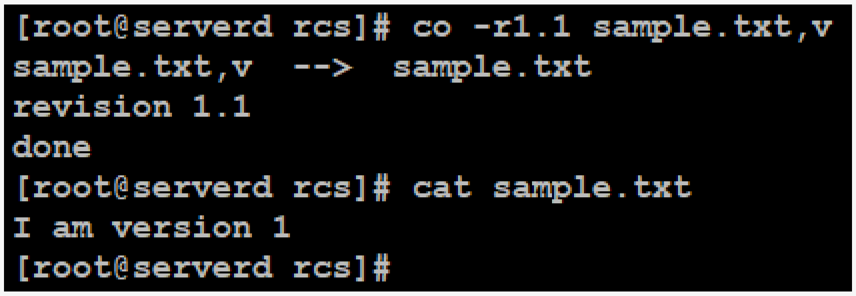
Now check out and append file with more details



```
[root@serverd rcs]# echo "I am version 3" >> sample.txt
[root@serverd rcs]# ci sample.txt
sample.txt,v  <--  sample.txt
new revision: 1.3; previous revision: 1.2
enter log message, terminated with single '.' or end of file:
>> Version3
>> .
done
[root@serverd rcs]#
[root@serverd rcs]# ls -l
total 4
-r--r--r--. 1 root root 469 Mar 20 20:01 sample.txt,v
```

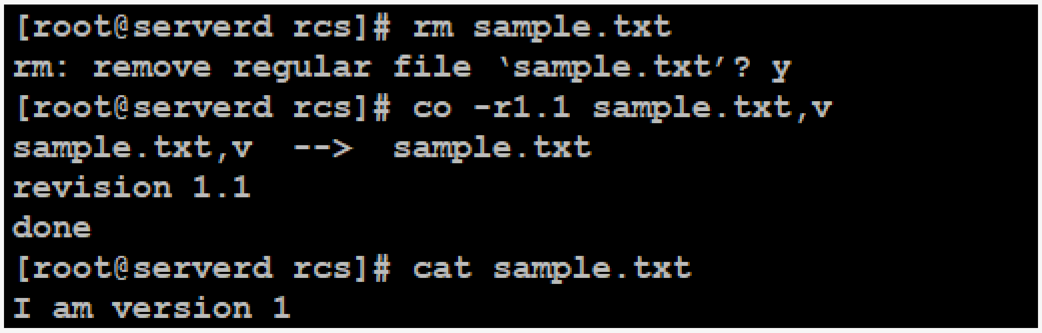
*Restoring back file to specific revision*

Restoring to last stable revision (eg 1.1)



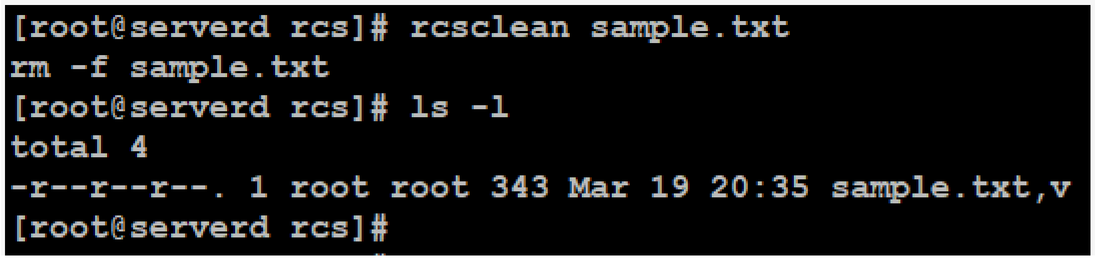
```
[root@serverd rcs]# co -r1.1 sample.txt,v
sample.txt,v  -->  sample.txt
revision 1.1
done
[root@serverd rcs]# cat sample.txt
I am version 1
[root@serverd rcs]#
```

*Recovering deleted file to last stable revision*



```
[root@serverd rcs]# rm sample.txt
rm: remove regular file ‘sample.txt’? y
[root@serverd rcs]# co -r1.1 sample.txt,v
sample.txt,v  -->  sample.txt
revision 1.1
done
[root@serverd rcs]# cat sample.txt
I am version 1
```

**rcsclean** removes files that are not being worked on. **rcsclean -u** also unlocks and removes files that are being worked on but have not changed.



```
[root@serverd rcs]# rcsclean sample.txt
rm -f sample.txt
[root@serverd rcs]# ls -l
total 4
-r--r--r--. 1 root root 343 Mar 19 20:35 sample.txt,v
[root@serverd rcs]#
```

**References:**

1. https://learningtechnix.wordpress.com/2019/03/22/linux-understanding-revision-control-system-rcs/
2. https://edutechwiki.unige.ch/en/Revision_control_system_tutorial
3. https://campus.barracuda.com/product/cloudgenfirewall/doc/96026542/how-to-configure-revision-control-system-monitoring-rcs/
4. https://zoomadmin.com/HowToInstall/UbuntuPackage/rcs
5. https://www.devmanuals.net/install/ubuntu/ubuntu-12-04-lts-precise-pangolin/install-rcs.html?expand_article=1
6. https://installati.one/install-rcs-ubuntu-20-04/?expand_article=1
7. https://www.gnu.org/software/rcs/

**Ex:2**
**Virtualization & Containerization to Install and Configure Docker for creating Containers of different Operating System Images**

Install using the apt repository
Before you install Docker Engine for the first time on a new host machine, you need to set up the Docker repository. Afterward, you can install and update Docker from the repository.

1. **Set up Docker's apt repository.**

**# Add Docker's official GPG key:**

```
$ sudo apt-get update
$ sudo apt-get install ca-certificates curl
$ sudo install -m 0755 -d /etc/apt/keyrings
$ sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc
$ sudo chmod a+r /etc/apt/keyrings/docker.asc
```

**# Add the repository to Apt sources:**

```
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc]
https://download.docker.com/linux/ubuntu \
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update
```

**Install the Docker packages.**
**To install the latest version, run:**

```
$ sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
```

**Verify that the Docker Engine installation is successful by running the hello-world image.**

```
$ sudo docker run hello-world
```

This command downloads a test image and runs it in a container. When the container runs, it prints a confirmation message and exits.

You have now successfully installed and started Docker Engine.

```
$sudo systemctl start docker
$sudo systemctl enable docker
$sudo systemctl status docker
```

**Pulling an Image:** To pull a Docker image from Docker Hub, you can use the **docker pull** command followed by the image name.

```
$docker pull <image_name>
```

Example: **docker pull ubuntu**

**Running a Container:** To run a container from an image, you can use the **docker run** command.

```
$docker run -it <image_name> /bin/bash
```

Example: **docker run -it ubuntu /bin/bash**

**Listing Running Containers:** To list all running containers, you can use the **docker ps** command.

```
$docker ps
```

**Listing All Containers:** To list all containers, including the ones that are not running, you can use the **docker ps -a** command.

```
$docker ps -a
```

Inside the container, execute some commands (e.g., **ls**, **uname -a**, etc.):

```
$ls
uname -a
```

**Stopping a Container:** To stop a running container, you can use the **docker stop** command followed by the container ID or name.

```
$docker stop <container_id or container_name>
```

**Exit the container:**

```
$ exit
```

**Optionally, you can stop and remove the container:**

```
$docker stop <container_id>
$docker rm <container_id>
```

**Removing a Container:** To remove a container, you can use the **docker rm** command followed by the container ID or name.

```
$docker rm <container_id or container_name>
```

**Removing an Image:** To remove a Docker image from your system, you can use the **docker rmi** command followed by the image ID or name.

```
$docker rmi <image_id or image_name>
```

## Ex:3 Virtualization & Containerization to Build, deploy and manage web or Java application on Docker

To create a java application and run it by using a container-based application. This example includes the following steps:

1. Create a directory by using below command.

```
$ mkdir  java-docker-application
```

2. Create a java class and save this as Sample.java file inside the directory, **java-docker-application as Sample.java**.

**//Sample.java**

```
class Sample{
public static void main(String[] args)  {
System.out.println("This is java application by using docker
");
}
}
```

3. Create a Dockerfile

The dockerfile does not contain any file extension. Please save it simply with a Dockerfile name.

**// Dockerfile**

```
FROM java:8
COPY . /var/www/java
WORKDIR /var/www/java
RUN javac Sample.java
CMD ["java", "Sample"]
```

Your folder inside must look like the below.

4. Building a Docker Image

Now, create an image by following the command below. In order to create an image, we must log in as root. In this example, we have switched to a root user. In the following command, **java-application** is the name of the image. We can have any name for our docker image.

```
$ docker build -t java-application .
```

The output is shown below:

```
[root@xyz java-docker-app]# docker build -t java-application
.
Sending build context to Docker daemon 3.072kB
```

```
Step 1/5 : FROM java:8
---> d23bdf5b1b1b
Step 2/5 : COPY . /var/www/java
---> Using cache
---> c34d8257c2c6
Step 3/5 : WORKDIR /var/www/java
---> Using cache
---> 1c620a43fe4e
Step 4/5 : RUN javac Sample.java
---> Using cache
---> b0d9d44eead7
Step 5/5 : CMD java Sample
---> Using cache
---> 474c196d8cc1
Successfully built 474c196d8cc1
Successfully tagged java-application:latest
```

After making a successful image build, we are now ready to run our docker image in the next step.

5. Run Docker Image

Now we are going to run docker by using the following run command.

```
$ docker run java-application
```

The output from the above command is as shown.

```
[root@xyz java-docker-app]# docker run java-application
This is java application
by using Docker
```

Now, the docker image ran successfully. Apart from all these, you can also use other commands as well.

The docker containerized environment is a software platform, which provides an abstraction of operating-system-level virtualization. To know more you can read this article: https://docs.docker.com/engine/docker-overview/

You need an available container-based application. Check the documentation for the right procedure for installation for your OS: https://docs.docker.com/engine/getstarted/step_one.

**References:**

https://developers.redhat.com/blog/2017/11/17/deploy-java-application-docker
https://docs.docker.com/get-started/overview/
https://docs.docker.com/get-started/
https://docs.docker.com/get-started/02_our_app/
https://www.simplilearn.com/tutorials/docker-tutorial/how-to-install-docker-on-ubuntu

# Software Configuration Management and provisioning using Chef

**Installing a Chef Server Workstation on Ubuntu**

```
sudo apt update && sudo apt upgrade
```

## The Chef Server

### Install the Chef Server

Download the [latest Chef server core](https://packages.chef.io/files/stable/chef-server/13.1.13/ubuntu/18.04/chef-server-core_13.1.13-1_amd64.deb):

```
wget https://packages.chef.io/files/stable/chef-server/13.1.13/ubuntu/18.04/chef-server-core_13.1.13-1_amd64.deb
```

1. Install the server:

```
2. sudo dpkg -i chef-server-core_*.deb
```

3. Remove the downloaded file:

```
4. rm chef-server-core_*.deb
```

5. The Chef server includes a command line utility called `chef-server-ctl`. Run `chef-server-ctl` to start the Chef server services:

```
6. sudo chef-server-ctl reconfigure
```

### Create a Chef User and Organization

In order to link workstations and nodes to the Chef server, create an administrator and organization with associated RSA private keys.

1. From the home directory, create a `.chef` directory to store the keys:

```
2. mkdir .chef
```

3. Use `chef-server-ctl` to create a user. In this example, change the following to match your needs: `USER_NAME`, `FIRST_NAME`, `LAST_NAME`, `EMAIL`, and `PASSWORD`. Adjust `USER_NAME.pem`, and leave the `.pem` extension:

```
4. sudo chef-server-ctl user-create USER_NAME FIRST_NAME
   LAST_NAME EMAIL 'PASSWORD' --filename ~/.chef/USER_NAME.pem
```

To view a list of all users on your Chef server issue the following command:

```
sudo chef-server-ctl user-list
```

5. Create an organization and add the user created in the previous step to the admins and billing admins security groups. Replace `ORG_NAME` with a short identifier for the organization, `ORG_FULL_NAME` with the organizations' complete name, `USER_NAME` with the username created in the step above and `ORG_NAME.pem` with organization's short identifier followed by `.pem`:

```
6. sudo chef-server-ctl org-create ORG_NAME "ORG_FULL_NAME" --
   association_user USER_NAME --filename ~/.chef/ORG_NAME.pem
```

Note
`ORG_NAME` must be in all lower case.

To view a list of all organizations on your Chef server, use the following command:

```
sudo chef-server-ctl org-list
```

With the Chef server installed and the RSA keys generated, you can begin configuring your workstation. The workstation is where all major configurations will be created for your nodes.

**Chef Workstations**

The Chef workstation is where you create and configure any recipes, cookbooks, attributes, and other changes necessary to manage your nodes. Although this can be a local machine running any OS, there is some benefit to keeping a remote server as your workstation so you can access it from anywhere.

In this section, you will download and install the Chef Workstation package, which provides all tools also included with the ChefDK, Chef's development kit.

**Setting Up a Workstation**

1. Download the latest Chef Workstation:

```
2. wget  https://packages.chef.io/files/stable/chef-
   workstation/0.2.43/ubuntu/18.04/chef-workstation_0.2.43-
   1_amd64.deb
```

3. Install Chef Workstation:

```
4. sudo dpkg -i chef-workstation_*.deb
```

5. Remove the installation file:

```
6. rm chef-workstation_*.deb
```

7. Create your Chef repository. The `chef-repo` directory will store your Chef cookbooks and other related files.

```
8.  chef generate repo chef-repo
```

9. Ensure that your workstation's `/etc/hosts` file maps its IP address to your Chef server's fully qualified domain name and workstation hostnames. For example:

File: /etc/hosts

```
127.0.0.1 localhost
192.0.1.0 example.com
192.0.2.0 workstation
...

```

10. Create a `.chef` subdirectory. The `.chef` subdirectory will store your Knife configuration file and your `.pem` files that are used for RSA key pair authentication with the Chef server. Move into the `chef-repo` directory:

```
11.  mkdir ~/chef-repo/.chef
12.  cd chef-repo
```

### Add the RSA Private Keys

Authentication between the Chef server and workstation and/or nodes is completed with public key encryption. This ensures that the Chef server only communicates with trusted machines. In this section, the RSA private keys, generated when setting up the Chef server, will be copied to the workstation to allow communicate between the Chef server and workstation.

1. If you do not already have an RSA key-pair on your workstation, generate one. This key-pair will be used to gain access to the Chef server and then transfer their `.pem` files:

```
2.  ssh-keygen -b 4096
```

   Press **Enter** to use the default names `id_rsa` and `id_rsa.pub` in `/home/your_username/.ssh` before entering your passphrase.
   Note
   If you have disabled SSH password authentication on your Chef server's Linode, as recommended by the How to Secure Your Server guide, re-enable SSH password authentication prior to performing these steps. Be sure to disable it again once you have added your workstation's public ssh key to the Chef server's Linode.
   Upload your workstation's public key to the Linode hosting the Chef server. Ensure you replace `example_user` with the Chef server's user account and `192.0.2.0` with its IP address:

```
 ssh-copy-id example_user@192.0.2.0
```

3. Copy the `.pem` files from your Chef server to your workstation using the `scp` command. Replace `user` with the appropriate username, and `192.0.2.0` with your Chef server's IP:

```
4.  scp example_user@192.0.2.0:~/.chef/*.pem ~/chef-repo/.chef/
```

5. Confirm that the files have been copied successfully by listing the contents of the `.chef` directory:

```
6. ls ~/chef-repo/.chef
```

   Your `.pem` files should be listed.

### Add Version Control

The workstation is used to create, download, and edit cookbooks and other related files. You should track any changes made to these files with version control software, like Git. The Chef Workstation adds the Git component to your workstation and initializes a Git repository in the directory where the `chef-repo` was generated. Configure Git by adding your username and email, and add and commit any new files created in the steps above.

1. Configure Git by adding your username and email, replacing the needed values:

```
2. git config --global user.name yourname
3. git config --global user.email user@email.com
```

4. Add the `.chef` directory to the `.gitignore` file:

```
5. echo ".chef" > ~/chef-repo/.gitignore
```

6. Move into the `~/chef-repo` directory, if you are not already there and add and commit all existing files:

```
7. cd ~/chef-repo
8. git add .
9. git commit -m "initial commit"
```

10. Make sure the directory is clean:

```
11. git status
```

It should output:

```
On branch master
nothing to commit, working directory clean
```

**Generate your First Cookbook**

1. Generate a new Chef cookbook:

```
2.  chef generate cookbook my_cookbook
```

**Configure Knife**

1. Create a knife configuration file by navigating to your `~/chef-repo/.chef` directory and creating a file named `config.rb` using your preferred text editor.
2. Copy the following configuration into the `config.rb` file:

File: ~/chef-repo/.chef/config.rb

```
current_dir = File.dirname(__FILE__)
log_level                 :info
log_location              STDOUT
node_name                 'node_name'
client_key                "USER.pem"
validation_client_name    'ORG_NAME-validator'
validation_key            "ORGANIZATION-validator.pem"
chef_server_url
'https://example.com/organizations/ORG_NAME'
cache_type                'BasicFile'
cache_options( :path => "#{ENV['HOME']}/.chef/checksums" )
cookbook_path             ["#{current_dir}/../cookbooks"]
```

3. Change the following:

- The value for `node_name` should be the username that was created on the Chef server.
- Change `USER.pem` under `client_key` to reflect your `.pem` file for your user.
- The `validation_client_name` should be your organization's `ORG_NAME` followed by `-validator`.
- `ORGANIZATION-validator.pem` in the `validation_key` path should be set to the `ORG_NAME` followed by `-validator.pem`.

- Finally the `chef_server_url` should be the Chef server's domain with `/organizations/ORG_NAME` appended. Be sure to replace `ORG_NAME` with your own organization's name.

4. Move to the `chef-repo` directory and copy the needed SSL certificates from the server:

```
5. cd ..
6. knife ssl fetch
```

Note
The SSL certificates are generated during the installation of the Chef server. These certificates are self-signed, which means there isn't a signing certificate authority (CA) to verify. The Chef server's hostname and FQDN should be the same so that the workstation can fetch and verify the SSL certificates. You can verify the Chef server's hostname and FQDN by running `hostname` and `hostname -f`, respectively. Consult the Chef documentation for details on regenerating SSL certificates.

7. Confirm that `config.rb` is set up correctly by running the client list:

```
8. knife client list
```

This command should output the validator name.

Now that your Chef server and workstation are configured, you can bootstrap your first node.

**Bootstrap a Node**

Bootstrapping a node installs the Chef client on the node and validates the node. This allows the node to read from the Chef server and pull down and apply any needed configuration updates detected by the chef-client.

Note
If you encounter any `401 Unauthorized` errors ensure that your `ORGANIZATION.pem` file has `700` permissions. See Chef's troubleshooting guide for further information on diagnosing authentication errors.

1. Update the `/etc/hosts` file on the *node* to identify the node, Chef server's domain name, and the workstation.

File: /etc/hosts

```
127.0.0.1 localhost
198.51.100.0 node-hostname
192.0.2.0 workstation
192.0.1.0 example.com
...

```

2. From your *workstation*, navigate to your `~/chef-repo/.chef` directory:

```
3.  cd ~/chef-repo/.chef
```

4. Bootstrap the client node either using the client node's root user, or a user with elevated privileges:

- **As the node's root user**, change `password` to your root password and `nodename` to the desired name for your client node. You can leave this off if you would like the name to default to your node's hostname:

```
•   knife bootstrap 192.0.2.0 -x root -P password --node-name nodename
```

- **As a user with sudo privileges**, change `username` to a node user, `password` to the user's password and `nodename` to the desired name for the client node. You can leave this off if you would like the name to default to your node's hostname:

-
  ```
  knife bootstrap 192.0.2.0 -x username -P password --use-sudo-password --node-name nodename
  ```

- **As a user with key-pair authentication**, change `username` to a node user, and `nodename` to the desired name for the client node. You can leave this off if you would like the name to default to your client node's hostname:

-
  ```
  knife bootstrap 192.0.2.0 --ssh-user username --sudo --identity-file ~/.ssh/id_rsa.pub --node-name hostname
  ```

5. Confirm that the node has been bootstrapped by listing the client nodes:

```
6. knife client list
```

Your new client node should be included in the list.

7. Add the bootstrapped node to your workstation's `/etc/hosts` file. Replace `node-hostname` with the hostname you just assigned to the node when it was bootstrapped:

File: /etc/hosts

```
127.0.0.1 localhost
192.0.1.0 example.com
192.0.2.0 workstation
198.51.100.0 node-hostname
...

```

## Download a Cookbook (Optional)

When using Chef, the Chef client should periodically run on your nodes and pull down any changes pushed to the Chef server from your workstation. You will also want the `validation.pem` file that is uploaded to your node upon bootstrap to be deleted for security purposes. While these steps can be performed manually, it is often easier and more efficient to set them up as a cookbook.

This section is optional, but provides instructions on downloading a cookbook to your workstation and pushing it to a server, and includes the skeleton of a basic cookbook to expand on and experiment with.

1. From your *workstation*, navigate to your `~/chef-repo/.chef` directory:

```
2.  cd ~/chef-repo/.chef
```

3. Download the cookbook and dependencies:

```
4. knife cookbook site install cron-delvalidate
```

5. Open the `default.rb` file to examine the default cookbook recipe:

File: ~/chef-repo/cookbooks/cron-delvalidate/recipes/default.rb

```
#
# Cookbook Name:: cron-delvalidate

```

```
# Recipe:: Chef-Client Cron & Delete
Validation.pem
#
#

cron "clientrun" do
  minute '0'
  hour '*/1'
  command "/usr/bin/chef-client"
  action :create
end

file "/etc/chef/validation.pem" do
  action :delete
end
```

The resource `cron "clientrun" do` defines the cron action. It is set to run the `chef-client` action (`/usr/bin/chef-client`) every hour (`*/1` with the `*/` defining that it's every hour and not 1AM daily). The `action` code denotes that Chef is *creating* a new cronjob.
`file "/etc/chef/validation.pem" do` calls to the `validation.pem` file.
The `action` defines that the file should be removed (`:delete`).

These are two very basic sets of code written in Ruby that provide an example of the code structure that will be used when creating Chef cookbooks. These examples can be edited and expanded as needed.

6. Add the recipe to your node's run list, replacing `nodename` with your node's name:

```
7. knife node run_list add nodename 'recipe[cron-
   delvalidate::default]'
```

8. Push the cookbook to the Chef server:

```
9. knife cookbook upload cron-delvalidate
```

This command is also used when updating cookbooks.

10. Use `knife-ssh` to run the `chef-client` command on your node.
    Replace `nodename` with your node's name. If you have set up your node with a limited user account, replace `-x root` with the correct username, i.e. `-x username`.

```
11.  knife ssh 'name:nodename' 'sudo chef-client' -x root
```

The recipes in the run list will be pulled from the server and run on the node. In this instance, it will be the `cron-delvalidate` recipe. This recipe ensures that any cookbooks pushed to the Chef Server, and added to the node's run list will be pulled down to bootstrapped nodes once an hour. This automated step eliminates connecting to the node in the future to pull down changes.

**References:**

https://www.linode.com/docs/guides/install-a-chef-server-workstation-on-ubuntu-18-04/
https://www.digitalocean.com/community/tutorials/how-to-set-up-a-chef-12-configuration-management-system-on-ubuntu-14-04-servers
https://docs.chef.io/workstation/getting_started/
https://confluence.slac.stanford.edu/display/SCSPub/Chef+Configuration+Management
https://www.edureka.co/blog/install-chef/