2014

Hands On Lab

Enterprise Backup with Rsync

Table of Contents

Introduction

Rsync is a common method for backing up and/or synchronizing filesystems across multiple systems or servers. It literally stands for "remote synchronization". It uses algorithms that minimize the amount of data copied from job to job by only moving new files or new portions of files (changes only).

Goals

This lab will introduce you to the concepts of using Rsync to make backups. We will explore how to synchronize files and directories on a local system as well as how to synchronize and backup systems remotely.

Packages, Resources and Prerequisites

The packages involved in our lab are:

- rsync

The resources you will be accessing during the course of this lab are:

- Ubuntu 14.04 Server
  - You will use this to test your enterprise backup using the rsync with CRON
- Ubuntu 14.04 Client
  - This is the system that we will be backing up, first locally with scripts that synchronize directories and then by connecting to a server and executing a push with CRON

Prerequisites to this lab:

- A LinuxAcademy.com Lab+ Subscription
- Internet Access and SSH Client
  - You will need to connect to the public IP of the servers in order to complete this lab
    - SSH client can be Windows (i.e. Putty) or from another Linux system shell
- Login Information (Provided When The Server Starts Up)

Document Conventions

Just a couple of housekeeping items to go over so you can get the most out of this Hands On Lab without worrying about how to interpret the contents of the document.

When we are demonstrating a command that you are going to type in your shell while connected to a server, you will see a box with a dark blue background and some text, like this:

```
linuxacademy@ip-10-0-0-0:~$ sudo apt-get install package
[sudo] password for linuxacademy: <PASSWORD PROVIDED>
```

That breaks down as follows:

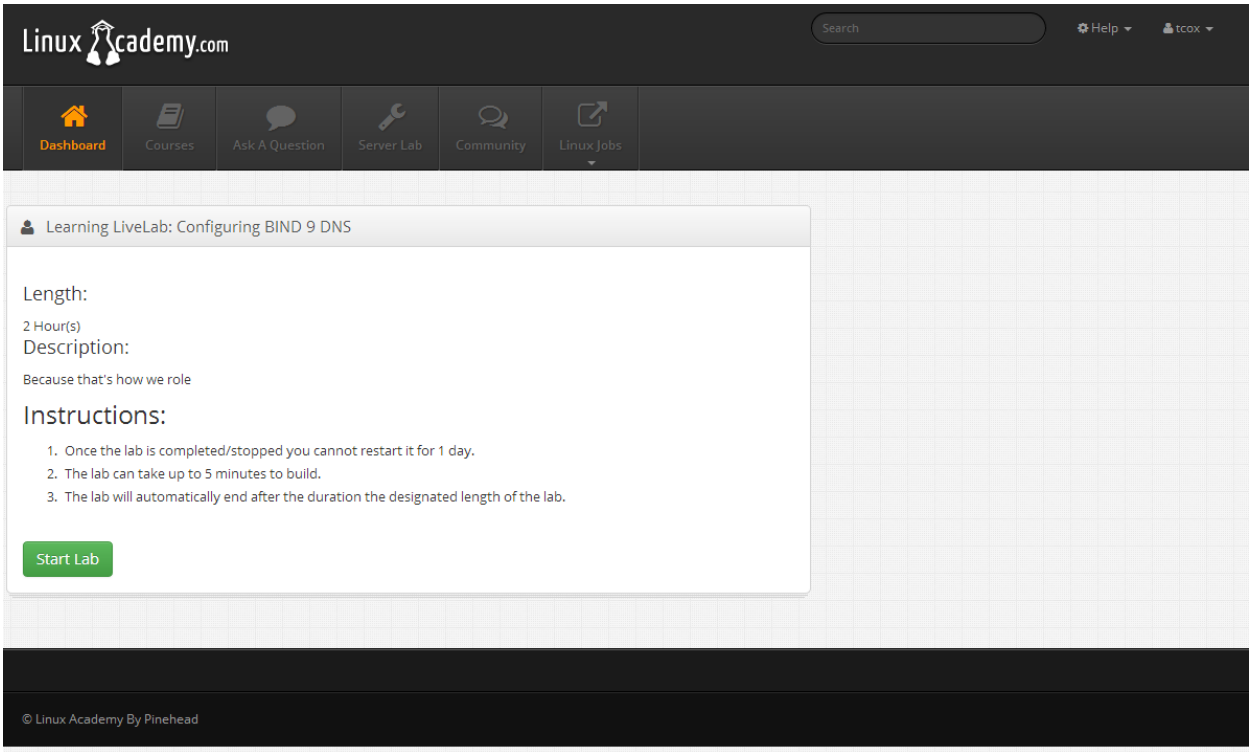
- The white text that looks something like "linuxacademy@ip-10-0-0-0:~$: "should be interpreted as the console prompt your cursor will be sitting at when you are logged into the server. You are not typing that into the console, it is just there. Note that the portion that says "ip-10-0-0-0" will be different for you based on the IP address of your system.
- The bold yellow text in any command example is the actual command you will type in your shell.
- Any lines subsequent to the bold yellow command that you type in is to be considered as the response you can expect from your command entry.

If you do not see the command prompt as the first line in the example, then all the white text is an example of a text, script or configuration file and is intended to be typed in its entirety (in the event you are instructed to create a previously non-existent file) or to compare against the contents of the file on your system.

One final convention, if you see a "~" at the end of a line in a command or text example, that will indicate that the line overflowed the end of line. Meaning you should just keep typing without hitting enter to start a new line until the natural end of the command or entry.

General Process

When you are ready to begin the Hands On Lab, log into your Linux Academy Lab+ subscription and navigate to the "Live Labs" section on the Dashboard. Once you choose the "Enterprise Backup with Rsync" Lab from the list, you will see a similar screen to the one below:

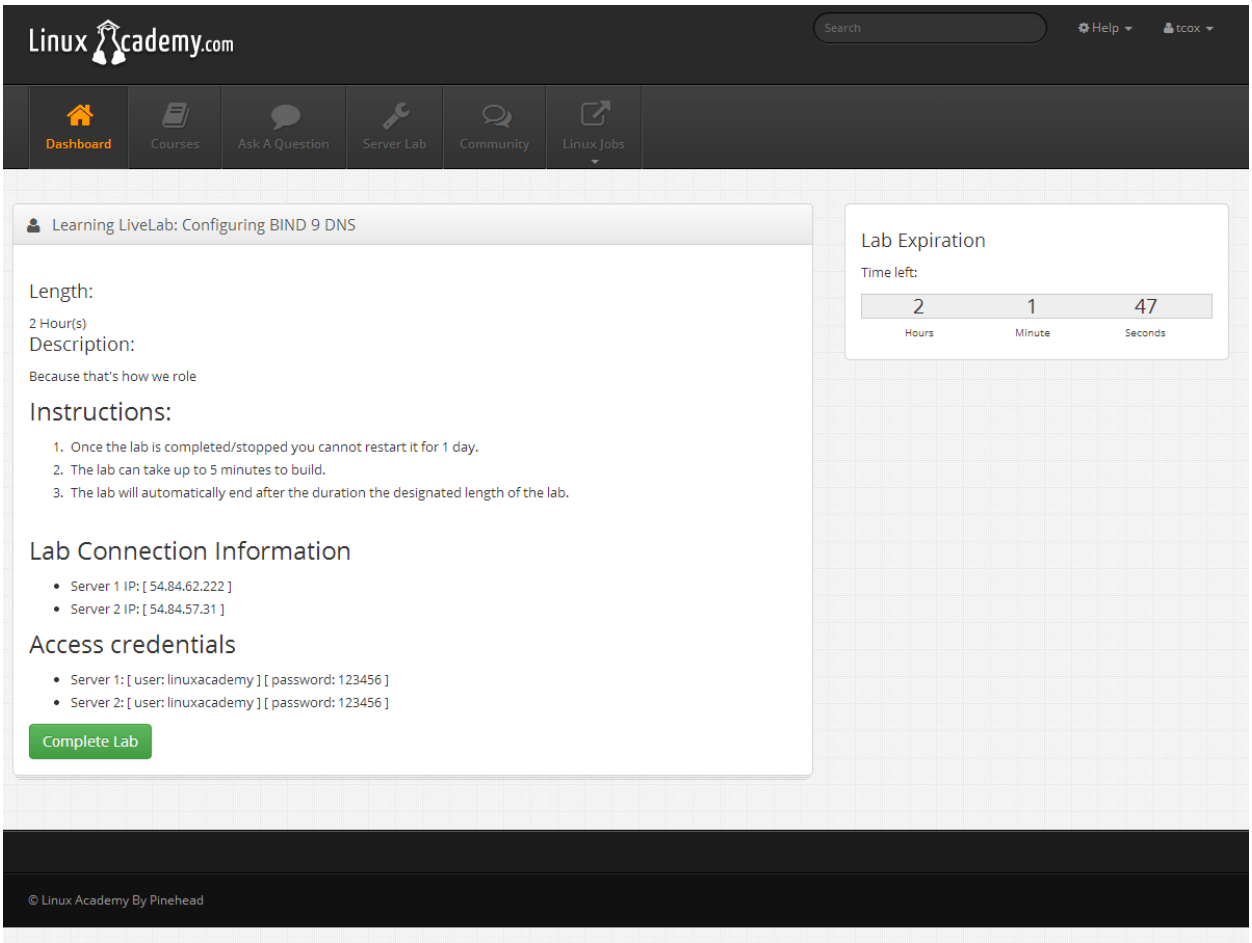


[Linux Academy Live Labs+ - Start Screen]

A few things to note before you start this process:

- When you launch the lab from this screen, it may take up to FIVE MINUTES for your servers to be deployed and be available for your use.
- Do not leave your desktop and come back, once the servers are launched, you will only have ONE HOUR to complete this lab from start to finish. After that point, the servers time out and are deleted permanently. Any and all work that you have done will then be lost.
- You can only use this lab as many times as you want per day as long as you maintain your Linux Academy subscription.

Once you have clicked on the 'Start Lab' button that you see above, a process will launch on our servers that will deploy the two servers we will use in our lab for testing. After a few minutes of processing (and you will see a status message that says "Creating Lab… Please Wait"), you should see a screen that looks similar to this one:



[Linux Academy Live Labs+ - Start Screen]

You will see all the information you need to access your servers from another system. Specifically, you need:

- The two server public IP addresses
- Access credentials for both

One thing to note is that, in addition to the two IPs that you see above, each server will have another IP assigned to it in the 10.0.0.x subnet. This is a private IP address and will not route outside of your private server pool.

In our setup, Server 1 will function as an local server that we are going to perform local directory to directory backups on. Server 2 will function as our remote backup server that we will be pushing our backups to both manually and by executing scripts through CRON.

Rsync Installation

Unlike many of our labs, even in a minimal installation, rsync should be installed (regardless of distribution) by default. You can be sure rsync is installed by doing the following:

```
linuxacademy@ip-10-0-0-162:~$ which rsync
/usr/bin/rsync
```

As long as you see the path and the rsync binary as a result of the above command, you have it installed on your system. If, for whatever reason, you do not see that result, execute:

```
linuxacademy@ip-10-0-0-162:~$ sudo apt-get install rsync
```

And you will be ready to go.

Using Rsync

Despite the fact that Rsync is a power backup tool, it can be extremely simple to use as well. Let's create our own backup scenario using a couple of directories that we want to copy the contents of from one to the other. While in your home directory, execute the following commands:

```
linuxacademy@ip-10-0-0-162:~$ mkdir  bkup1
linuxacademy@ip-10-0-0-162:~$ mkdir  bkup2
linuxacademy@ip-10-0-0-162:~$ touch bkup1/myfile{1..75}
linuxacademy@ip-10-0-0-162:~/bkup1$ cd bkup1
linuxacademy@ip-10-0-0-162:~/bkup1$ ls
myfile1   myfile21  myfile33  myfile45  myfile57  myfile69
myfile10  myfile22  myfile34  myfile46  myfile58  myfile7
myfile11  myfile23  myfile35  myfile47  myfile59  myfile70
myfile12  myfile24  myfile36  myfile48  myfile6   myfile71
myfile13  myfile25  myfile37  myfile49  myfile60  myfile72
myfile14  myfile26  myfile38  myfile5   myfile61  myfile73
myfile15  myfile27  myfile39  myfile50  myfile62  myfile74
myfile16  myfile28  myfile4   myfile51  myfile63  myfile75
myfile17  myfile29  myfile40  myfile52  myfile64  myfile8
myfile18  myfile3   myfile41  myfile53  myfile65  myfile9
myfile19  myfile30  myfile42  myfile54  myfile66
myfile2   myfile31  myfile43  myfile55  myfile67
myfile20  myfile32  myfile44  myfile56  myfile68
```

As you can see, we now have files in the bkup1/ directory, seventy-five empty files to be exact. So let's start by using Rsync to work with these files across the two directories that we created earlier.

Directory Copying

Now that we have some files to work with, let's do a quick synchronization between the directory containing our seventy-five empty files and the empty directory. As we said earlier, rsync is both powerful and simple. We can synchronize the contents of bkup1/ and bkup2/ from our home directory as follows:

```
linuxacademy@ip-10-0-0-162:~$ rsync -r bkup1/ bkup2
```

That's all there is to it, although there are a couple of things you need to keep in mind. The '-r' switch tells the rsync command to be sure to sync the contents recursively, so if there are any subdirectories in the indicated path, be sure to include them. Let's see what happens with links in our directory. Execute the following command from the bkup1/ directory:

```
linuxacademy@ip-10-0-0-162:~/bkup1$ ln -s myfile75 myfile100
```

All we did here was to create a soft link to the 'myfile75' file from the 'myfile100' file. So what happens if we run our rsync command again? Let's find out:

```
linuxacademy@ip-10-0-0-162:~$ rsync -r bkup1/ bkup2
skipping non-regular file "myfile100"
```

The message we get indicates that the soft linked file we just created was skipped during the synchronization of the two directories. Fortunately, we can fix that easily enough with the following command instead:

```
linuxacademy@ip-10-0-0-162:~$ rsync -a bkup1/ bkup2
```

If you list the directory contents of bkup2/ now, you will see that the soft linked file "myfile100" now exists and points to "myfile75" just as we would expect. All we did was change from the recursive regular file command line switch to what's called a combination flag. The '-a' flag stands for "archive" and tells rsync to synchronize both recursively as well as preserve symbolic links, device files, special files and file modification and ownership information (date, group, permissions).

One final note in this process is the trailing '/' in our command line above. This tells rsync to synchronize the contents of the bkup1/ directory only. If you omit that trailing slash, you would end up with a directory called "bkup1" inside of directory "bkup2".

Better Safe Than Sorry

Rsync provides a method to test what you are going to do without actually executing the command (a very useful thing as you write more and more complex scripts and need to test them

out without ruining anything). This is called a "dry run" and is executed as follows (using our example):

```
linuxacademy@ip-10-0-0-162:~$ rsync -anv bkup1/ bkup2
sending incremental file list
./
myfile1
myfile10
myfile100 -> myfile75
myfile11
myfile12
myfile13
myfile14
myfile15
myfile16
myfile17
myfile18
myfile19
... (etc)
```

As you can see, this provides you with a method that shows you what your rsync command will do. This is increasingly important as you create the scripts you will use for backups, especially running them on remote systems as we will see below.

Remote System Backups

In addition to the typical "synchronize these files systems or directories" locally, rsync can operate across multiple systems the same as it does locally. Let's prepare our lab systems to demonstrate this capability.

SSH Key Exchange for Remote Access

Since we want to be able to script synchronization between servers, we need to exchange some SSH keys between them so traffic will be trusted without a password. Although this process is covered in great detail in other Linux Academy labs and courses, here is a quick walk through of the process

On the server we have been working on, execute the following commands:

```
linuxacademy@ip-10-0-0-162:~$ ssh-keygen
<once you see the key is created after following prompts…>
linuxacademy@ip-10-0-0-162:~$ ssh-copy-id linuxacademy@10.0.0.114
linuxacademy@ip-10-0-0-162:~$ ssh 10.0.0.114
linuxacademy@ip-10-0-0-114:~$ <you did not have to enter your pwd>
```

This will allow your "linuxacademy" user to log in as the "linuxacademy" user to your second system in this lab (keep in mind your IP addresses will be different, see your console when you started up the lab for the correct IPs) without entering a password. Now, let's be sure to run this in reverse so the second server can log into the first as well:

```
linuxacademy@ip-10-0-0-114:~$ ssh-keygen
<once you see the key is created after following prompts…>
linuxacademy@ip-10-0-0-114:~$ ssh-copy-id linuxacademy@10.0.0.162
linuxacademy@ip-10-0-0-114:~$ ssh 10.0.0.114
linuxacademy@ip-10-0-0-162:~$ <you did not have to enter your pwd>
```

Now we have the ability to move back and forth amongst our lab servers without a password. You would do something similar in a production environment so that the hosts you are backing up or synchronizing can trust each other without a password (which is difficult and insecure to be providing inside your scripts).

Sending a Backup to Remote System (Rsync Push)

On your second lab server, in your home directory, please create the bkup1/ and bkup2/ directories exactly the same way we did in the process above on our first system. Once you have that done, we want to take the contents from our local filesystem directories bkup1/ and bkup2/ and synchronize with their remote partners.

At this point, we are going to take the contents of our bkup1/ directory and "push" them to the remote directory of the same name. Again, because rsync is both powerful and simple to use, the syntax for doing that is almost exactly the same:

```
linuxacademy@ip-10-0-0-162:~$ rsync -a bkup1/
10.0.0.114:/home/linuxacademy/bkup1
```

Now if you log into the remote server at address 10.0.0.114 (again, your IP will be different based on the IPs assigned to you and indicated on your Linux Academy browser when you started the lab, please substitute as appropriate), you will see the contents of the local bkup1/ directory exactly replicated on the remote server's bkup1/ directory.

Requesting a Backup from Remote System (Rsync Pull)

Connect to your second server and log in. Now, we are going to execute what is called an rsync "pull". We will cause the remote system's bkup1/ directory to synchronize locally by "pulling" the contents over. Again, the syntax is very simple and is effectively a reverse of the push command (like copying directories around, you indicate what you are copying to where):

```
linuxacademy@ip-10-0-0-114:~$ rsync -a
10.0.0.162:/home/linuxacademy/bkup1 bkup1/
```

This tells our rsync application to go get those contents and pull them over here. Much like most other file management tools (cp, mv, etc), your source location is first, followed by the intended destination.

Enterprise Backups Using Rsync

Now that we have taken a look at how to execute commands for backing up and synchronizing our directories locally and remotely, let's put it together in a way that you will more typically see in an enterprise.

Create a Backup Script

Let's create a basic backup script that we can execute via a CRON job on our local system that will push the directory we want backed up to its intended location. First things first, let's create the script. In your local home directory, let's create a file called "mybackup.sh" and make sure it contains the following:

```
#!/bin/bash

/usr/bin/rsync -a /home/linuxacademy/bkup1/
linuxacademy@10.0.0.100:/home/linuxacademy/bkup2
```

Be sure that we give this file the proper permissions:

```
linuxacademy@ip-10-0-0-162:~$ chmod 755 mybackup.sh
```

If you would like, at this point you can execute the script directly to be sure it behaves as you expect OR (more appropriately), execute the command in the script as we talked about earlier so that it gives you a dry run:

```
linuxacademy@ip-10-0-0-162:~$ rsync -anv bkup1/
10.0.0.114:/home/linuxacademy/bkup1
```

Assuming the dry run provides you with acceptable output (and if not, fix your error or adjust your script), we can create a CRON job that will schedule this backup to run and push to our backup server.

Create the CRON Job

So all we have left on our Enterprise Backup set up is to schedule this task to run. Now, although we are going to set our backup to happen at midnight every night, you may want to change the first run of this to a couple minutes from whatever time you save the CRON job so you can see it run. Here is a quick "cheat sheet" for CRON entries on your system:

```
MINUTE HOUR DAYOFMONTH MONTH DAYOFWEEK
00     12   *          *     5
```

The above timing, for example, would run a CRON task at 12pm on every day of every month that is a Friday (0 – Sunday, 6 – Saturday). In our case, we can edit CRON jobs for the "linuxacademy" user by executing:

```
linuxacademy@ip-10-0-0-162:~$ crontab -e
```

This will put us into our default editor (or, the first time, will prompt you to pick a default editor like VI or NANO). Under the documentation/comments you find yourself, add the following line (or change the time for your testing now):

```
00 00 * * * /home/linuxacademy/mybackup.sh >
/home/linuxacademy/mybackup.log 2>&1
```

Once you save the file, you will be presented with a message that indicates your CRON job was saved. If you have created the CRON job to run at a time just a few minutes from now so you can watch it execute, you will have two ways to know it ran. The first way is in our CRON itself – we are directing CRON to take all the output from that script and send it to a file called "mybackup.log". The way we have this written is that this file will overwrite itself on each run, you could of course, change the redirect from overwrite to append by changing the redirect to '>>' instead of '>'. Additionally, we are redirecting all error messages to the same file, so if anything goes wrong you will see the file created with the error contents.

The other way of making sure your cron job ran is to tail and follow your CRON log file. You can do that by executing the following:

```
linuxacademy@ip-10-0-0-162:~$ sudo tail -f /var/log/syslog
...some other text, but you should see a line like:
Jul 20 00:00:01 10.0.0.162 CRON[13541]: (linuxacademy) CMD
(/home/linuxacademy/mybackup.sh > /home/linuxacademy/mybackup.log
2>&1)
```

That's it! You have managed to backup your directory to a remote server automatically like the pros do it. Rsync offer a lot of power options that give you complete control over what and how your backups work, see Appendix B: Additional Rsync Options for more information.

Appendix A – CRONTAB Example

```
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command
    00  00  *  *  *  /home/linuxacademy/mybackup.sh > /home/linuxacademy/mybackup.log 2>&1
```

Appendix B – Additional Rsync Options

`rsync –az` *source destination*

- This will transfer the files indicated in the source location to the destination using compression. This alleviates load on the network and for large file transfers or lots of easily compressed files (like text or video), can save significant time.

`rsync –azP` *source destination*

- This will transfer the files indicated in the source location to the destination using compression. This alleviates load on the network and for large file transfers or lots of easily compressed files (like text or video), can save significant time. Additionally, the '-P' flag is useful when you are monitoring the transfer since it will show you the progress of each file as it is being sent or received.

`rsync –a --delete` *source destination*

- This is a very common implementation of truly keeping directories or filesystems in sync. Unlike using the '-a' flag alone, the '–delete' flag will also delete remote files if they have been deleted from the source. Without the delete flag, if a source file is deleted completely, it still remains in the destination location.

`rsync –a –exclude 'dir1'` *source destination*

- In addition to the normal rsync behavior, by passing the '–exclude' flag with a directory, file or wild card, rsync will exclude whatever content matches. In this case, rsync will push everything from *source* to *destination* except the file or directory called 'dir1'.

rsync –a –exclude-from ‘/home/linuxacademy/list’ *source destination*

- In addition to the normal rsync behavior, by passing the ‘–exclude-from’ flag with a path to a file, rsync will read the contents of that file and then exclude ALL contents in that file that match relative to the backup path (in this case /home/linuxacademy)

Example of ‘exclude-from’ file contents:

dir2

.ssh

.bash*

This would cause rsync to ignore the contents of the ‘dir2’ directory and ‘.ssh’ directory as well as any files that start with ‘.bash’ (such as .bash_history, .bashrc, etc)