Linux

Jim Harner

10/2/2020

1.3 Linux

Data science requires underlying tools and the most basic of these is the operating system (OS). Linux is most commonly used since it is open source and has advanced features, e.g., its kernel and file system, which make handling big data feasible.

1.3.1 Linux Features

Linux manages the communication between your software and your hardware resources. This tutorial uses Docker containers based on Ubuntu Linux. For a more complete reference see: What is Linux?.

Linux has a number of components, including:

- the kernel for managing the CPU, memory, devices, etc.;
- daemons for running background processes;
- a shell for issuing commands (the bash shell in our case).

You have access to the `bash` shell in RStudio by selecting the `Shell...` menu item from the `Tools` menu or simply by clicking in the `Terminal` tab in the `Console Pane`. The `bash` shell is found in `/bin/bash`.

```
which bash
```

```
## /bin/bash
```

The Linux kernel executes command line tools consisting of several types.

- binary executables (compiled source code), e.g., `psql` for running PostgreSQL interactively or `R` for running R;
- shell builtins, e.g., `cd` for changing directories and `exit` for exiting the shell;
- interpreted scripts, e.g., `Rscript` for running R scripts;
- shell functions, i.e., functions that are executed by the bash shell.

Commands are typed into a shell. In our case we will be using the `bash` shell, or "born again shell." The original UNIX shell was called `sh` and hence the name for the improved shell we now use. We will use the binary executable `psql` in module 2, i.e, M2_DataExtraction. R scripts associated with Hadoop are run in Module 4.

1.3.2 Files and Directories

We now illustrate various shell builtin commands.

Select `Shell...` from the `Tools` menu or click on `Terminal` in the `Console` panel. To get oriented, type the following into the shell. Or you can run this chunk by clicking on the right-pointing arrow (after executing the preceding chunks with the downward point arrows).

`pwd` prints the working directory, which by default is the directory containing the file we opened, i.e., `linux.Rmd`. `echo` sends what follows in quotes to standard output. The `-e` option in the `echo` command allows backslash escapes, which in this case create a new line specified by `\n`. For reference: `\b` creates backspaces; `\c` suppresses the trailing new line; `\t` creates horizontal tabs, etc. We now change directory (`cd`) to the home (login) directory using the `HOME` environmental variable, whose value is extracted by `$`. The home directory is also denoted by `~`.

```
pwd
echo -e "\nchange to your home directory and print the home directory path"
cd $HOME
pwd
echo -e "\nthe list of the files and subdirectories in your login ($HOME) directory"
ls
```

```
## /home/rstudio/rspark-tutorial/m1_fundamentals/s3_linux
##
## change to your home directory and print the home directory path
## /home/rstudio
##
## the list of the files and subdirectories in your login (/home/rstudio) directory
## rspark-tests
## rspark-tutorial
```

You are now be in the directory `/users/<username>`, where `username` is your login name. The directory contents is given by `ls`, which lists the files and sub-directories in your home directory.

However, once you go to another bash or R code chunk, you are back in the directory containing the file you are editing, i.e., `linux.Rmd` here..

```
pwd
echo -e "\nthe files in the directory containing the file we opened, i.e., linux.Rmd"
ls
```

```
## /home/rstudio/rspark-tutorial/m1_fundamentals/s3_linux
##
## the files in the directory containing the file we opened, i.e., linux.Rmd
## linux.html
## linux.pdf
## linux.Rmd
```

Let's get more detail on the files and directories in your working directory. This can be done using the `l` or`-al` options with `ls`, or `ls -a -l` if you prefer.

```
echo -e "\nlist the files in your working directory using long format"
ls -l
```

```
##
## list the files in your working directory using long format
## total 884
## -rw-r--r-- 1 rstudio rstudio 654031 Oct  3 02:43 linux.html
## -rw-r--r-- 1 rstudio rstudio 228966 Oct  1 20:13 linux.pdf
## -rw-r--r-- 1 rstudio rstudio  18679 Oct  3 02:45 linux.Rmd
```

Also, let's list the files in your home directory without changing directories (your output will differ from my login directory).

```
echo -e "\nlist the files in your home directory from your working directory using the long format and s
ls -al $HOME
```

```
##
## list the files in your home directory from your working directory using the long format and showing
## total 48
## drwxr-xr-x  8 rstudio rstudio 4096 Oct  1 21:25 .
## drwxr-xr-x  1 root    root    4096 Jun 30 06:09 ..
## -rw-------  1 rstudio rstudio  119 Oct  2 00:13 .bash_history
## -rw-r--r--  1 rstudio rstudio  297 Sep 28 19:40 .bashrc
## drwxr-xr-x  3 rstudio rstudio 4096 Oct  1 20:35 .cache
## drwxr-xr-x  4 rstudio rstudio 4096 Oct  1 18:10 .config
## -rw-r--r--  1 rstudio rstudio   51 Oct  1 20:34 .gitconfig
## drwxr-xr-x  3 rstudio rstudio 4096 Oct  1 14:18 .local
## -rw-r--r--  1 rstudio rstudio  273 Oct  1 20:13 .Rhistory
## drwxr-xr-x  2 rstudio rstudio 4096 Oct  1 14:03 rspark-tests
## drwxr-xr-x 13 rstudio rstudio 4096 Oct  3 01:18 rspark-tutorial
## drwxr-xr-x 15 rstudio rstudio 4096 Oct  2 00:12 .rstudio
```

The option `-a` causes `ls` to list all files and directories, including those that are normally hidden, whereas `-l` invokes the long format (`l`), which gives the permissions and other information about the owner, group, size, modification date, time, etc. The file and directory names are given in the last column of the output.

Lines beginning with a `d` indicate a directory, whereas those beginning with a `-` indicate a file. For example, `rspark-tutorial`, is the directory containing the tutorial notes. Files or directories beginning with a `.` are hidden, e.g., `.bash_history` is a hidden file, whereas `.rstudio` is a hidden directory. There are two special hidden directories: `.` refers to the current directory and `..` is its parent. A common use case for `..` is to type `cd ..` to move up one level in the directory hierarchy.

We often use `.`, current directory, to execute interpreted scripts, e.g., `./start.sh` to launch the Docker containers for `rspark` from Docker Hub (see Section 1.3.3 below). Here `start.sh` is the name of the bash script with the extension `.sh`. An R script has `.R` as the extension.

The permission string is given by the first 10 characters in each output line from `ls -al`. After the first character, we have permissions (file/directory modes) in three sequences of three characters each.

For example, for `rspark-tutorial`:

- The first `rwx` indicates you (the owner, i.e., `rstudio`) have read, write, and execute permissions;
- The second `r-x` indicate your group, `rstudio` here, has read and execute permissions, but not write permissions.
- The third `r-x` indicates all users who can log into your Linux instance have read and execute permissions, but not write permissions.

These permissions are also denoted by `755`, where $7 = 2^2 + 2^1 + 2^0$ and the exponent is the position in the sequence `rwx` from right to left, starting at 0. In a similar vein, `r-x` can be represented as $5 = 2^2 + 0 + 2^0$. The default when you create a directory is `755`. Execute permission is important for a directory since otherwise you cannot `cd` to the directory.

Note that if you want privacy you should use `700` for permissions. Let's make a temporary directory in the working directory and then change its permissions. The command `mkdir` makes a directory, in this case `temp`.

```
mkdir temp
ls -al
```

```
## total 896
## drwxr-xr-x 3 rstudio rstudio   4096 Oct  3 02:45 .
## drwxr-xr-x 9 rstudio rstudio   4096 Oct  1 20:13 ..
## -rw-r--r-- 1 rstudio rstudio 654031 Oct  3 02:43 linux.html
## -rw-r--r-- 1 rstudio rstudio 228966 Oct  1 20:13 linux.pdf
## -rw-r--r-- 1 rstudio rstudio  18679 Oct  3 02:45 linux.Rmd
## drwxr-xr-x 2 rstudio rstudio   4096 Oct  3 02:45 temp
```

We use `chmod` (change mode) to change the permissions of `temp` from the default `755` to `700`.

```
chmod 700 temp
ls -al
```

```
## total 896
## drwxr-xr-x 3 rstudio rstudio   4096 Oct  3 02:45 .
## drwxr-xr-x 9 rstudio rstudio   4096 Oct  1 20:13 ..
## -rw-r--r-- 1 rstudio rstudio 654031 Oct  3 02:43 linux.html
## -rw-r--r-- 1 rstudio rstudio 228966 Oct  1 20:13 linux.pdf
## -rw-r--r-- 1 rstudio rstudio  18679 Oct  3 02:45 linux.Rmd
## drwx------ 2 rstudio rstudio   4096 Oct  3 02:45 temp
```

We then remove the directory, which must be empty, by `rmdir`.

```
rmdir temp
```

To remove a file, use `rm` followed by the file name. To remove a directory containing files and other directories, use `rm -R` followed by the directory name. The `-R` option recursively removes files and directories, and thus should be used with extreme caution!

Linux has a large number of built-in commands, which you should gradually learn. Extensive information is given by the `man` or manual pages for each command. For example:

```
man chmod | head -n 15
```

Most commands have many options, but usually you can get by with only a few. Often the output from `man` is overwhelming and difficult to read. Since `man` is not part of our Ubuntu Linux distribution used here due to its size, consider googling the command.

1.3.3 Docker Virtualization

Docker does virtualization at the operating system level using containers. This allows Docker to package applications and their dependencies in one or more virtual containers running on the Linux kernel with resource isolation. In its basic configuration `rspark` consists of four containers—`rstudio`, `postgres`, `hadoop`, and `hive`. Spark is installed within the `rstudio` container. An experimental version of `rspark` builds a virtual Spark cluster based on a `master` container and multiple `worker` containers.

Each container has an associated `Dockerfile`, which is a script for building the layers of a Docker image. Images can be built from a base image. For example, `rstudio` is built from `rocker:verse`, an image for running R from RStudio with both the `tidyverse` and publishing capabilities. A Docker container is simply a running instance of an image.

Installing Docker for the Mac The Mac is a BSD UNIX-based system and therefore has many of the tools required for data science. Other tools are easy to install. For the most part, data science runs on open-source Linux-based environments, which are compatible with macOS. Your principal interface to the UNIX side of the Mac is the Terminal.app found in the Utilities directory (folder) in the Applications directory (folder). You should drag this App to your Dock for easy access. You can also run the X11 windowing system, but it is not necessary since we will be working from the command line.

macOS comes preinstalled with git, but it should be updated. You can determine the version of git by typing git version at the command prompt. If git is not present or if the version is earlier than 2.17.1, you need to install or update git.

Go to: git for Mac and download/install the latest release. You probably will get a message that the package cannot be opened because it is from an unidentified developer. If you see this, then open up System Preferences > Security & Privacy > General, and click ‘Open Anyway’ for this pkg in the lower right.

The Docker solutions work on a Mac with macOS 10.13 (High Sierra) or later. Go here to get Docker Desktop for the Mac: and follow the directions for installation. You will need to create an account on Docker to download the disk image (.dmg file) and you may get a verification email. After dragging Docker to your Applications folder, run it. You may see ‘“Docker…” can’t be opened…’, so again click ‘Open Anyway’ in Security & Privacy preferences. Docker Desktop provides a graphical interface for managing Docker containers.

Installing Docker for Windows Windows 10 is not UNIX based and requires additional software. However, the basic requirement we need here is access to the `bash` shell. The easiest solution is a bash emulator.

For bash access install git for Windows. This will give you an emulated bash shell which we need for running Docker and/ or Vagrant and of course `git` for version control.

Docker requires a hypervisor (Hyper-V in the case of Windows) to run and it is not installed by default on Windows 10. Hyper-V can be installed on Windows 10 Enterprise, Professional, and Education, but not on Windows 10 Home, Mobile, and earlier versions of Windows. If you meet the system requirements, follow the instruction for installing Hyper-V here.

Once Hyper-V is installed, get Docker for the Windows and follow the directions for installation.

Note: Windows Subsystem for Linux v2 (WSL 2) is now available for Windows 10 Professional and it supports Docker containers. Directions will be fothcoming.

The following installations assume you have a `bash` shell available. Ideally, you should be on a high-speed network.

Running `rspark` from Docker Hub For a more complete description of the Docker Hub installation process, see the `README` file from the `rspark-docker` repo: https://github.com/jharner/rspark-docker

Before beginning, launch Terminal.app on the Mac or git bash on Windows (start typing bash from the Start menu to find it). Once running, you should be in your home directory, e.g, type `pwd` to verify. Stay there or `cd` to wherever you want `rspark` installed.

`rspark` can be installed by pulling the Docker images from Docker Hub. Run the following command in the terminal:

```
git clone https://github.com/jharner/rspark-docker.git
```

This should only be run the first time. If `rspark-docker` gets updated (I will notify you if this is the case.), execute:

```
git pull origin master
```

You are now ready to start your session. Assuming you are in the parent directory of `rspark-docker`, execute:

```
cd rspark-docker
./start.sh
```

The last command runs the `start.sh` shell script, pulls the pre-built tagged images from Docker Hub (currently 1.0.3), and launches the containers. You must have execute privileges to run the shell script. If not, use `chmod`, e.g., `chmod 755 start.sh`.

It will take awhile the first time or after a pull from GitHub. Eventually, you will return to the line prompt.

Now open a browser in Chrome (or Firefox or Safari) and type the url as: `localhost:8787` and sign into RStudio:

Username: rstudio
Password: rstudiojh

When finished, quit the current R session (red button). The containers can be stopped (or terminated) from Desktop Docker or from the command line. Stopped containers can be restarted using Docker Desktop.

To start the containers again if the containers have been terminated, assuming you are in your `rspark-docker` directory, execute the following commands:

```
./start.sh
```

Then login to RStudio as before.

Running `rspark` from GitHub If you are interested in how the rspark environment is created, go to:

https://github.com/jharner/rspark

You can clone the development environment to your local machine by issuing the following command:

```
git clone https://github.com/jharner/rspark.git
```

In this case you must build the Docker images yourself by the following:

```
./start.sh build
```

This will build the images and launch the containers. Login as before.

When finished, quit the current R session (red button). The containers can be stopped (or terminated) from Desktop Docker or from the command line. Stopped containers can be restarted using Docker Desktop.

To restart the containers, simply issue,

```
./start.sh
```

i.e., you do not need the `build` option.

To update your local repo of `rspark`, type:

```
git pull origin master
```

Since you local repo has been changed, you will need to specify the `build` option the first time you execute `start.sh`.

1.3.4 Running Docker Commands from the Command Line

You can `ssh` into any container by specifying its name, which can be found by issuing the following Docker command in the terminal of your host machine.

```
docker ps
```

The output gives details concerning the currently running containers. To see all running and stopped containers specify the `-a` option.

You can enter containers from the local command line using `ssh` (secure shell). For example, to `ssh` into the `rstudio` container, run the following in your local (host) terminal:

```
docker exec -ti rstudio bash
cd /home/rstudio
```

where “rstudio” is the container `NAME`, the `-ti` option requests an interactive terminal, and `bash` is the process we wish to run. You will be logged in as `root`, which means that your home directory is `/root`.

At this point you are in the `bash` shell within the `rstudio` container. You can now run a command-line version of R by simply typing `R`, a binary executable, at the `#` or `$` prompt. This will give you an R console in your terminal with `>` as its prompt. However, `ssh`ing into a container is generally done for debugging in creating the container's Dockerfile—not for running application programs.

Note that the `Terminal` tab within RStudio's `Console` panel is running bash within the `rstudio` container. The terminal prompt is `rstudio@<CONTAINER ID>:<PATH to working directory>`. Ironically, you can run a separate R process within the Terminal.

Container Management You can get a list of running containers by typing `docker ps` into your local terminal or `docker ps -a` to list both the running and stopped containers. The bash script `dockill.sh` in the `rspark` GitHub repo has a series of `docker` commands to control your images, containers, and volumes. For example, to stop running containers, issue:

```
docker stop $(docker ps -a -q)
```

or to delete all containers, issue:

```
docker rm $(docker ps -a -q)
```

Likewise, images can be deleted by:

```
docker rmi -f $(docker images -q)
```

If you are having problems with your containers, you can stop and then delete the containers and images (in this order). Once you delete the containers, images, and perhaps the volumes, then you must download new images from Docker Hub or rebuild the images from GitHub.

1.3.5 Vagrant/ VirtualBox Virtualization

If you cannot install `rspark` using Docker, then try Vagrant/ VirtualBox. Vagrant is heavier than Docker and the download is longer, but it works well. For a more complete description of the installation process see the `README` file from the `rspark-vagrant` repo: https://github.com/jharner/rspark-vagrant

First your must install VirtualBox and Vagrant for macOS or Windows:

https://www.virtualbox.org/wiki/Downloads
https://www.vagrantup.com/downloads.html

Windows users will need to install git for Windows for bash access (as above) and get the 64-bit version of Vagrant. However, you do not need Hyper-V, which may not be available for your Windows version.

The installation is similar to the what we did with Docker. We assume you are in your home directory, but you can `cd` to any directory. The following should only be done the first time:

```
git clone https://github.com/jharner/rspark-vagrant.git
```

If `rspark-vagrant` gets updated, execute:

```
git pull origin master
```

You are now ready to start your session.

```
cd rspark-vagrant
./start.sh
```

The last command runs the start shell script. It will take awhile the first time or after a pull. Eventually, you will reach a line somewhat like: "hive_1 | SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]". Leave the terminal running.

Now open a browser and type the url as: `localhost:8787` and theb sign into RStudio:

Username: rstudio
Password: rstudiojh

When finished, quit the current R session (red button). Then return to the terminal and type: Control-C, which will stop VirtualBox `rspark` and return the prompt.

To start the session again, assuming you are in your home directory, execute the following commands:

```
cd rspark-vagrant
./start.sh
```

This is done each time you start a session after the first time.