Unix Shell

301113 Programming for Data Science

WESTERN SYDNEY
UNIVERSITY

School of Computer, Data and Mathematical Sciences

Week 12 - Shotts, 2019

Outline

Motivation

You are working on a project that requires fitting a statistical machine learning model to data. It is estimated that the model fitting process will take a week to run on you local machine.

Rather than leaving your machine running at full power all week (with the fan likely at full speed, hopefully not in your bedroom), you use a remote server to run the code and fit the model. You can upload your code, set it running, log in to the machine a week later to download the fitted model.

It is common for research servers to run a form of Unix or Linux and have a command line interface.

Motivation

You are working on a project that requires fitting a statistical machine learning model to data. It is estimated that the model fitting process will take a week to run on you local machine.

Rather than leaving your machine running at full power all week (with the fan likely at full speed, hopefully not in your bedroom), you use a remote server to run the code and fit the model. You can upload your code, set it running, log in to the machine a week later to download the fitted model.

It is common for research servers to run a form of Unix or Linux and have a command line interface.

This lecture will closely follow the "Learning the Shell" section of the book "The Linux Command Line" by William Shotts.

Outline

What is the Unix Shell?

The Unix Shell is the command line interface to Unix/Linux machines. It is likely that you have seen use of the Shell in movies whenever someone is hacking into another computer.

The command line interface might look outdated, but when we have mastery over it, it provides great power, and with that comes great responsibility.

The Shell is very similar to the R console, since it provides interactive sessions (letting the user type in commands), but also allows scripted sessions (allowing the user to run scripts).

Getting Access to the Shell

The best way to master the Shell is to use it. Before we can use it, we must install it.

If you have a machine that is running Linux or OSX, then you already have a Unix shell.

- For Linux: open a program called Terminal
- For OSX: open a program called Terminal

If you have a machine running Windows 10, the "Windows Subsystem for Linux" must be setup.
`https://docs.microsoft.com/en-us/windows/wsl/install-win10`

- If Ubuntu was chosen at step 6, then a Shell is opened by opening Ubuntu.

Once the Terminal program is open, we are presented with a Shell prompt (just like the prompt in the R console), waiting for us to enter a command.

Unix Philosophy

The Unix philosophy is to have each program do one thing and do it well. The operating system was designed in the early 1970s, consisting of a Kernel that interfaces with the computer hardware and a Shell that covers the Kernel providing a user interface to the Kernel. This modular structure is still used today in Unix and Linux OSes.

The Shell provides us with access to commands (functions) flow control and branching statements and also pipe operations that allow us to combine each command into powerful statements.

For example,

```
cat blog.txt | tr " " "\n" | sort | uniq -c | sort -n
```

takes the blog.txt file and returns a sorted list of words and their count from the blog file. Each command (cat, tr, sort, uniq) did one task, but when they are combined provide great power.

First Commands

After you have opened the Terminal and changed the background to be transparent and the font to be green (because everyone wants their Terminal to look like The Matrix movies), we can start typing commands.

- `pwd` prints the name of the working directory
- `ls path` lists the contents of the provided directory
- `cd path` changes the working directory to `path`

When providing paths, note that

- `/` is the root (top level) directory,
- `.` is the current directory
- `..` is the parent directory
- `-` is the previously visited directory
- `~` is your home directory

Also, `cd` with no path changes the working directory to your home directory and `ls` with no path lists the contents of the working directory.

Where am I?

Problem

After issuing the following commands, which directory am I working in?

```
cd /tmp
cd larry
cd barry
cd ..
cd -
```

Unix Filesystem Hierarchy

A note about the Linux filesystem structure. Linux filesystems generally follow the Filesystem Hierarchy Standard. The root directory contains the directories:

- `/bin` Essential binary files
- `/boot` Files to boot the machine
- `/dev` Device files (most interface with hardware)
- `/etc` System configuration files
- `/home` Users' home directories
- `/lib` Libraries
- `/media` Mount directories for removable media (USB drives, CD drives)
- `/proc` Process and kernel information
- `/root` Root user's home directory
- `/sbin` System binaries
- `/tmp` Directory for temporary files (the files are regularly deleted).
- `/usr` Read only user data
- `/var` Variable files (caches, logs)

When a terminal is opened, the working directory is set to our home directory.

Command Options

All commands have the format

```
command -options arguments
```

E.g. `ls -l`, `ls -a`, `ls -la`, `ls /home`, `ls -l /home`

Each of the commands are documented in detail in the manual. To read the manual page for a command:

```
man command
```

E.g. the manual page for `ls` is shown using `man ls`.

For more information about the manual, look at its manual page using `man man`.

Manipulating Files

Files can be manipulated using the commands:

- `cat file` prints the contents of the file to the screen
- `cp filefrom fileto` copy a file to a new file
- `mv filefrom fileto` move a file to another directory or to a new name
- `rm file` remove a file (delete)
- `mkdir directory` create a new directory
- `rmdir directory` remove a directory (delete)

We can use the following notation when referring to files

- `.` the current directory
- `..` the parent directory
- ~ your home directory

Moving to tmp

Problem

How do we create the directory structure `/tmp/larry/barry`, the copy the file `gary.txt` from our home directory to the directory `barry`?

Pattern Matching

There are many times when we want to move all files of a certain type to another destination, or remove a set of files, or copy a set of files. Pattern matching can be used on file names to select a set of files. Commonly used pattern matching symbols are:

- `*` match everything
- `?` match any one character
- `[abc]` match any one of the characters in the brackets

Example

- To move all R scripts to the directory `scripts`, use `mv *.R scripts/`
- To move all files experiment000.csv to experiment009.csv to directory `results` (but not experiment010.csv onwards), use `mv experiment00?.csv results/`
- To copy all files in the current directory to the `/tmp` directory, use `cp ./* /tmp/`

File Permissions

You might have noticed that files and directories can be created in our home directory but not in other directories outside of our home.

Each file and directory has an owner, a group and its own set of permissions. We can see the owner, group and permissions when we run `ls -l`

```
-rw-------   41 frank  staff   1312 Nov 17 11:36  franks_file.txt
-rw-r--r--  107 lapark staff   3424 Feb 11  2020  report.doc
drwxrwx---   19 lapark wheel    608 Oct 10  2020  expenses
drwxr-xr-x   28 lapark staff    896 Nov 27  2018  web_pages
```

- The first column shows the permissions for the file/directory (e.g. `-rw-r--r--`)
- The third shows the owner (e.g. `lapark`)
- The fourth shows the group (e.g. `staff`)

Reading File Permissions

The permission line has four sections:`d rwx rwx rwx`, each letter can be turned off, showing –.

- The first letter indicates that the file is a directory (or not).
- The second set show the owner permissions.
- The third set show the group permissions.
- The fourth set show the world (all other user) permissions.

Each user type can have `read`, `write` and `execute` permission.

- `read` the user can read the file
- `write` the user can write (edit) the file
- `execute` the user can run the file (if it is a program or script).

Examples

- `rw-------` only the owner can read and write to the file.
- `rw-r--r--` everyone can read the file, but only the owner can write to the file.
- `rw-rw----` the owner and members of the assigned group can read and write to the file.

Setting permissions

To set the permissions of a file, we must know how to convert binary to decimal.

Each set `rwx` is a binary sequence (each attribute is either on or off, 1 or 0).

- The permission `r--` is equivalent to the binary `100` which is 4 in decimal.
- The permission `-w-` is equivalent to the binary `010` which is 2 in decimal.
- The permission `--x` is equivalent to the binary `001` which is 1 in decimal.

We can combine these obtain any set of permissions.

Setting permissions

To set the permissions of a file, we must know how to convert binary to decimal.

Each set `rwx` is a binary sequence (each attribute is either on or off, 1 or 0).

- The permission `r--` is equivalent to the binary `100` which is 4 in decimal.
- The permission `-w-` is equivalent to the binary `010` which is 2 in decimal.
- The permission `--x` is equivalent to the binary `001` which is 1 in decimal.

We can combine these obtain any set of permissions.

The permissions for a file or directory can be set using `chmod`

```
chmod 600 file.txt # set the file permission to only read and write for the owner
chmod 777 file.txt # set the file permission to read, write and execute fot all users
```

- For the first example `600` is `110 000 000` which maps to `rw- --- ---`
- For the second example `777` is `111 111 111` which maps to `rwx rwx rwx`

Reading gary.txt

Problem

We have written a program that might be helpful to others. We want everyone to be able to run the program, but we don't want anyone to be able to read it or modify it.

Outline

Running, killing, stopping

When a program is run, the shell prompt returns when the program finishes running. There are a few ways to control the shell while a program is running.

- `Ctrl-c` kill a program
- `Ctrl-z` stop a program

A stopped can be thought of as paused, and can be resumed again.

The list of stopped programs is shown by running `jobs`. To resume one of the stopped programs, find its number in the jobs list and run `%number`, e.g. the first job can be resumed using `%1`.

Background and foreground jobs

A program that we can see currently running is called a “foreground job”. A shell running a foreground job only provides a prompt when the job finishes.

If we want to run a job, but still get access to the shell prompt, the job can be sent to the background. To run a job in the background, append an ampersand “&” to the end of the command to run the program.

```
find / -name "*.sh"   # find all files ending in .sh starting at /
find / -name "*.sh" & # run in the background
```

Note that `jobs` also shows background processes.

Background and foreground jobs

A program that we can see currently running is called a “foreground job”. A shell running a foreground job only provides a prompt when the job finishes.

If we want to run a job, but still get access to the shell prompt, the job can be sent to the background. To run a job in the background, append an ampersand “&” to the end of the command to run the program.

```
find / -name "*.sh"   # find all files ending in .sh starting at /
find / -name "*.sh" & # run in the background
```

Note that `jobs` also shows background processes.

To kill a job, use `kill %number`, where number is the job number.

Outline

Standard Out

The output of command line programs is sent to the screen by default. We saw this when listing the contents of a directory (`ls`).

In fact, the program output is being sent to a stream called standard out, which by default is set to the screen. We are able to control where standard out is sent using `redirection`.

Standard Out

The output of command line programs is sent to the screen by default. We saw this when listing the contents of a directory (`ls`).

In fact, the program output is being sent to a stream called standard out, which by default is set to the screen. We are able to control where standard out is sent using `redirection`.

To redirect standard out to a file, use the > symbol, followed by the file name.

```
ls > directory_list.txt
```

If the file `directory_list.txt` does not exist, it will be created and the output of `ls` will be stored in it. If the file does exist, it will be overwritten by standard out.

Standard Out

The output of command line programs is sent to the screen by default. We saw this when listing the contents of a directory (`ls`).

In fact, the program output is being sent to a stream called standard out, which by default is set to the screen. We are able to control where standard out is sent using `redirection`.

To redirect standard out to a file, use the > symbol, followed by the file name.

```
ls > directory_list.txt
```

If the file `directory_list.txt` does not exist, it will be created and the output of `ls` will be stored in it. If the file does exist, it will be overwritten by standard out.

To append the standard out to a file, use >>.

```
ls >> directory_list.txt
```

If the file `directory_list.txt` exists, standard out will be appended to it.

Standard In

Many command line programs accept input from the standard in stream. For example:

```
sort < directory_list.txt
```

where < redirects the file contents of `directory_list.txt` to standard in.

Standard In

Many command line programs accept input from the standard in stream. For example:

```
sort < directory_list.txt
```

where < redirects the file contents of `directory_list.txt` to standard in.

Note that the command line program `sort` makes use of both standard in and standard out, so if needed, we can redirect both.

```
sort < directory_list.txt > sorted_directory_list.txt
```

Doing this sends the contents of `directory_list.txt` to the input of `sort`, then the output of `sort` is sent to `sorted_directory_list.txt`.

Pipelines

Pipelines are a powerful feature of the shell that let us string together a set of commands, where the output of one is sent to the input of another.

For example, if we have many files in the directory and listing them scrolls off the screen, we can send the output of `ls` to `less`, allowing us to scroll the results.

```
ls -l | less
```

The pipe operator `|` sends the left side program output to the right side program's input.

We could also sort the file list.

```
ls | sort | less
```

Commonly used pipeline programs

Pipes can be used with any program that reads from standard in and outputs to standard out. Here are a few commonly used programs.

- `cat` sends a file contents to standard out
- `sort` sorts the standard input lines, sends the result to standard output
- `uniq` removes duplicate lines from a sorted input
- `grep` sends out only lines that match a specified pattern
- `head` outputs the first few lines
- `tail` outputs the last few lines
- `tr` translates characters from one to another
- `sed` stream editor, can perform more complex translations
- `awk` a scripted language for text processing

Counting words

We showed this command earlier.

```
cat blog.txt | tr " " "\n" | sort | uniq -c | sort -n
```

We can examine what it is doing by breaking down the components.

- `cat blog.txt` sends the contents of `blog.txt` to standard out
- `tr " " "\n"` takes standard in, changes all spaces to new lines (putting each word on its own line) and sends that to standard out.
- `sort` sorts the lines of standard in and sends it to standard out.
- `uniq -c` removes all duplicate lines from standard in, provides count of duplicates and sends it to standard out.
- `sort -n` sorts the lines from standard in, using a numeric sort (rather than sorting characters).

Middle of the file

Problem

- `head -n` provides the top `n` lines of a file
- `tail -n` provides the bottom `n` lines of a file

How can we extract lines 9 and 10 of the file `blog.txt`?

Outline

Running R

We have used R through an IDE called RStudio, but the R program itself is an interpreter, that also provides an interactive console.

- To start the R console from the command line, enter `R`.
- To leave the R console, enter `quit()` or press Ctrl-d.

Running R

We have used R through an IDE called RStudio, but the R program itself is an interpreter, that also provides an interactive console.

- To start the R console from the command line, enter `R`.
- To leave the R console, enter `quit()` or press Ctrl-d.

To run an R script, we can open the console and source the script (`source("script.R")`), or we can use an non-interactive version of R called `Rscript`.

To use `Rscript` from the Shell, type `Rscript script.R`. Any printed output will be printed to the console, so it should be redirected to a file if wanted. E.g.

```
Rscript script.R > scriptResults.txt
```

Standard in and out in R

We can make use of standard in and out when running R scripts from the command line.

The following R script reads from standard in using `readLines` then prints to standard out using `cat`. Note that `print` also prints to standard out.

```
# read from standard in
lines <- readLines(file("stdin"))

words <- unlist(strsplit(lines, split = " "))
wordTable <- sort(table(words), decreasing = TRUE)

# print top 10 words to standard out
cat(names(wordTable)[1:10])
```

To run this script, save it to a file (e.g. `topWords.R`), then using the shell

```
Rscript topWords.R < blog.txt
```

Outline

Using remote machines

When using a remote machine, we need to know how to:

- login to the machine
- copy files to and from the machine
- leave processes running on the machine when we logout

The programs used to complete these tasks are `ssh`, `scp` and `screen`.

SSH

ssh is used to login to a Unix/Linux server. `ssh` stands for “secure shell”. All data transferred across a ssh connection are encrypted. But remember that encryption does not stop “man in the middle” attacks, so make sure that the server you are connecting to is the server you want to connect to.

To connect to a server, we need the server address, and a username and password for the server.

SSH

ssh is used to login to a Unix/Linux server. `ssh` stands for "secure shell". All data transferred across a ssh connection are encrypted. But remember that encryption does not stop "man in the middle" attacks, so make sure that the server you are connecting to is the server you want to connect to.

To connect to a server, we need the server address, and a username and password for the server.

CDMS Student Server

- Any student enrolled in a unit run by the School of Computer, Data and Mathematical Science has an account on the student server `student.cdms.westernsydney.edu.au`.
- Your username is not your student number, so if you don't know what it is, you have to ask IT.
- Access to the student server is only available within the WSU network. So if access is needed outside of the network (e.g. at home), a VPN connection must first be made.

Connecting to the server

`ssh` needs your username and the server name. Open a terminal and type:

```
ssh username@student.cdms.westernsydney.edu.au
```

replacing `username` with your username for the server.

If the connection is successful, `ssh` will ask for your password. Once the correct password is provided, the server shell will provide you with a prompt and you must say "we're in!" and then talk about hacking the mainframe (as done in most computer movies).

The server shell

The server shell will be very similar to the shell on your own computer, so you should be able to navigate the directories, create files and run programs. There might be different programs installed on the server to use compared to your own computer.

We have limited permissions on the server, so we will not be able to perform tasks such as installing software, or viewing other peoples files. Remember that we can view file and directory permissions using `ls -l`

If there are programs that you want installed on the server, let the system administrator know.

Moving files to the server

Before working on the server, we must either create files on the server or copy existing files from our local machine to the server. `scp` is similar to `cp`, but can operate between machines.

```
scp filefrom fileto
```

To copy a file from our local machine to our CDMS student server home directory, run on our local machine:

```
scp file username@student.cdms.westernsydney.edu.au:~/
```

To copy the file to the servers /tmp directory:

```
scp file username@student.cdms.westernsydney.edu.au:/tmp/
```

To copy a file from our home directory on the server to our local machine:

```
scp username@student.cdms.westernsydney.edu.au:~/file ./
```

To copy a directory, the recursive flag must be used:

```
scp -r scriptsDirectory username@student.cdms.westernsydney.edu.au:~/project/
```

Screen

There are often times that we want to run multiple shell processes on the server, so we can edit code and run code at the same time. To do this, we could open multiple ssh connections to the server.

There are also times where we want to logout from the server, but leave processes running on it (such as R running our script).

Screen

There are often times that we want to run multiple shell processes on the server, so we can edit code and run code at the same time. To do this, we could open multiple ssh connections to the server.

There are also times where we want to logout from the server, but leave processes running on it (such as R running our script).

GNU Screen is software that provides us with ability to create multiple virtual consoles and the ability to attach and detach from them. All of this is done on the server side.

To start screen, `ssh` into the server, then run:

```
screen
```

To stop screen, press `Ctrl-d`

Accidental disconnection

When not using screen, a disconnection from the server (e.g. due to a network problem) leads to our processes being killed. Using screen allows us to login to the server and reattach to the screen session with all of out processes still running.

Using Screen

Screen is highly configurable, we will examine the default settings. To interact with Screen, use `Ctrl–a` then provide a another key combination to send a command.

- `Ctrl–a c` Create a new window with a shell
- `Ctrl–a "` Present a list of all windows
- `Ctrl–a Ctrl–a` Switch to the previous window
- `Ctrl–a digit` Switch to window number `digit`
- `Ctrl–a d` Detach screen

There are many more commands that you can find in the manual (`man screen`).

Using Screen

Screen is highly configurable, we will examine the default settings. To interact with Screen, use `Ctrl–a` then provide a another key combination to send a command.

- `Ctrl–a c` Create a new window with a shell
- `Ctrl–a "` Present a list of all windows
- `Ctrl–a Ctrl–a` Switch to the previous window
- `Ctrl–a digit` Switch to window number `digit`
- `Ctrl–a d` Detach screen

There are many more commands that you can find in the manual (`man screen`).

If you have detached a screen, the list of detached screen is shown using `screen –ls`. To attach a screen, use `screen –dr`.

Using `Ctrl–a`

If you use a program that requires the use of `Ctrl–a`, Screen has remapped it to `Ctrl–a Ctrl–a`.

Editing Code

Editing code on a server requires us to use a text editor that resides on the server. The two main text editors are:

- Emacs and
- Vim;

The rivalry between users of these two text editors has led to the editor war.

Both of these editors are very powerful, but it will take time to become proficient with them.

nano is a simpler text editor that also usually exist on Unix/Linux servers.

We will not provide details of these editors here. Beginner's guides can be found on the Web.

A typical session

Scenario: we have a script to model data. The model is complex, so the script will take days to run. We can either:

1. Run the script in Rstudio on our computer, leaving the computer on for days, with the CPU running at 100% and the fan spinning loudly.
2. Run the script on a remote server.

A typical session

Scenario: we have a script to model data. The model is complex, so the script will take days to run. We can either:

1. Run the script in Rstudio on our computer, leaving the computer on for days, with the CPU running at 100% and the fan spinning loudly.
2. Run the script on a remote server.

To run the script on the remote server, we must:

1. Upload our script to the server.
2. Login to the server.
3. Start a screen session.
4. Start the script
5. Detach the screen session and logout of the machine.

Checking on the script

Over the next few days, we should check on the script to see if it is still running, or if it has finished successfully.

1. Login to the server.
2. Attach the screen session.
3. Check the R session, and any log our output files.
4. Detach the screen session and logout of the machine.

If the R process has finished and you have the results, the screen session can also be closed. The results can be copied to our local machine for analysis.

Sharing the server

Research servers are shared, so we should make sure that we don't consume all of the server resources.

- Check the list of processes and the used memory using `top`
- Check for other logged in users using `w`
- If your process will take a long time to run, `nice` the process (giving it lower priority).
- Make sure you are subscribed to any server announcement email lists.

Summary

- Using our own machine for computation is convenient, but there are times when we should or must use a remote server.
- Unix/Linux machines provide us with shell access
- Programs can be run in the foreground or background.
- Program input and output can be redirected to obtain program pipelines
- R can be used on the command line
- GNU Screen allows us to run multiple shells and leave programs running on the server when logged out.