Using Linux as a Development Platform for Scala Projects

Journal:	*Computer*
Manuscript ID	Draft
Manuscript Type:	Computing Practices
Date Submitted by the Author:	n/a
Complete List of Authors:	Tesileanu, Roxana; National Institute of Forest Research and Management (INCDS), Environmental statistics and Bioinformatics
Keywords:	D.1.1 Applicative (Functional) Programming < D.1 Programming Techniques < D Software/Software Engineering, D.2.17.i Programming paradigms < D.2.17 Software Construction < D.2 Software Engineering < D Software/Software Engineering

author: Roxana Tesileanu, roxana.te@web.de

affiliation: National Institute of Forest Research and Management (INCDS), Environmental Statistics and Bioinformatics, Brasov Station

date: October 2017

Using Linux as a Development Platform for Scala Projects

Abstract

Scala has already proven its advantages, and many big projects are implemented using this language. Though, newcomers and researchers from other branches than computer science might experience anxiety facing this functional programming language. With no reason though, because there already are many useful learning resources available. This paper offers an introduction into the software development process of Scala projects pointing to some of the available learning resources.

Keywords: Scala language, Simple Build Tool (SBT), functional programming

1. Introduction

In order to implement software development projects, one should consider which tools to choose for the different components of the development system, which according to Rehman and Paul (2003) are the following: the hardware platform, the operating system, editors, compilers and assemblers, debuggers, version control system and bug tracking (Tesileanu 2017). For implementing Scala projects, working with the Simple Build Tool (SBT) (http://www.scala-sbt.org) condenses the list of the components of a development system, as it unites different steps in one tool (compiling, building, testing and debugging) (see Suereth and Farwell, 2016). Another advantage of using SBT is the opportunity of working interactively in Scala REPL, a tool for evaluating expressions in Scala (https://docs.scala-lang.org/overviews/repl/overview.html) within a SBT session, to develop bits of code which can then be inserted in the editor of choice.
It is also important to understand that software development "is not just writing code", but rather a more comprehensive process (Rehman and Paul, 2003). Rehman and Paul (2003) describe it as comprising the following steps: requirement gathering, writing functional specifications, creating

architecture and design documents, implementation and coding, testing and quality assurance, software release, documentation, and, support and development and release of new features. Each project starts with a requirement analysis which investigates the real-world need of the final product. Which functions should the new software carry in real-world problem solving within the specified domain? Further, functional specifications are declared to state the functionality of a software product at an abstract level "defining its input/output behavior". On the basis of the functional specifications, an architecture of the product is created. The architecture "defines the different components of the product and how they interact with each other", without providing the explicit details on how they should be implemented to reach the desired functionality. This happens at the design stage, when design documents are created which define each individual component to the level of functions and procedures. Using the design documents and development tools (SBT and editor) the code is then implemented and tested. There are more types of testing: unit testing (testing one part or one component of the product using test cases to test functionality of this part of the software), sanity testing to check if all components compile, regression or stress testing to check the long-term behavior of the product when used continuously over a period of time, and functional testing using test cases built on functional specifications. If a bug (an anomaly) is found it must be reported and fixed. The documentation includes: technical documentation developed during the development process, technical documentation prepared for technical support staff, and end-user manuals and guides. Finally, the last stage of the life cycle of a software development project is the support and release of new versions depending on requirements.

The present article aims at introducing the reader to the tools needed to put in motion the development system of Scala projects on Linux, with emphasis on the Ubuntu distribution. The following sections will cover brief introductions in using the VIM editor as an editor for Scala code and using SBT to create and manage Scala projects.

2. Using VIM as an editor for Scala code

VIM is an editor which can be used from within the Linux command line and can be adapted to support the editing of Scala code by installing the vim-scala package of Derek Wyatt found at https://github.com/derekwyatt/vim-scala.
Under Linux you can install VIM using the package tools at hand (like e.g. dphg, apt-get, aptitude, rpm or yum) depending on the installed distribution. In Ubuntu the command for installing VIM is: “$ sudo apt install vim”. Afterwards, you open the command line and launch VIM by typing “$ vim <ENTER>”. The following instructions will make the first VIM session easy to pass through:
- because VIM starts in command mode you can change to insert mode with the "i" key
- to exit insert mode and return to command mode, press the <ESC> key

- to write the open file to the hard drive use the command ":w filename" (in command mode) or just ":w" if the file already has a name
- to save the changes type ":" in command mode
- to exit VIM type ":wq" to quit and save changes, or ":q!" to discard changes (both in command mode)
- pressing <ESC> will not just place you in command mode but also cancel an unwanted and partially completed command in command mode.

The following paragraphs will help you navigate through the VIM file, and, process text (append, delete, copy, paste, search and replace, edit multiple files and get help).

To move the cursor you use: "l" (or right arrow), "h" (or left arrow), "j" (meaning going down one line), or "k" (meaning going up one line). Moving inside the current line is possible with "0" (to go to the beginning of the current line) and "$" (to go to the end of the current line). To go to the end of the file, i.e. to the last line of the file you use "G" and to go to the beginning of the file, i.e. to the first line you use "gg". To go to a specific line you use "numberG" (for example: 2G to go the the second line).

When you are in command mode, you can start inserting text by using the "i" key to place you in insert mode, but there are also other commands which can place you in insert mode so that you can start appending text. One of them is the "a" key, which lets you append text right at the spot where the cursor is placed, or the "A" key which lets you append text directly at the end of the line. You can also use the "o" key to open a line below the current line or "O" to open a line above the current line.

You can undo changes by using the "u" key and redo changes the using "<CTRL>r".

Deleting text in command mode goes with (but not only) the following commands:
- the "x" key deleted the current character; "3x" will delete the current character and the next two characters
- the "dd" command deletes the current line; "6dd" will delete the current line and the next five lines
- the "dG" command will deleted to the end of the file
- the "d$" command will delete to the end of the line
- the "d0" command will delete to the beginning of the line.

The commnads based on "d" nut just delete text but also copy it to a paste baffer, which can be later recalled with the "p" command to paste the contents of the buffer after the cursor or the "P" command to paste the contents of the buffer before the cursor.

Cutting, copying and pasting text is done more traditionally with the "y" command (which stands for "yank", i.e. copy). To copy the current line you use "yy" ("6yy" means copy the current line and the next five lines). To copy to the end of the line use "y$" and to the beginning of the line use "y0". You can join lines with "J".

Searching and replacing is done within the line and within the whole file. Searching within the line is done with "f" from "find" (for example "fa" will move the cursor to the next occurence of "a"). You can also use the substitute command for a line (for example: while in command mode "s:caar/car<ENTER>" means substitute "caar" with "car", and replaces the first occurence of the searched word). The ":#,#s/big/small/g" replaces "big" with "small" within a range of lines (":" starts the command, "#" stands for a line number and "g" stands for "global").
To search the entire file use "/" followed by the searched word and the <ENTER> key. You can go to the next occurrence of the searched word with the "n" (from "next") command. To search and replace over the entire file, i.e. globally, you can use the command ":%s/you/YOU/g" (":" starts the command, "%s" find and substitute, "g" globally).
The search can also be done with options. For example entering ":set ic" will allow finding also combinations with capitals. To disable ignoring case enter ":set noic". You can also enable the "is" option (with the ":set is" command), which can be disabled with ":set nois".

To edit multiple files you can open them together from the beginning (for example "$ vim file1, file2, file3) or you open additional files after you started VIM with the command ":e additional_file". The command "buffers" displays a list of files under editing. You can switch between files with the command ": buffer 1" for example to go to the first file listed by the ":buffers" command, or ":buffer 2" to go to the second file listed by the ":buffers" command. To copy content from one file to another you copy with yank, switch with buffer and paste. This works only if the files were opened together or with the ":e" command. Opening different VIM windows will not allow copying between files in this way. Instead, you have to use <SHIFT><CTRL><C> to copy from one file and <SHIFT><CTRL><V> to paste into another file. The <SHIFT><CTRL><V> command lets you paste text from all kind of sources (from documents, web sites, etc.). To insert an entire file into another you can use the ":r other_file_name_to_be_inserted_into_the_current" command, where "r" stands for "retrieving".

To select text to paste you can also use the visual mode by typing "v" in command mode and then move the cursor to the end of your selection. The selected text will be highlighted. Then press "y" to yank it and then "p" to paste it.

A useful debugging command is the matching parentheses command. You can find out if the corresponding parenthesis or bracket is missing with "%" by moving the cursor on the first parenthesis or bracket and typing "%".

To excute and external command, i.e. a command as if you were in the command line, with the ":! your_command" command. For example the ":! ls <ENTER>" command will list the objects of the working directory.

Finally for getting help in VIM you type ":help <ENTER>". To close the help window use ":q". To find help on a specific command use ":help some_command", for example type ":help user-manual" to get to the user manual. For an interactive tutorial for VIM you can also use the "VIM Tutor" initially created by Pierce and Ware, which can be started in the command line with the command "$ vimtutor <ENTER>". A short introduction in VIM can be also found in Shotts (2009), a lecture which gently introduces readers to the Linux command line in general.

3. Using SBT to create and manage Scala projects

In this section you will find out how to install SBT and use it to create and manage Scala projects.

3.1 Installing SBT

You can use the Scala REPL either directly from within the command-line, initiating a REPL sesssion with the command "$ scala", or, by launching a SBT-session with the command "$ sbt" and then from within SBT launch the Scala REPL using the command "> console". On Ubuntu you can install the Scala language using the apt command ("$ sudo apt install scala") with no additional commands. Check the package management tool you use to get the same result. You can open the Scala REPL and type in some expressions; to exit type ":q" (VIM is useful after all). For guidance, check the book of Jason Swartz, "Learning Scala" for a great introduction to the Scala language.

Installing SBT requires a look at the SBT homepage (http://www.scala-sbt.org/) in order to get the four commands needed to install it using the command line. At the time the present document is written the commands for Ubuntu are:

```
$ echo "deb https://dl.bintray.com/sbt/debian /" | sudo tee -a /etc/apt/sources.list.d/sbt.list
$ sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 2EE0EA64E40A89B84B2DF73499E82A75642AC823
$ sudo apt-get update
$ sudo apt-get install sbt
```

If you are using rpm distributions visit the SBT download page: http://www.scala-sbt.org/download.html.

Next, open a Scala REPL session in SBT using the commands introduced at the beginning of this section and type in some expressions. You exit the console with ":q" and the command "> exit" gets you out of the SBT and back to the command-line.

3.2 Creating a Scala project

In this section we will create a Scala project called Scala_Playground in order to continue the playground series of Linux introductions started by William Shotts (2009) in his book "The Linux Command Line".

Open the terminal with <CTRL-ALT-t>. It starts directly in your home directory. Next, create a directory called Scala_Playground ("$ mkdir Scala_Playground <ENTER>"). Check if the directory was created with the command "$ls <ENTER>" which lists the contents of the directory in which you are placed. Further, switch to the Scala_Playground directory ("$ cd Scala_Playground") and create a part of the inner structure of the current directory by adding another directory called src with its subdirectory main, with its subdirectory scala. This is done with the command "$ mkdir -p src/main/scala" which creates the entire chain of directories. Check it with "$ls" and "$cd" commands, return to the Scala_Playground directory with "cd - ".

Next, in the Scala_Playground directory, create a second directory called project. The ls command should print now for the Scala_Playground two contents: project and src.

The final step in creating the Scala project is to create two files: one directly in the Scala_Playground directory called build.sbt and the second in the Scala_Playground/project directory called build.properties. Open VIM when you are in the Scala_playground/project directory and type in:

```
sbt.version = 0.13.15
```

Go in the command mode with <ESC> and save the changes and give the open file a name: ":w build.properties". Exit vim with ":q".

The build.properties file sets the version of SBT.

Next, when you are in the Scala_Playground directory open VIM and edit the basic settings of the Scala project: project name, Scala version used, and the dependencies you need. For the moment we will ignore dependencies and create the basic Scala project. Type in the file opened in VIM the following two lines:

```
name := "Scala_Playground"
scalaVersion := "2.11.10"
```

The above two lines are not in Scala, they are in SBT's own language.
Now, get into the command line mode with <ESC> and save the changes and give the file the name build.sbt (":w build.sbt"). Exit VIM with ":q".
You should now have three objects in the Scala_Playground directory: the build.sbt file, the project directory and the src directory.

You can now launch SBT while you are in the Scala_Playground directory using the "$ sbt" command. You can see the SBT's output as it uses the information from the two edited files and creates the desired Scala project. When the output is finished you get to see the SBT prompt ">". The project was created successfully. Now, launch the Scala REPL with the command "> console". When the REPL is open you get to see the Scala prompt "scala>". You can type in some expressions to see if it works.
Type ":q" to get back to SBT and "exit" to get back to the command-line.

You can see that the Scala_Playground directory has gained some additional directories: the Scala_Playground/target directory and the Scala_Playground/project/target directory. These are created by SBT when it creates the Scala project.

In the src/main/scala directory you should place your Scala codes in form of scala files.

3.3 Some basic SBT tasks

Open SBT. When you see the SBT prompt ">" type in your first SBT command: "help". This generates an output which you can use to get further information about SBT and about the configuration of your project (command "about", command "settings"). The command "reload" reloads the project in the current directory, which is useful after you've changed the project settings or after you've changed the source files. It does the sanity check as it compiles the source files and outputs the errors encountered in the compilation process, accompanied by messages indicating possible reasons. It is very often helpful to try to read them in order to debug the code.

The command "tasks" is also listed in the help output. The SBT tasks show much of the functions SBT can take over. Type in "tasks" and check the default list of SBT tasks:

- starting the Scala REPL with the command "console"
- compile source files with the command "complile"
- run an application with the command "run"
- tests the code executing all tests
- and many more.

For a detailed introduction in SBT you should check the SBT documentation at http://www.scala-sbt.org/documentation.html and read the book of Joshua Suereth and Matthew Farwell (2016) "SBT in action: the Simple Scala Build Tool".

4. Conclusion

The advantages of learning Scala are manifold (Swartz 2016) and it is not the scope of this paper to cover them all. Though, from my personal point of view, the biggest advantage of Scala is that it allows you, through its functional nature, to express your ideas in a natural manner, and this is very important in developing code for statistical computation. This is why I believe Scala will allow researchers advance their statistical skills by paying more attention to the pseudocode, and not to the details of how to implement it. This paper should offer a starting point for those willing to join this principle.

Acknowledgments:

The author thanks to Jeff Druce and Mike Reposa from CRA for useful learning tips.

Biography:

Roxana Tesileanu graduated Forestry and Environmental Sciences (Dipl. Forstwirtin, the German equivalent to BSc plus MSc in Forestry) in 2008 at Albert-Ludwigs-Universitaet in Freiburg i. Br., Germany, worked as project team member in more international projects (at WSL and Agroscope in Switzerland), and since 2014 as research assistant at INCDS Romania. She is specialized in soil chemistry, environmental statistics and bioinformatics.

Bibliography:

1. Rafeeq Ur Rehman and Christopher Paul. The Linux Development Platform. Pearson, 2003.

2. Roxana Tesileanu. Introduction to Statistical Computing in Scala - an Implementation of the K-Nearest Neighbors Classifier. IEEE Transactions on Neural Networks and Learning Systems. Submitted, under review, 2017.

3. Joshua Suereth and Matthew Farwell. SBT in action: the Simple Scala Build Tool. Manning Publications Co., Shelter Island. 2016

4. Michael C. Pierce and Robert K. Ware. VIM Tutor 1.7.

5. William E. Shotts. The Linux Command Line. 2009

6. Jason Swartz. Learning Scala. O'Reilly. 2009