USING LINUX AS A DEVELOPMENT PLATFORM FOR SCALA PROJECTS

Author: Roxana Tesileanu, INCDS, [roxana.te@web.de](mailto:roxana.te@web.de), August 2017

A) INTRODUCTION

Components of a development system (Rehman, Paul 2003, p.10):

- hardware platform

- operating system

- editors

- compilers and assemblers

- debuggers

- version control system

- bug tracking

For the Scala projects developed on Linux as an operating system, working with SBT condenses the list of the above components, making the process of development more accessible, as it unites different steps in one tool (compiling, building, testing and debugging). Another advantage of using SBT is the opportunity of working interactively in REPL to develop bits of code which can be then inserted in the editor of choice, working from within the Linux command line.

Life cycle of a software development project (Rehman, Paul 2003, p.2):

- requirement gathering

- writing functional specifications

- creating architecture and design documents

- implementation and coding

- testing and quality assurance

- software release

- documentation

- support and new features

Software development "is not just writing code". 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? Functional specifications present the functionality of a software product at an abstract level "defining its input/outpt behavior". Further, 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 you create design documents which define each individual component to the level of functions and procedures. Using the design documents and development tools (SBT, editor) the code is then implemented and tested. Testing involves: 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. The last stage of the life cycle of a software development project is the support and release of new versions depending on requirements.

B) USING VIM AS AN EDITOR FOR THE SCALA CODE

As an editor which can be used from within the command line, VIM can also be adapted to support editing Scala code, after installing the vim-scala package of Derek Wyatt found at https://github.com/derekwyatt/vim-scala. You install vim under Linux using the package tools (dpkg, apt-get, aptitude, rpm, yum, etc.). 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>”.

- vim starts in command mode

- change to insert mode with the "i" key

- to exit insert mode and return to command mode, press the <ESC> key

- save your changes with ":"

- to write the file to the hard drive we use ":w"

- to exit vim type ":wq" to save the changes, or ":q!" to trash all changes

PRESSING <ESC> WILL PLACE YOU IN COMMAND MODE OR WILL CANCEL AN UNWANTED AND PARTIALLY COMPLETED COMMAND!

Move the cursor with "l" (right arrow), "h" (left arrow), "j" down one line, "k" up one line, "0" (to the beginning of the current line), "^" (first non-white space of the line), "$" (the end of the current line, "C^-f" (page down), "C^-b" (page up), "numberG" (to line number, ex: 1G), "G" the last line of the file, "gg" moves to the first line.

Appending text: move the cursor to the end of the line and type "a". This will append text at the end of that line. Or use "A" to move to the end of the line and start appending.

- you can use "o" to insert a line below the current line; and "O" to inser a line above the current line

- undo changes by using "u"

- redo changes by using "C^r"

Deleting text:

- "x" will delete the current character; "3x" will delete the current and the next two characters

- "dw" deletes the word,

- "dd" deletes the current line, "6dd" deletes the current line and the next 5 lines

- "dG" to the end of the file

- "d20G" to the 20th line of the file

- "d$" to the end of the current line

- "d0" to the beginning of the line

Cutting, copying, and pasting text:

- the d commad also cuts text (copies it into a paste buffer) that we can later recall with the p command to paste the contents of the buffer after the cursor or the P command to paste the contents before the cursor

- the "y" command ("yank" - copy) to copy text: "yy" the current line, "6yy" the current line and the next 4 lines, "y$" to the end of the current line, "y0" to the beginning of the line, "yG" from the end of the line to the end of the file, "y20G" from the current line to the 20th line of the file

- the change operator "c" allows you to change from the cursor to where the motion takes you (ce - to the end of the word, c$ to the end of the line).

- joining lines with "J"

Search and replace: move the cursor to locations based on searches. vim can also perform replacements.

- searching within a line: "f" searches a line and moves the cursor to the next instance of a specified character. Ex: "fa" will move the cursor to the next occurence of "a". The search may be repeated by typing a semicolon. You can also use the substitute command for a line: ":s/thee/the <ENTER>" to substitute the FIRST OCCURRENCE of the searched word.

The ":#,#s/old/new/g" command replaces old with new within a range of lines.

- search the entire file: use "/" followed by the searched word and the Enter key and repeat the search using the previous search with the "n" command.

- global search and replace: substitution over a range of lines or the entire file: ":%s/line/LINE/g" (g stands for global, i.e. the subst. is performed on every instance of the search string); to change the file back to its previous form type ":%s/line/LINE/gc"

- you can search using the 'ic' option by entering: ":set ic" (it will find also combinations with capitals). To disable ignoring case enter: ":set noic".

- you can also enable the 'is'(":set is") option which shows partial matches for a search phrase. You can diasble it with ":set nois".

TYPING ":set xxx" SETS THE OPTION 'xxx'.

PREPEND "no" TO SET AN OPTION OFF (":set noxxx")

Editing multiple files:

- vim file1 file2 file3...

- switch betw. files: ":n" and ":N" to switch back

- ":buffers" displays a list of files being edited

- you can perform switching with ":buffer 2" to switch to file 2

Opening additional files:

- to add an additional file use the ":e your\_file\_name" command

- then check the files open with ":buffers"

- you can only switch with "buffer no."

Copying content from one file into another:

- copy with yank, swich with buffer and paste with p

Inserting an entire file into another (retrieving and merging files):

- while you are in the current file use the command ":r other\_file\_to\_be\_inserted\_in\_the\_current"

Printing:

- ":ha" and the file is sent to the default printer

- ":[range]ha[rdcopy][!] [arguments]"

- ":[range]ha[rdcopy][!] > {filename} to print to another file

Matching parentheses:

- find the corresponding parenthesis or bracket with "%" (move the cursor on the first parenthesis or bracket then type %). This command is useful in debugging a program with unmatched parentheses!

Executing an external command:

- type ":!" to place you in command-line mode, then type and execute any external shell command

- for ex. ":! ls <ENTER>" to execute the ls command.

Selecting text to write:

- type v to place you in visual mode, and move the cursor to the end line of your selection. The selected text will be highlighted.

- then press the ":" and at the bottom of the screen ":'<,>'" will appear, this indicatesyou've selected smth.. Then, complete the command with "w TEST <ENTER>", to write the selection to the file named TEST.

You can use the selection in visual mode with other operators too (d deletes the text, y yanks the text).

Replace mode:

- enter the replace mode by typing "R", and leave it with <ESC>. In replace mode every typed character deletes an existing character.

Getting help:

- type ":help <ENTER>"

- type ^W ^W to jump from one window to another and :q<ENTER> to close the help window.

- type ":help some\_command" to find out more on a particular command.

- type ":help user-manual" to get to the user manual.

Auto-completion:

- when starting a ":" command you can use the auto-completion with ^D

C) USING SBT

- installing SBT

- creating a Scala project

- some SBT basic tasks (console, compile, reload, a.o)

C1) 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 commands. 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 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 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 commands. You exit the console with <C^d>and a second <C^d> will get you out of the SBT and back to the command-line (which can also be closed with <C^d>).

C2) 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 in his book "The Linux Command Line".

Open the terminal with <C^-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 vim the following 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 commands to see if it works.

Type <C^d> to get back to SBT and a second <C^d> 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.

C3) 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 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 "SBT in action: the Simple Scala Build Tool".

References:

* The Linux Command Line, William E. Shotts, Jr., 2009
* The Linux Development Platform, Rafeeq Ur Rehman, Christopher Paul, 2003
* VIM Tutor 1.7, Michael C. Pierce, Robert K. Ware (started in the shell with the vimtutor command)
* vim-scala: Vim support for writing Scala code, Derek Wyatt, https://github.com/derekwyatt/vim-scala
* SBT in Action: the Simple Scala Build Tool, Joshua Suereth, Matthew Farwell, 2016
* Learning Scala, Jason Swartz, 2015