Scala in the Terminal

Spring 2013

Contents

Introduction

This guide is meant as a reference guide for using Scala in a terminal to run and compile `.scala` files. In addition, it will cover the use of the REPL in Scala.

1 REPL

The Scala REPL is a powerful tool used from the terminal to execute Scala phrases line-by-line. You can enter the REPL by typing *scala* into a cs department computer terminal or any other terminal on a computer with Scala installed. You should see the following appear:

```
Welcome to Scala version 2.11.0-20130117-095302-d7b59f452f (OpenJDK 64-Bit Server
VM, Java 1.6.0_27).
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

From here you can input Scala phrases and code snippets and have them executed. For example:

```
scala> 1 + 1
```

```
res0: Int = 2

scala> println("It's so cold!")
res1: String = It's so cold!

scala> List[Int](1, 2, 3)
res2: List[Int] = List(1, 2, 3)

scala> val i = "Sulley"
i: String = Sulley

scala> i + " Mike"
res4: String = Sulley Mike
```

It is possible, but very impractical, to input long programs line by line and have them executed one line at a time by the REPL.

1.1 REPL results storage

Notice that lines that evalutate to a value are stored by the REPL in REPL variables. The first variable name is res0, and successive variables are number successively. They can be used as such:

```
scala> 5
res0: Int = 5

scala> 10
res1: Int = 10

scala> res0 + res1
res2: Int = 15
```

1.2 Loading Files

Consider the following file defined in `test.scala`:

```
object test {
    def test1 = 5 + 3

    val sulley = List[String]("strong", "blue", "furry");

}
```

One can load the contents of this file into the REPL using the command :load like this:

```
scala> :load test.scala
```

```
Loading test.scala...
defined module test
test2: Int
sulley: List[String] = List(strong, blue, furry)
```

Now the objects and methods defined in the file can be used in the REPL as though one had simply typed them in.

```
scala> test1
res0: Int = 8

scala> sulley
res1: List[String] = List(strong, blue, furry)

scala> sulley.head
res2: String = strong
```

Use :load to help test and debug your code as you develop programs for CS 18.

1.3 Paste Mode

To avoid needlessly retyping code, the Scala REPL has a handy feature called paste mode. To access it, type

```
scala> :paste
```

Then, your terminal should display the following:

```
// Entering paste mode (ctrl-D to finish)
```

As instructed, paste your code. You can paste as much code as you want. When you are finished type `ctrl-D`.

For example, if you paste the example test object from above, you will see:

```
scala> :paste
// Entering paste mode (ctrl-D to finish)

object test {
    def test1 = 5 + 3

    val sulley = List[String]("strong", "blue", "furry");

}
```

```
// Exiting paste mode, now interpreting.

defined object test
```

2 Running a File Using Scala

Sometimes, using :load to run a file directly takes too long, or is not possible for some other reason. In this case, it is possible to use the *scala* command to run a file directly. Using Scala in this way requires the following:

1. All methods, variables, and values are only inside classes and objects.
2. There exists exactly one main method in the file. That method must have the signature $Array[String] \Rightarrow$ **Unit** and will be the entry point for the program to begin execution.

Note that violation of these conditions will not stop Scala from running the file, but it will stop the main method from being run, which prevents the file from being run as a script.

To run `.scala` files, use the command *scala <filename>* For example, with the following file:

```
object test {
    def printTest = {
        println("Say hello to the Scream Extractor");
    }

    def main(args: Array[String]) : Unit = {
      printTest;
    }
}
```

The following happens in a terminal:

```
$ scala test.scala
Say hello to the Scream Extractor
```

3 Compiling and Running Scala Programs

Compiling is the process of transforming code from one language to another. In this case, we want to transform our Scala code into Java bytecode so that it can be run more quickly. If we do not compile first, Scala will have to compile our code every time we run it, which will take a very long time. In order to compile code before running it, we use something called a *compiler*. The compiler for Scala is called scalac. For a Scala file to be valid for compilation, it must have all methods,

variables, and values inside objects or classes. The command to run the Scala compiler is: *scalac <filename>* If there are no errors, you should recieve another prompt on the terminal. Now, if you list the files in the directory, you should see one or more `.class` files.

```
$ ls
test.scala
$ scalac test.scala
$ ls
test$.class  test.class  test.scala
```

After compiling you can now run the main methods from individual objects that were in the file. This notation is similar to running a file, but notice there is no filename, just an objectname. The notation is `scala <objectname>`

Note: There is no `.class` or `.scala` extension, simply type the object name

3.1 Example

Suppose we add the following at the bottom of the previous file:

```
object test2 {
  def main(args: Array[String]) : Unit = {
    println("test2");
  }
}
```

Here is how we would compile the file `test.scala` and run the main methods in the objects *test1* and *test2*

```
$ scalac test.scala
$ scala test
Say hello to the Scream Extractor
$ scala test2
test2
```

3.2 Multi-File Programs

Some programs, including many which you will write this semester, are split amongst multiple files to help with organization. To compile and run multiple files, simply pass all the necessary files to scalac as command line arguments at the same time like so:

```
scalac <file1> <file2> <file3> <file4> ...
```

Alternatively, if you want to compile all of the files ending in ".scala" you can run `scalac *.scala` in the terminal. The * is used to say that every file matching the given pattern should be used. For example, `scalac LL*.scala` would compile all files in the current directory that begin with "LL" and end with ".scala."

3.3 Packages

A package is an incredibly useful tool that can be used to organize your `.class` files. A package declaration looks like **package** <*packagename*> and usually occurs at the beginning of a file. When a `.scala` file is compiled, all the `.class` files associated with a package are placed in a folder of that package name. For example, let's add a package to our `test.scala` by adding `package randall` to the top of the file.
Compiling this file looks as such:

```
$ ls
test.scala
$ scalac test.scala
$ ls
randall/  test.scala
```

Remember that when we compiled earlier, all of the `.class` files appeared in the same directory as the `.scala` files after compilation. Now, these files are instead stored in the `randall` directory. In order to run comiled files stored in a package, we type `scala <packagename>.<objectname>`. Note that it is not necessary to descend into the package directory. Rather, Scala automatically searches the directories below the current one for the correct object when looking for a qualified path that results from using packages. The results of the example are:

```
$ scala randall.test
Say hello to the Scream Extractor
$ scala randall.test2
test2
```

4 Class Path

When you run Scala from the command line, either to run a file or just to activate the REPL, Scala can only look for files in a specified list of places. This list of places is called the **Classpath**. To add to the classpath when you run Scala, use the notation `scala -cp <path> ...`. A commonly added classpath is `.`, which indicates the current directory. For example: `scala -cp . test` tells Scala to look for an object named test in the current directory (in addition to the other places where it normally looks). This usage of -cp is so common, that we have a version of Scala that automatically adds . to the classpath for you. When working on Scala at home, be aware that you will have to add the current directory to your classpath.

Please let us know if you find any mistakes, inconsistencies, or confusing language in this or any other CS18 document by filling out the anonymous feedback form:

`http://cs.brown.edu/courses/cs018/feedback`.