Notes

Scala 1

Packages 1

Testing 2

Pattern Matching 2

Links 2

Variable Binding 2

Sealed 3

Match Sequences 3

Guards 3

Patterns to pick out arguments 4

Emacs 4

Highlighting 4

Macros 5

Undo 5

Git 5

Kafka 8

SBT 8

# Scala

## General

“def” = evaluate at time of call

“val” = evaluate at time of definition

“lazy val” = evaluate when evaluated the first time

## Access

From <http://www.ibm.com/developerworks/library/j-scala07298/>

“you can use import anywhere inside the client Scala file, not just at the top of the file and correspondingly, will have scoped relevance”

“use import to bring not just nested types into lexical scope, but any member”

“import can take multiple, comma-separated targets”

“Scala:

* Uses "public" by default
* Specifies "private" to mean "accessible only to this scope"

By contrast, "protected" is definitely different from its counterpart in Java code; where a Java protected member is accessible to both subclasses and the package in which the member is defined, Scala chooses to grant access only to subclasses. This means that Scala's version of protected is more restrictive (although arguably more intuitively so) than the Java version.”

“access modifiers in Scala can be "qualified" with a package name, indicating a level of access *up to* which the member may be accessed”

“the *object-private* specification, illustrated by private[this], which stipulates that the member in question can only be seen by members called on that same object, not from different objects, even if they are of the same type”

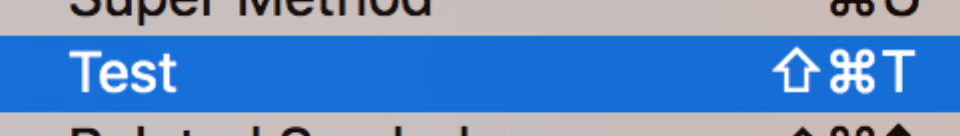
## Testing

Add the following dependency:

“org.scalatest” % “scalatest\_2.10” % “2.1.0” % “test”

to build.sbt and let Idea load the library.

Use Idea to create a test by using “Navigate -> Test”



Your test class should extend org.scalatest.FunSuite and you can do this automatically by selecting ScalaTest as the Testing Library.



## Pattern Matching

### Links

<http://www.artima.com/pins1ed/case-classes-and-pattern-matching.html>

### Variable Binding

“A variable pattern matches any object, just like a wildcard. Unlike a wildcard, Scala binds the variable to whatever the object is. You can then use this variable to act on the object further.”

case Some(x) => println(“I got x: ” + x)

“In addition to the standalone variable patterns, you can also add a variable to any other pattern. You simply write the variable name, an at sign (@), and then the pattern. This gives you a variable-binding pattern. The meaning of such a pattern is to perform the pattern match as normal, and if the pattern succeeds, set the variable to the matched object just as with a simple variable pattern.”

case x @ Some(\_) => println(“I got the entire Option object, not just what is inside: ” + x)

“A constructor pattern looks like

case BinOp("+", e, Number(0)) => println(“good”)

It consists of a name (e.g. BinOp) and then a number of patterns within parentheses (e.g. "+", e, and Number(0)). Assuming **the name designates a case class**, such a pattern means to first check that the object is a member of the named case class, and then to check that the constructor parameters of the object match the extra patterns supplied.”

### Sealed

When using the “match” statement, you must supply every possible match pattern because unlike Java’s switch statement, there is no “default” fall through statement. You would usually do this on your own by specifying a pattern that matches anything at the end.

case \_ => println(“matches everything and anything”)

“In fact, you can enlist the help of the Scala compiler in detecting missing combinations of patterns in a match expression. To be able to do this, the compiler needs to be able to tell which are the possible cases; make the superclass of your case classes **sealed**. A sealed class cannot have any new subclasses added except the ones in the same file.”

**sealed** abstract class Expr

  case class Var(name: String) extends Expr

  case class Number(num: Double) extends Expr

### Match Sequences

Sequence patterns can be thought of as a special case to constructor patterns for case classes.

“You can match against sequence types like List or Array just like you match against case classes. Use the same syntax, but now you can specify any number of elements within the pattern.”

case **List(0, \_, \_)** => println("found it")

This matches any list that starts with zero and has a length of three. To match an arbitrary length list you need to use “\_\*”

case List(0, \_\*) => println("found it")

### Guards

“Scala restricts patterns to be linear: a pattern variable may only appear once in a pattern.”

case BinOp("+", x, x) => BinOp("\*", x, Number(2))

This will fail to compile because you used “x” twice in the pattern. But you really want the two arguments to be the same or else you can’t just assume that on the right side of the case statement. That is, if you use

case BinOp("+", x, y)

That will match any two arguments and they don’t have to be the same. To resolve this, add an “if” (before the arrow) to guard against matches that succeed according to your pattern but is not exactly what you wanted.

case BinOp("+", x, y) **if x == y** => BinOp("\*", x, Number(2))

### Patterns to pick out arguments

If you already have a value defined like this

val myTuple = (123, "abc")

And you would like to draw out the data from the value, you can use a pattern like this to assign the data in the tuple to other values.

val (number, string) = myTuple

Now the values “number” and “string” have values from the tuple, 123 and “abc”, respectively.

You can do this with any constructor and not just with tuples

val exp = new BinOp("\*", Number(5), Number(1))

val BinOp(op, left, right) = exp

Now, the values op, left, and right have the data “\*”, Number(5), and Number(1), respectively. A nice way to remember this is to think that the pattern is “de-constructing” the contructor.

# Emacs

## Highlighting

<http://stackoverflow.com/questions/18090378/turn-on-background-color-when-highlighting-with-c-spc-on-a-mac>

M-x transient-mark-mode to toggle the highlighting

## Macros

c-x, ( to start

c-x, ) to end

f4 to execute

## Undo

c-\_

# Git

<https://git-scm.com/docs>

<http://ftp.newartisans.com/pub/git.from.bottom.up.pdf>

Make your own local repository by copying from a remote:

git clone -o MY\_NAME\_FOR\_REMOTE -b BRANCH <https://github.com/REPO.git> MY\_DIR\_TO\_WRITE\_TO

Or if you grabbed the remote without the branch information like this

git clone -o REMOTE\_NAME URL DIRECTORY

You can fetch the branch and switch to it with this

git fetch REMOTE\_NAME BRANCH\_NAME

git checkout BRANCH\_NAME

Ask for a list of references on any remote repository

git ls-remote REMOTE\_NAME\_OR\_URL

If you're in a git repository that you cloned, you can leave off the REMOTE\_NAME\_OR\_URL and it'll default to the remote you cloned from.

References are "alias" to sha1 commit ID numbers.

If you're inside a git directory, you can list the remotes your local git is tracking (most likely from cloning, pushing, or pulling)

git remote

If there isn't one listed that you want, you can add it with

git remote add WHATEVER\_NAME URL

After adding a remote, you can fetch and merge it into your repo

Fetch references (branches, tags, updated objects, etc.) from a remote repo to your repo

git fetch REMOTE\_NAME\_OR\_URL

Fetching will get all the changes in references from the remote but it will not touch files in your working directory. You can merge the changes yourself after the fetch with:

git merge REMOTE/BRANCH

merge will merge and commit to your local branch unless you run it with --no-commit option.

This tells me what remote branches I am tracking in my local repo and where it'll push and pull from

git remote show REMOTE\_NAME

List your remotes

git remote -v

Change which the URL of your remotes

git remote set-url REMOTE\_NAME NEW\_URL

Rename your remote

git remote rename OLD\_NAME NEW\_NAME

This tells me which local branch I am currently using

git branch -vv

If I want to add another remote for my local to track:

git checkout --track REMOTE/BRANCH\_IN\_REMOTE

(above) will create a local branch for me and then I can switch to it in my local with

git checkout BRANCH

Once I am in my local branch I can reconfigure where it pushes and pulls from

git branch -u REMOTE/SOME\_OTHER\_BRANCH\_IN\_REMOTE

One of the more helpful options is -p, which shows the difference introduced in each commit. You can also use -2, which limits the output to only the last two entries:

git log -p -2

List the branches you have

git branch

If you don’t have the branch locally yet, get it with this

git checkout -t REMOTE\_NAME -b BRANCH\_NAME

Synching your forked repo with the original

1. Make sure you added the original as a remote (assume you named it ‘origin’)
2. Fetch the original remote/branch (creates a new branch locally for you)
3. Switch over to your forked branch
4. Merge in changes from the original branch

git remote add YOUR\_ORIGIN\_REMOTE\_NAME YOUR\_ORIGIN\_REMOTE\_URL

git fetch YOUR\_ORIGIN\_REMOTE\_NAME

git checkout YOUR\_FORK\_BRANCH

git merge YOUR\_ORIGIN\_REMOTE\_NAME/YOUR\_ORIGIN\_BRANCH\_NAME

The above will synch up your local repo but if you want to update the branch that you’re on, then push the merged changes from your local to your remote.

git push

Merge from the upstream master to your local so that you can commit it back into your forked remote. Note that by default, merge will commit to your local repo if there are no merge conflicts. If there are, you will need to resolve them by hand and then commit them into your repo. Or you can abort the merge and git will return merged files back to the way it was before merging although it may not work all the time, so it is best to only do merges when you have already committed everything to your repo.

git merge upstream/master

List what is checked into your repo

git ls-files

Revert a file back to is checked in

git checkout FILE

Remove files from git (not from your working directory)

git rm FILE

Doing a diff on a previous version

git diff HEAD~1:FILE\_PATH YOUR\_FILE\_PATH

For example, diffing what I have locally in my BDFD-284 branch in the src directory to what is in second look master

git diff BDFD-284:src upstream/master:src

Delete a branch on a remote

git push REMOTE\_NAME --delete BRANCH\_NAME

# Kafka

Create a topic

/Applications/kafka/bin/kafka-topics.sh --create --zookeeper localhost:2181 --partitions 1 --replication-factor 1 --topic mytopic

List all topics

/Applications/kafka/bin/kafka-topics.sh --list --zookeeper localhost:2181

Write to topic

/Applications/kafka/bin/kafka-console-producer.sh --broker-list localhost:9092 --topic mytopic

Read from topic

/Applications/kafka/bin/kafka-console-consumer.sh --zookeeper localhost:2181 --topic mytopic --from-beginning

# SBT

The % method is used to construct an Ivy module ID from strings

Keys have an overloaded method called in used to set the scope. The argument to in can be an instance of any of the scope axes.

For example, set the name scoped to the Compile configuration

name in Compile := "hello"

it’s important to understand that in and := are just methods, not magic. Scala lets you write them in a nicer way, but you could also use the Java style:

name.in(Compile).:=("hello")

To change the value associated with the compile key, you need to write compile in Compile or compile in Test. Using plain compile would define a new compile task scoped to the current project, rather than overriding the standard compile tasks which are scoped to a configuration.

Assignment with := is the simplest transformation, but keys have other methods as well. If the T in SettingKey[T] is a sequence, i.e. the key’s value type is a sequence, you can append to the sequence rather than replacing it.

* += will append a single element to the sequence.
* ++= will concatenate another sequence.

Unmanaged dependencies work like this: add jars to lib and they will be placed on the project classpath.

Dependencies in lib go on all the classpaths (for compile, test, run, and console).

There’s nothing to add to build.sbt to use unmanaged dependencies, though you could change the unmanagedBase key if you’d like to use a different directory rather than lib.

sbt uses Apache Ivy to implement managed dependencies

you can simply list your dependencies in the setting libraryDependencies

The key “libraryDependencies” is a settings key of a sequence (aka list) of “module IDs”

Module ID objects are created with the “%” method and can be chained together like this:

libraryDependencies += groupID % artifactID % revision % configuration

Using ivy, sbt will download your dependencies from either its standard list of repositories or ones you described and store them in the “.ivy2” directory in your home directory. The “update” task will do this. The “compile” task depends on “update”

Of course, you can also use ++= to add a list of dependencies all at once:

libraryDependencies ++= Seq(  
  groupID % artifactID % revision,  
  groupID % otherID % otherRevision  
)

resolvers does not contain the default resolvers; only additional ones added by your build definition.

sbt combines resolvers with some default repositories to form externalResolvers.

Therefore, to change or remove the default resolvers, you would need to overrideexternalResolvers instead of resolvers.

If you want a dependency to show up in the classpath only for the Test configuration and not the Compile configuration, add % "test" like this:

libraryDependencies += "org.apache.derby" % "derby" % "10.4.1.3" % "test"

If your project is in directory hello, and you’re adding sbt-site plugin to the build definition, create hello/project/site.sbt and declare the plugin dependency by passing the plugin’s Ivy module ID to addSbtPlugin:

addSbtPlugin("com.typesafe.sbt" % "sbt-site" % "0.7.0")

If you’re adding sbt-assembly, create hello/project/assembly.sbt with the following:

addSbtPlugin("com.eed3si9n" % "sbt-assembly" % "0.11.2")

Not every plugin is located on one of the default repositories and a plugin’s documentation may instruct you to also add the repository where it can be found:

resolvers += Resolver.sonatypeRepo("public")