Course Management - Workflow and tools

Introduction

Over the last couple of years, a specific set of tools and workflow were used to maintain the exercise source code for Lightbend training courses. Based on the experience gained over time, a new approach has been devised that meets the following objectives:

- Maintain a regular history of course versions in git
    - Easy retrieval of current and older version of a course and allow for *normal* handling of pull requests
- Ability to linearize and de-linearize (using the `linearize` and `delinearize` commands respectively) a version of a course exercise master. A linearized course master can be changed using `git` interactive rebasing
- The ability to run *all* tests for *all* exercises in a course master repository
- Generation of self-contained student exercise repositories using `studentify`
    - The student repository has no external dependencies except:
        - Java (8) SDK
        - sbt (Scala build tool, version 0.13.13 or higher)
        - dependencies defined as part of the course master itself
    - Ability to *save* the current state of an exercise with the possibility to restore it at a later moment
    - Ability to *pull* the complete solution of an exercise
    - Ability to selectively *pull* source files from the complete solution of an exercise
    - Generalization of the use of *manual pages* for all exercises
    - Support for (*Akka*) *multi-jvm* tests
    - Ability to *list* all the exercises in a master repository
    - Ability to *jump* to an exercise based on its sequence number
    - A brief (13') demo of the student repo functionality is available here

Note: the course management tools (currently three tools: `studentify`, `linearize` and `delinearize`) have been verified to run on a *nix system (MacOS). No effort was made to make these Windows 'compatible'. If there's a need to run the tools on Windows®, Windows® 10 now has an integrated `bash` shell which should be sufficient to get the tools working. Do note that a `studentified` version of a course should run on *nix and Windows 10 systems.
Note: Testing has revealed that some 'older' version of `git` pose problems. `git` version 2.10.0 should be fine.

Course master repository structure

Getting started

The following section details the structure of a course master project.

To give anyone who wants to use this approach a leg-up, a template project is available here. Next:

- Clone the project
- You probably may want to *'re-git'* it:
  - `rm -rf .git`
  - `git init`
- ... and start hacking!

Course master set-up

A course master repository is a multi-project sbt build that needs to adhere to a few conventions:

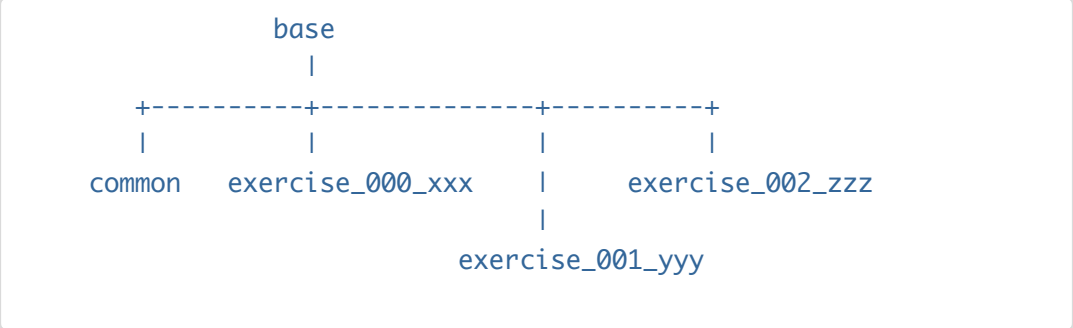
- Each exercise is an sbt project who's name has to start with `exercise_[0-9][0-9][0-9]` followed by a description of the exercise (for example `exercise_001_create_a_class`)
- There should be an `sbt` project containing *common* code with the exact name `common`
- The `build.sbt` file in the root folder of the master repository uses a fixed layout described in more detail below
- There should be a `.sbtopts` file in the root folder of the master repository which sets a number of options that are important when running integration tests
- README files:
  - There should be a global `README.md` file containing the overall course

description for students and a list of `sbt` commands that the user can use to 'navigate' the course.

- Each exercise project should have a `README.md` file containing a description of the exercise and 'run', 'test' and 'next step' instructions and should be located at `exercises_xxx_.../src/test/resources`
- The `common` project should have a `README.md` file located under `common/src/test/resources`
- The `base` project should have a `README.md` file located under `src/test/resources`
- Note that the `listExercises` command, available on a *studentified* repo extracts the exercise descriptions from the exercise project name. For example, the exercise project name `exercise_001_create_a_class` generates `1. Exercise 1 > Define A Class` in the output of the command.

Course master project structure

The following diagram depicts structure of a course master with 3 exercises:



Project `base` aggregates *all* projects below it (`common` and all exercise projects)

Furthermore, all exercise projects depend on `common` .

The layout of the `build.sbt` file is quite simple as illustrated for the sample structure in the previous diagram:

```
lazy val base = (project in file("."))
  .aggregate(
    common,
    exercise_000_xxx,
    exercise_001_yyy,
    exercise_002_zzz,
  )
  .settings(CommonSettings.commonSettings: _*)
```

```
lazy val common = project.settings(CommonSettings.commonSettings:
_*)

lazy val exercise_000_xxx = project
  .settings(CommonSettings.commonSettings: _*)
  .dependsOn(common % "test->test;compile->compile")

lazy val exercise_001_yyy = project
  .settings(CommonSettings.commonSettings: _*)
  .dependsOn(common % "test->test;compile->compile")

lazy val exercise_002_zzz = project
  .settings(CommonSettings.commonSettings: _*)
  .dependsOn(common % "test->test;compile->compile")
```

The attentive reader will notice references to `CommonSettings.commonSettings`. It is these settings that will allow for the definition of all project specific settings. ***All*** project specific settings should be put under the `project` folder.

It is recommended to use the following file structure in the `project` folder:

```
./
 |
 +- build.sbt
 |
 +--/project
       |
       +- AdditionalSetting.scala
       |
       +- build.properties
       |
       +- CommonSettings.scala
       |
       +- CompileOptions.scala
       |
       +- Dependencies.scala
       |
       +- Navigation.scala
       |
       +- Man.scala
       |
       +- plugins.sbt
       |
```

```
                +- StudentCommandsPlugin.scala
                |
                +- StudentKeys.scala
```

When settings up a new course master repository, the easiest way to get started is to follow the instructions in the *Getting started* section above.

> Note: plan is to add a tool in the course management tool set to automatically add these files **and** generate the `build.sbt` file based of the project folder layout automatically

Course master editing approach

Once the initial set-up of a course master repository has been completed, the question arises about how to evolve it.

There's a recommended workflow and a set of tools that can be used for this.

Tools

The course management tools contain two utilities that can convert a multi-project course master project with one project per exercise into a so-called '*linearized*' git repository with one commit per exercise. A second command named 'delinearize' performs the opposite conversion: it applies the changes made in the linearized version of the course master on the course master repo itself.

Hence, one can choose different approaches to implement a certain modification to the exercises in the course master.

Applying changes to common files

The simplest approach to changing any of the source files in the `common` project, files in the root folder or in the `project` folder, is to apply them directly to these files.

When the changes have been made, it is very easy to verify if all the tests in the different exercises still pass. In an `sbt` session, this can be done by running the `base/test:test` command.

Applying changes to exercise projects

Two possible approaches can be utilized.

First approach: direct changes to files in course master

One can apply changes directly to files in the exercise project(s) and verify

correctness by running, possibly modified, tests. (run `base/test:test` in sbt). In most cases, any change made in a particular exercise, will have an impact on subsequent exercises. As such, making changes implies being able to efficiently search for occurrences of certain classes, methods and variable names. A very nice tool that can assist in this process is the 'Silver Searcher' (https://github.com/ggreer/the_silver_searcher). It's basically a `find` / `grep` / `awk` on steroids.

Combined with some simple scripting, many changes can be implemented very efficiently. A video recording showing this approach can be viewed at: course master editing workflow.

Of course, the (best) practice to commit often in `git` applies here.

Second approach: applying changes to a linearized version of the course master

In some cases, applying changes to a linearized version of a course master repo may be easier than applying them directly on the master.

Suppose we have a course master repo and an empty folder that will hold the linearized version of the master repo. Suppose that these are located in folders `/lbt/FTTAS-v1.3.0/fast-track-akka-scala` and `/lbt/Studentify/as-linearized` respectively.

The editing workflow looks as follows:

1. Linearize the master repo: `linearize /lbt/AS-v1.2.0/fast-track-scala-advanced-scala /lbt/Studentify/as-linearized`

   > Note: always make sure that, when running `linearize`, the `workspace` and `index` in the course master repository is clean: any modifications in the `index` and `workspace` will not be carried over to the linearized repo.

2. Apply changes to the linearized repo in `/lbt/Studentify/as-linearized/fast-track-scala-advanced-scala` using git interactive rebasing (e.g. `git rebase -i --root`)

   > Note: When `git` gives you the possibility to change the commit message, **don't change it**. Any change to a commit message will result in `delinearize` refusing to do its job. Also, don't add or delete commits in the linearized repo.

3. Test the modified exercise(s) as far as possible in the linearized git repo

4. Apply the changes made in the previous step by delinearizing the linearized repo back on the course master: `delinearize /lbt/AS-v1.2.0/fast-track-scala-advanced-scala /lbt/Studentify/as-linearized/fast-track-scala-advanced-scala`

   > Note: always make sure that, when running `delinearize`, the `workspace` and `index` in the course master repository is clean. If this is not the case, modifications in the `index` and `workspace` may be silently overwritten.

5. Run **all** tests on the master repo: `base/test:test`.

6. If the tests pass, commit the changes on the master repo. If they don't, reset the `git workspace` / `index` to the last commit that was 'ok'

7. Repeat this process as often as necessary by repeating the process from step 2 onward.

> Note1: consider making many 'small' changes that are delinearized and commited. Once a successful result is obtained, the linearized repo should be discarded, and, if desired, the sequence of commits that were made during the repetitive execution of this process can be squashed into one or a limited number of commits
> Note2: even though the 'common' (project `common`, `project/*`) content will appear in the delinearized repo, ***don't change them in the linearized repo*** as any change will **not** be brought back to the master repo during delinearization. Apply such changes directly on the course master.

Combining approaches

Of course, the two approaches described above can be combined repeatedly and in different combinations in a workflow. However, when a linearized repo exists and subsequently changes are made to the master repo, the linearized version should be discarded and re-created using `linearize`.

Selecting a particular *exercise* (sbt project)

It is recommended to use the regular `project` and `projects` `sbt` commands.

`projects` will list all projects in the build. For a course master repo, it will show project `base`, `common` and all of the exercises.

Selecting a particular exercise or project can be done with the `project` command as shown here:

```
[ericloots@Eric-Loots-MBP] $ sbt
 .
 . <elided>
 .
man [e] > akka-collect > initial-state > projects
[info] In file:/Users/ericloots/Trainingen/LightbendTraining/AAS-
v1.4.0/advanced-akka-with-scala/
[info]     base
[info]     common
[info]   * exercise_000_initial_state
[info]     exercise_001_complete_initial_state
[info]     exercise_002_remoting
[info]     exercise_003_cluster_events
[info]     exercise_004_cluster_aware_routers
[info]     exercise_005_cluster_singleton
[info]     exercise_006_cluster_sharding
[info]     exercise_007_persistent_actors
[info]     exercise_008_data_replication

man [e] > akka-collect > initial-state > project
exercise_004_cluster_aware_routers
[info] Set current project to exercise_004_cluster_aware_routers (in
build file:/Users/ericloots/Trainingen/LightbendTraining/AAS-
v1.4.0/advanced-akka-with-scala/)

man [e] > akka-collect > cluster-aware-routers > project common
[info] Set current project to common (in build
file:/Users/ericloots/Trainingen/LightbendTraining/AAS-
v1.4.0/advanced-akka-with-scala/)
LightbendTraining/AAS-v1.4.0/advanced-akka-with-scala/)

man [e] > akka-collect > initial-state > project
exercise_007_persistent_actors
[info] Set current project to exercise_007_persistent_actors (in
build file:/Users/ericloots/Trainingen/LightbendTraining/AAS-
v1.4.0/advanced-akka-with-scala/)

man [e] > akka-collect > persistent-actors >
```

Creating student repositories

As mentioned before, the `studentify` command can be used to generate a self-contained repository.

The `studentify` command has a few options to customize the generated

repository.

First of all, one can generate a student repo that contains a subset of the exercises available in the course master repo.

This is done by using the `-fe` (first exercise) and `-le` (last exercise) options. Either of them can be omitted resulting respectively in selecting all exercises up-to the last exercise or from the first exercise.

There's also the `-sfe` option that allows one to 'bookmark' an exercise in the generated student repo. When the student runs `sbt`, he/she will be positioned at the selected exercise.

These three options come in handy when a course master contains exercises for more than one course (e.g. `Fast Track to Scala` and `Advanced Scala`)

Finally, there's the `-mjvm` option that will generate a `build.sbt` file that support `Akka`'s `multi-jvm` testing.

> Note: Course master repos that use `multi-jvm` should include the dependencies required for this feature (see the `Advanced Akka with Scala` course for an example).

Validating student repositories

While all the tests for a project can be run in the Master project, it is also valuable to verify that all the tests still run correctly once the repository has been converted to the student version. This can be accomplished using the `validateStudentRepo.sh` script. To use this script you simply run it and pass in the path to the student repo you want to validate. For example: `./validateStudentRepo.sh ../FTTS-fast-track-scala`.

When run, this script will advance through each exercise in the student repo. It will pull the solution and run all the tests against that solution to verify that they work. It completes once it reaches the final exercise.

Creating Releases

When you are ready to release your student repo into the wild, you can create a versioned zip file to distribute. This zip file is easily created using the `createRelease.sh` script. To use this script you run it, passing in a path to the master repo that you want to release. This script will generate the student repo, validate it using the `validateStudentRepo.sh`, package it into a zip file, and attach a version number to that zip. It also embeds a file in the zip that contains the version number. This zip file is now ready to distribute.

By default, the version used is "SNAPSHOT". You can specify your own version

by using the `-v <Version>` option.

So to release Fast Track to Scala version 2.0.0 you would run a command such as: `./createRelease.sh -v 2.0.0 ../FTTS-fast-track-scala`

It is important to note that the final zip name is determined by the name of the repo that you pass in. So in the above scenario, the final zip generated would be `FTTS-fast-track-scala-exercises-2.0.0.zip`

Appendix 1 - Course management tools summary

studentify

`studentify` generates a *student* repository from a given course master repository.

A student will clone a copy of the generated student repository, load it in his/her favorite IDE (IntelliJ or Scala IDE (Eclipse)).

Using *sbt*, the student can move between exercises by issuing the `nextExercise`, `prevExercise` and `gotoExerciseNr` commands. Please note that these commands pull only exercise tests and do NOT change other (solution) sources. There is the `pullSolution` command that pulls the solution code.

Detailed, per-exercise instructions can be obtained via the `man e` command.

General course instructions can be obtained via the `man` command.

The `listExercises` command will generate a list of exercises and their description.

Requirements:

- a master repository and its file-path
- the path to an (empty) folder in which the student distribution will be created

Invocation

```
studentify 1.x
Usage: studentify [options] masterRepo out

  masterRepo               base folder holding master course
repository
  out                      base folder for student repo
```

```
  -mjvm, --multi-jvm       generate multi-jvm build file
  -fe, --first-exercise <value>
                           name of first exercise to output
  -le, --last-exercise <value>
                           name of last exercise to output
  -sfe, --selected-first-exercise <value>
                           name of initial exercise on start
```

Example

```
./studentify  /lbt/Studentify/aas-base/advanced-akka-scala \
              /lbt/Studentify/aas-out -mjvm
```

In the above example, a folder `fast-track-akka-scala` will be created under the `out` folder.

> Note: `studentify` will copy over a number of files verbatim from the master build definition and overwrite or create a number of files in the student repo. A diagram, `images/sbtSourceCode.png` provides more details on this.

linearize

`linearize` will generate a new git project in which every exercise is a commit in the project's history.

This repo can then be utilized to apply changes to the exercises via interactive rebasing.

Invocation

```
linearize 1.x
Usage: linearize [options] masterRepo linearRepo

  masterRepo          base folder holding master course repository
  linearRepo          base folder for linearized version repo
  -mjvm, --multi-jvm  generate multi-jvm build file
  -f, --force-delete  Force-delete a pre-existing destination folder
```

Example

```
linearize /lbt/FTTAS-v1.3.0/fast-track-akka-scala \
          /lbt/Studentify/fttas-linearized
```

Note that if the destination folder exists, `linearize` will abort. Two options are available: either delete the pre-existing destination folder and re-run the command, or, specify the `-f` command line option: `linearize` will delete the folder (and its contents) before proceeding.

In the above example, a folder `fast-track-akka-scala` will be created. This folder contains a git repository containing all the course exercises with one commit per exercise. Note that each commit contains an sbt multi-project build. More precise, two projects are defined, `common` and `exercises`.

```
[ericloots@Eric-Loots-MBP] $ cd /lbt/Studentify/fttas-
linearized/fast-track-akka-scala
[ericloots@Eric-Loots-MBP] $ git log --oneline
fe111ab exercise_021_fsm
64f1307 exercise_020_akka_extension
8c4bc1a exercise_019_use_ask_pattern
59e16ea exercise_018_become_stash
67206d5 exercise_017_config_dispatcher
4d900a7 exercise_016_use_router
5e90ed0 exercise_015_detect_bottleneck
6424eb2 exercise_014_self_healing
c7e0f8c exercise_013_another_faulty_actor
0ab3f71 exercise_012_custom_supervision
30c10a1 exercise_011_faulty_actor
72e9cf4 exercise_010_lifecycle_monitoring
6accfc2 exercise_009_stop_actor
3140ad7 exercise_008_keep_actor_busy
3372ff7 exercise_007_use_scheduler
cea8563 exercise_006_actor_state
e890cc4 exercise_005_create_child_actors
4684cac exercise_004_use_sender
d95dae6 exercise_003_message_actor
fc818b7 exercise_002_top_level_actor
03bd482 exercise_001_implement_actor
9b00254 exercise_000_initial_state
```

This repository is well suited to apply changes to exercises using interactive rebasing.

Note

In each commit, apply changes to files in the `exercises` project. **Important**: Any changes applied on files outside of the `exercises` folder will be discarded when the repo is `delinearized`. If changes need to be applied outside of the `exercises` folder, apply them directly on the course master instead.

Example course editing flow

Let's apply a small change to the course. Let's add a text file named `SampleTextFile.txt` under `exercises/src` in exercise `exercise_009_stop_actor` and remove it again in exercise `exercise_016_use_router`.

```
[ericloots@Eric-Loots-MBP] $ git rebase -i --root
   pick 9b00254 exercise_000_initial_state
   pick 03bd482 exercise_001_implement_actor
   pick fc818b7 exercise_002_top_level_actor
   pick d95dae6 exercise_003_message_actor
   pick 4684cac exercise_004_use_sender
   pick e890cc4 exercise_005_create_child_actors
   pick cea8563 exercise_006_actor_state
   pick 3372ff7 exercise_007_use_scheduler
   pick 3140ad7 exercise_008_keep_actor_busy
   edit 6accfc2 exercise_009_stop_actor
   pick 72e9cf4 exercise_010_lifecycle_monitoring
   pick 30c10a1 exercise_011_faulty_actor
   pick 0ab3f71 exercise_012_custom_supervision
   pick c7e0f8c exercise_013_another_faulty_actor
   pick 6424eb2 exercise_014_self_healing
   pick 5e90ed0 exercise_015_detect_bottleneck
   edit 4d900a7 exercise_016_use_router
   pick 67206d5 exercise_017_config_dispatcher
   pick 59e16ea exercise_018_become_stash
   pick 8c4bc1a exercise_019_use_ask_pattern
   pick 64f1307 exercise_020_akka_extension
   pick fe111ab exercise_021_fsm

   # Rebase fe111ab onto 99ef9f2
```

We apply the change:

```
[ericloots@Eric-Loots-MBP] $ touch exercises/src/SampleTextFile.txt

[ericloots@Eric-Loots-MBP] $ git add -A

[ericloots@Eric-Loots-MBP] $ git rebase --continue

[ericloots@Eric-Loots-MBP] $ rm exercises/src/SampleTextFile.txt

[ericloots@Eric-Loots-MBP] $ git add -A

[ericloots@Eric-Loots-MBP] $ git rebase --continue
```

After applying the change, we `delinearize` the project.

```
delinearize /lbt/FTTAS-v1.3.0/fast-track-akka-scala
/lbt/Studentify/fttas-linearized/fast-track-akka-scala
```

Let's see that that did to the (clean) master course repo.

```
[ericloots@Eric-Loots-MBP] $ cd FTTAS-v1.3.0/fast-track-akka-scala/

[ericloots@Eric-Loots-MBP] $ git st
On branch master
Your branch is up-to-date with 'origin/master'.
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        exercise_009_stop_actor/src/SampleTextFile.txt
        exercise_010_lifecycle_monitoring/src/SampleTextFile.txt
        exercise_011_faulty_actor/src/SampleTextFile.txt
        exercise_012_custom_supervision/src/SampleTextFile.txt
        exercise_013_another_faulty_actor/src/SampleTextFile.txt
        exercise_014_self_healing/src/SampleTextFile.txt
        exercise_015_detect_bottleneck/src/SampleTextFile.txt

nothing added to commit but untracked files present (use "git add"
to track)
```

delinearize

`delinearize` does the opposite of `linearize`.

Invocation

```
delinearize 1.0
Usage: delinearize masterRepo linearRepo

  linearRepo  base folder for linearized version repo
  masterRepo  base folder holding master course repository
```

Notes

- Never forget that `delinearize` will only write modifications applied in the `exercises` project. Any other changes will be discarded. Directly apply these type of changes on the master course repo.

Example (continued from the previous example)

Let's undo the changes applied in the previous example, *and* add another text file named `SomeOtherTextFile.txt` under `exercises/src` in exercise `exercise_013_another_faulty_actor`.

```
[ericloots@Eric-Loots-MBP] $ cd /lbt/Studentify/fttas-
linearized/fast-track-akka-scala

[ericloots@Eric-Loots-MBP] $ git rebase -i --root
   pick 9b00254 exercise_000_initial_state
   pick 03bd482 exercise_001_implement_actor
   pick fc818b7 exercise_002_top_level_actor
   pick d95dae6 exercise_003_message_actor
   pick 4684cac exercise_004_use_sender
   pick e890cc4 exercise_005_create_child_actors
   pick cea8563 exercise_006_actor_state
   pick 3372ff7 exercise_007_use_scheduler
   pick 3140ad7 exercise_008_keep_actor_busy
   edit 6accfc2 exercise_009_stop_actor
   pick 72e9cf4 exercise_010_lifecycle_monitoring
   pick 30c10a1 exercise_011_faulty_actor
   pick 0ab3f71 exercise_012_custom_supervision
   edit c7e0f8c exercise_013_another_faulty_actor
   edit 6424eb2 exercise_014_self_healing
   pick 5e90ed0 exercise_015_detect_bottleneck
   pick 4d900a7 exercise_016_use_router
   pick 67206d5 exercise_017_config_dispatcher
   pick 59e16ea exercise_018_become_stash
   pick 8c4bc1a exercise_019_use_ask_pattern
   pick 64f1307 exercise_020_akka_extension
   pick fe111ab exercise_021_fsm

   # Rebase fe111ab onto 99ef9f2

[ericloots@Eric-Loots-MBP] $ rm exercises/src/SampleTextFile.txt

[ericloots@Eric-Loots-MBP] $ git add -A

[ericloots@Eric-Loots-MBP] $ git rebase --continue

[ericloots@Eric-Loots-MBP] $ touch
exercises/src/SomeOtherTextFile.txt

[ericloots@Eric-Loots-MBP] $ git add -A
```

```
[ericloots@Eric-Loots-MBP] $ git rebase --continue

[ericloots@Eric-Loots-MBP] $ rm exercises/src/SomeOtherTextFile.txt

[ericloots@Eric-Loots-MBP] $ git add -A

[ericloots@Eric-Loots-MBP] $ git rebase --continue

[ericloots@Eric-Loots-MBP] $ git st
On branch master
nothing to commit, working directory clean
```

Next, we `delinearize` the project and see what the impact is on the master course repo:

```
delinearize /lbt/FTTAS-v1.3.0/fast-track-akka-scala
/lbt/Studentify/fttas-linearized/fast-track-akka-scala
```

```
[ericloots@Eric-Loots-MBP] $ cd /lbt/FTTAS-v1.3.0/fast-track-akka-
scala

[ericloots@Eric-Loots-MBP] $ git st
On branch master
Your branch is up-to-date with 'origin/master'.
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        exercise_013_another_faulty_actor/src/SomeOtherTextFile.txt

nothing added to commit but untracked files present (use "git add"
to track)
```

validateStudentRepo

`validateStudentRepo` advances through the exercises in a student repository, one at a time. For each exercise it pulls the solution, then runs all the tests to verify they pass.

Invocation

```
usage: validateStudentRepo [directory]
```

createRelease

`createRelease` generates the student repo, validates the tests, packages it

into a zip file, and attaches a version number. This creates a packaged zip that could be distributed to students prior to the course.

Invocation

```
usage: createRelease -v <version> [directory]
```

Notes

If no version is provided, then the default is to use `SNAPSHOT`.

Some projects may require additional arguments to be passed to the studentify command (eg. `-mjvm`). These can be supplied by including a `course_management.conf` file in the root of the project. This file contains the arguments being passed to studentify in the format:

```
STUDENTIFY_ARGS="-mjvm"
```

Appendix 2 - *example* project/CommonSettings.scala

```
import com.typesafe.sbteclipse.core.EclipsePlugin.{EclipseCreateSrc,
EclipseKeys}
import sbt.Keys._
import sbt._
import sbtstudent.AdditionalSettings

object CommonSettings {
  lazy val commonSettings = Seq(
    organization := "com.lightbend.training",
    version := "1.3.0",
    scalaVersion := Version.scalaVer,
    scalacOptions ++= CompileOptions.compileOptions,
    unmanagedSourceDirectories in Compile := List((scalaSource in
Compile).value),
    unmanagedSourceDirectories in Test := List((scalaSource in
Test).value),
    EclipseKeys.createSrc := EclipseCreateSrc.Default +
EclipseCreateSrc.ManagedClasses,
    EclipseKeys.eclipseOutput := Some(".target"),
    EclipseKeys.withSource := true,
    EclipseKeys.skipParents in ThisBuild := true,
    EclipseKeys.skipProject := true,
    parallelExecution in GlobalScope := false,
```

```
    logBuffered in Test := false,
    parallelExecution in ThisBuild := false,
    fork in Test := true,
    libraryDependencies ++= Dependencies.dependencies
  ) ++
    AdditionalSettings.initialCmdsConsole ++
    AdditionalSettings.initialCmdsTestConsole ++
    AdditionalSettings.cmdAliases
}
```

Appendix 3 - *example* project/Dependencies.scala

```
import sbt._

object Version {
  val akkaVer         = "2.4.14"
  val logbackVer      = "1.1.3"
  val scalaVer        = "2.12.1"
  val scalaParsersVer = "1.0.4"
  val scalaTestVer    = "3.0.1"

}

object Dependencies {
  val dependencies = Seq(
    "com.typesafe.akka"       %% "akka-actor"                %
Version.akkaVer,
    "com.typesafe.akka"       %% "akka-slf4j"                %
Version.akkaVer,
    "ch.qos.logback"           % "logback-classic"           %
Version.logbackVer,
    "org.scala-lang.modules"  %% "scala-parser-combinators"  %
Version.scalaParsersVer,
    "com.typesafe.akka"       %% "akka-testkit"              %
Version.akkaVer % "test",
    "org.scalatest"           %% "scalatest"                 %
Version.scalaTestVer % "test"
  )
}
```

Appendix 4 - *example*

project/CompileOptions.scala

```
object CompileOptions {
  val compileOptions = Seq(
    "-unchecked",
    "-deprecation",
    "-language:_",
    "-encoding", "UTF-8"
  )
}
```

Appendix 5 - *sample* project/plugins.sbt

```
addSbtPlugin("com.typesafe.sbteclipse" % "sbteclipse-plugin" %
"5.0.1")
```

Appendix 6 - *sample* project/build.properties

```
sbt.version=0.13.13
```

Appendix 7 - *sample* project/AdditionalSettings.scala

```
package sbtstudent

import sbt._
import Keys._

object AdditionalSettings {

  // Change 'loadInitialCmds' to true when requested in exercise
instructions
  val loadInitialCmds = false
```

```
  val initialCmdsConsole: Seq[Def.Setting[String]] =
    if (loadInitialCmds) {
      Seq(initialCommands in console := "import
com.lightbend.training.coffeehouse._")
    } else {
      Seq()
    }

  val initialCmdsTestConsole: Seq[Def.Setting[String]]  =
    if (loadInitialCmds) {
//      Seq(initialCommands in(Test, console) := (initialCommands in
console).value + ", TestData._")
      Seq()
    } else {
      Seq()
    }

  // Note that if no command aliases need to be added, assign an
empty Seq to cmdAliasesIn
  val cmdAliasesIn: Seq[Def.Setting[(State) => State]] = Seq(
    //    addCommandAlias("xxx", "help"),
    //    addCommandAlias("yxy", "help")
  ).flatten

  val cmdAliases: Seq[Def.Setting[(State) => State]] =
    cmdAliasesIn
}
```

Appendix 8 - *sample* project/Man.scala

```
package sbtstudent
/**
  * Copyright © 2014, 2015, 2016 Lightbend, Inc. All rights
reserved. [http://www.typesafe.com]
  */

import sbt.Keys._
import sbt._
import sbt.complete.DefaultParsers._

import scala.Console
import scala.util.matching._

object Man {
```

```
  val manDetail: String = "Displays the README.md file. Use <noarg>
for setup README.md or <e> for exercise README.md"

  lazy val optArg = OptSpace ~> StringBasic.?

  def man: Command = Command("man")(_ => optArg) { (state, arg) =>
    arg match {
      case Some(a) if a == "e" =>
        val base: File = Project.extract(state).get(sourceDirectory)
        val basePath: String = base + "/test/resources/README.md"
        printOut(basePath)
        Console.print("\n")
        state
      case Some(a) =>
        Console.print("\n")
        Console.println(Console.RED + "[ERROR] " + Console.RESET +
"invalid argument " + Console.RED + a + Console.RESET  + ". " +
manDetail)
        Console.print("\n")
        state
      case None =>
        val base: File = Project.extract(state).get(baseDirectory)
        val readMeFile = new sbt.File(new
sbt.File(Project.extract(state).structure.root), "README.md")
        val basePath = readMeFile.getPath
        printOut(basePath)
        Console.print("\n")
        state
    }
  }

  val bulletRx: Regex = """- """.r
  val boldRx: Regex = """(\*\*)(\w*)(\*\*)""".r
  val codeRx: Regex = """(`)([^`]+)(`)""".r
  val fenceStartRx: Regex = """^```(bash|scala)$""".r
  val fenceEndRx: Regex = """^```$""".r
  val numberRx: Regex = """^(\d{1,3})(\. )""".r
  val urlRx: Regex = """(\()(htt[a-zA-Z0-9\-\.\/:]*)(\))""".r
  val ConBlue = Console.BLUE
  val ConMagenta = Console.MAGENTA
  val ConRed = Console.RED
  val ConReset = Console.RESET
  val ConYellow = Console.YELLOW

  def printOut(path: String) {
```

```
    var inCodeFence = false
    IO.readLines(new sbt.File(path)) foreach {
      case ln if !inCodeFence && ln.length > 0 && ln(0).equals('#')
=>
        Console.println(ConRed + ln + ConReset)
      case ln if !inCodeFence && ln.matches(".*" +
bulletRx.toString() + ".*") =>
        val lne = bulletRx replaceAllIn (ln, ConRed +
bulletRx.toString() + ConReset)
        Console.println(rxFormat(rxFormat(rxFormat(lne, codeRx,
ConBlue), boldRx, ConYellow), urlRx, ConMagenta,
          keepWrapper = true))
      case ln if !inCodeFence && ln.matches(numberRx.toString() +
".*") =>
        val lne = numberRx replaceAllIn (ln, _ match { case
numberRx(n, s) => f"$ConRed$n$s$ConReset" })
        Console.println(rxFormat(rxFormat(lne, codeRx, ConBlue),
boldRx, ConYellow))
      case ln if ln.matches(fenceStartRx.toString()) =>
        inCodeFence = true
        Console.print(ConBlue)
      case ln if ln.matches(fenceEndRx.toString()) =>
        inCodeFence = false
        Console.print(ConReset)
      case ln =>
        Console.println(rxFormat(rxFormat(rxFormat(ln, codeRx,
ConBlue), boldRx, ConYellow), urlRx, ConMagenta,
          keepWrapper = true))
    }
  }

  def rxFormat(ln: String, rx: Regex, startColor: String,
keepWrapper: Boolean = false): String = ln match {
    case `ln` if ln.matches(".*" + rx.toString + ".*") =>
      val lne = rx replaceAllIn (ln, _ match {
        case rx(start, in, stop) =>
          if (keepWrapper)
            f"$start$startColor$in$ConReset$stop"
          else
            f"$startColor$in$ConReset"
      })
      lne
    case _ =>
      ln
  }
```

```
}
```

Appendix 9 - *sample* project/Navigation.scala

```
package sbtstudent

/**
  * Copyright © 2014, 2015, 2016 Lightbend, Inc. All rights
reserved. [http://www.typesafe.com]
  */
import StudentKeys._
import sbt.Keys._
import sbt._

import scala.Console

object Navigation {

  val setupNavAttrs: (State) => State = (state: State) => state

  val loadBookmark: (State) => State = (state: State) => {
    // loadBookmark doesn't really load a bookmark for a master
repo.
    // It just selects the first exercise (project) from the repo
    val refs =
    Project.extract(state)
      .structure
      .allProjectRefs
      .toList
      .map(r => r.project)
      .filter(_.startsWith("exercise_"))
      .sorted
    Command.process(s"project ${refs.head}", state)
  }
}
```

Appendix 10 - *sample* project/StudentCommandsPlugin.scala

```
package sbtstudent

import sbt.Keys._
import sbt._
import Navigation.{loadBookmark, setupNavAttrs}

import scala.Console

object StudentCommandsPlugin extends AutoPlugin {

  override val requires = sbt.plugins.JvmPlugin
  override val trigger: PluginTrigger = allRequirements
  object autoImport {
  }
  override lazy val globalSettings =
    Seq(
      commands in Global ++=
        Seq(
          Man.man
        ),
      onLoad in Global := {
        val state = (onLoad in Global).value
        Navigation.loadBookmark compose(Navigation.setupNavAttrs
compose state)
      }
    )

  override lazy val projectSettings =
    Seq(
      shellPrompt := { state =>
        val base: File = Project.extract(state).get(sourceDirectory)
        val basePath: String = base + "/test/resources/README.md"
        val exercise = Console.BLUE + IO.readLines(new
sbt.File(basePath)).head + Console.RESET
        val manRmnd = Console.RED + "man [e]" + Console.RESET
        val prjNbrNme = IO.readLines(new sbt.File(new
sbt.File(Project.extract(state).structure.root),
".courseName")).head
        s"$manRmnd > $prjNbrNme > $exercise > "
      }
    )
}
```

Appendix 11 *sample*

project/StudentKeys.scala

```
package sbtstudent

import sbt._

object StudentKeys {
  val bookmarkKeyName = "bookmark"
  val mapPrevKeyName = "map-prev"
  val mapNextKeyName = "map-next"
  val bookmark: AttributeKey[File] = AttributeKey[File]
(bookmarkKeyName)
  val mapPrev: AttributeKey[Map[String, String]] =
AttributeKey[Map[String, String]](mapPrevKeyName)
  val mapNext: AttributeKey[Map[String, String]] =
AttributeKey[Map[String, String]](mapNextKeyName)
}
```

Appendix 12 - Demonstration Videos

- Demo of student course repository functionality
- Course Master repository edit workflow and tools

License & Support