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
  - 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
- 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 0.13.12 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
  - 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.
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`

Course master project structure

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

```
                base
                  |
      +-----------+--------------+-----------+
      |           |              |           |
   common   exercise_000_xxx     |     exercise_002_zzz
                                 |
                                 |
                          exercise_001_yyy
```

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
       |
       +- CommonSettings.scala
       |
       +- Dependencies.scala
       |
       +- AdditionalSetting.scala
       |
       +- CompileOptions.scala
       |
       +- plugins.sbt
       |
       +- build.properties
```

In addition to these files, a few other files need to be present in the repository's root folder:

```
./
 |
 +- navigation.sbt
 |
 +- man.sbt
 |
 +- shell-prompt.sbt
```

> 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).

Appendix 1 - *example* CommonSettings.scala

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

object CommonSettings {
  lazy val commonSettings = Seq(
    organization := "com.lightbend.training",
    version := "3.0.2",
    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.Resource,
    EclipseKeys.eclipseOutput := Some(".target"),
    EclipseKeys.withSource := true,
    EclipseKeys.skipParents in ThisBuild := true,
    EclipseKeys.skipProject := true,
    fork in Test := true,
    parallelExecution in Test := false,
    logBuffered in Test := false,
    parallelExecution in ThisBuild := false,
    libraryDependencies ++= Dependencies.dependencies
```

```
  )
}
```

Appendix 2 - *example* Dependencies.scala

```
import sbt._

object Version {
  val scalaVer     = "2.11.8"
  val scalaParsers = "1.0.3"
  val scalaTest    = "2.2.4"
  val playJson     = "2.5.9"
}

object Library {
  val scalaParsers = "org.scala-lang.modules" %% "scala-parser-
combinators" % Version.scalaParsers
  val scalaTest    = "org.scalatest"          %% "scalatest"
% Version.scalaTest
  val playJson     = "com.typesafe.play"      %% "play-json"
% Version.playJson
}

object Dependencies {
  import Library._

  val dependencies = List(
    scalaParsers,
    scalaTest % "test",
    playJson
  )
}
```

Appendix 3 - *example* CompileOptions.scala

```
object CompileOptions {
  val compileOptions = Seq(
    "-unchecked",
    "-deprecation",
```

```
    "-language:_",
    "-target:jvm-1.6",
    "-encoding", "UTF-8"
  )
}
```

Appendix 4 - *sample* plugins.sbt

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

Appendix 5 - *sample* build.properties

```
sbt.version=0.13.13
```

Appendix 6 - 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 either *sbt* or *activator*, the student can move between exercises by issuing the `nextExercise` or `prevExercise` commands.

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

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

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.

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)
```

Appendix 7 - Demonstration Videos

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

License & Support