Project

- Home/README: `https://github.com/harrah/xsbt`
- Documentation
  - Wiki: `https://github.com/harrah/xsbt/wiki`
  - API: `http://harrah.github.com/xsbt/latest/api/index.html`
  - SXR: `http://harrah.github.com/xsbt/latest/sxr/index.html`
- Mailing List: `http://groups.google.com/group/simple-build-tool/topics`
- Issues: `https://github.com/harrah/xsbt/issues`
- Fork: `https://github.com/harrah/xsbt/fork`

Contributing

1. Get a GitHub account: `https://github.com/signup/free`
2. Discuss/justify feature/fix on mailing list.
3. Fork harrah/xsbt to `username/xsbt`
4. Clone `username/xsbt` and branch:

   ```
   $ git clone git@github.com:username/xsbt.git
   $ git checkout -b myfeature
   ```

5. Develop feature/fix
6. Clean up history if necessary (`git rebase` local commits) and push
7. Send pull request, justifying changes if not done in step 2.

8. a) Get feedback. Go to step 5. or b) Pull request gets merged.

Developing / Building from Source

1. Set up latest stable release as `sbt` script: `https://github.com/harrah/xsbt/wiki/Setup`
2. Build:

   ```
   $ sbt
   > publish-all
   > proguard
   ```

3. Copy `sbt` script as `sbt-local` and change launcher path to be `<xsbt-dir>/target/sbt-launch-0.10.2-SNAPSHOT.jar`
4. To use in a project:
   i. Delete that project's `project/build.properties` or set `sbt.version=0.10.2-SNAPSHOT`
   ii. Delete `project/boot/`
   iii. `$ sbt-local`
5. After modifying sbt, repeat step 2. `proguard` may be omitted if the launcher was not modified.
6. To use the updated sbt-local in another project, do one of:
   - `> reboot full` (won't pick up a new launcher)
   - Exit `sbt-local`, delete project/boot/ and start `sbt-local` again

Unit Testing

- Put test sources in `<sub>/src/test/scala/`
- ScalaCheck 1.8 mainly used, some specs 1.6.8 as well
- `sub/test-only sbt.SomeTest` to run a specific test in a sub project

Some areas are more difficult to unit test than others. Project loading and compiler-related code is more difficult; well-defined components like parser combinators, task engine, and settings engine are easier.

Integration Testing

Layout

```
<xsbt>/sbt/src/sbt-test/<group>/<name>/
    test
    ...
```

Running

```
> scripted group/name
> scripted group/*
```

test script

Executing a scripted test involves evaluting the `test` script.

- `> command arg1 ...` runs sbt command
- `$ command arg1 ...` runs script command
- Lines starting with `-` are expected to fail.

Script commands

- File commands accepting one or more files as arguments:
  - `touch`
  - `delete`
  - `exists`
  - `mkdir`
  - `absent`
- `copy-file from/path to/path`
- Control commands
  - `sleep n` where `n` is time in milliseconds
  - `pause` stops execution until enter is pressed
  - `exec command arg1 ...`forks the given command

Example test

Layout:

```
<xsbt>/sbt/src/sbt-test/compile/basic/
    test
    Failure.scala
    changes/
        Success.scala
```

where `test` might be:

```
-> compile
$ delete Failure.scala
$ copy-file changes/Success.scala Success.scala
> compile
```