# OpenDaylight Controller:Eclipse CLI Setup

From Daylight Project

These instructions should enable you to use the CLI to pull and push to the git repository, but hack on code from Eclipse.

## Contents

- 1 Starting From Scratch
- 2 Eclipse Maven Integration (m2e) version
- 3 Debugging Weird Errors
- 4 Alternative way without m2eclipse plugin - for CLI-fans
  - 4.1 Frequent gotchas
  - 4.2 Advantages

## Starting From Scratch

1. Pull the code from the command line and build it with these instructions. The short version is:
   1. Pull the code: `git clone ssh://<username>@git.opendaylight.org:29418/controller.git`
   2. Build the code: `cd controller && mvn clean install` *You can also build from opendaylight/distribution/opendaylight, but it just builds the distribution as opposed to building everything*
2. Download and install Eclipse ("for Java EE Developers", "for Java Developers" and "for RCP and RAP Developers" all seem to work)
3. Install Eclipse Maven Integration
   1. From Eclipse, go to Help => Install New Software...
   2. Paste "http://download.eclipse.org/technology/m2e/releases" into the "Work with:" field
   3. Select m2e and (optionally) m2e - slf4j both *(1.4.0 is current as of the writing of these instructions)*
   4. Click Next, OK and Finish else until things install
4. Restart Eclipse when it asks
5. Install Eclipse Xtend plugin
   1. From Eclipse, go to Help => Install New Software...
   2. Paste "http://download.eclipse.org/modeling/tmf/xtext/updates/composite/releases" into the "Work with:" field

    3. Select Xtend SDK and Xtend M2E extensions under Xtend-2.4.3 *(2.4.3 is the most recent version as of the writing of these instructions)*
    4. Click Next, OK and Finish else until things install
6. Restart Eclipse when it asks
7. Import all of the OpenDaylight projects
    1. From Eclipse, go to File => Import => Maven => Existing Maven Projects
    2. Browse to the root directory of the git repository
    3. All of the projects should be selected by default, just click Finish
    4. It may ask you to install Tycho, do so if it asks
    5. It may also complain about jacoco, you should be able to ignore those errors
8. If the projects still have errors, they are usually because for whatever reason Eclipse didn't properly import the java for the xtend files. To fix this:
    1. Figure out what class seems to be missing.
    2. Find the xtend file for that class and open in it in Eclipse. (I find using the CLI helps me figure out where it is. For example `find . -name "FlowProgrammerAdapter.xtend"`.)
    3. That should cause the xtend file to be compiled and then Eclipse should find it.

## Eclipse Maven Integration (m2e) version

Previously, there were issues with m2e 1.3 and we recommended using 1.2, but those issues have been fixed and everything should works as expected with the most recent m2e version, 1.4.

## Debugging Weird Errors

If Eclipse starts giving you lots of weird errors that seem to make no sense, try the following things:

1. right click on the project with the issue and go to Maven => Update Project... and then press OK
    - You can do this more easily by creating a single "Java Working Set" for all of the OpenDaylight projects
    - Do that by going File => New => Java Working Set and giving it a name and adding all the projects you want
    - When you're importing Maven projects into Eclipse, you can also check the "Add project(s) to working set" box at the bottom of the dialog and pick the working set you're using for OpenDaylight
    - Then you can right-click on the project and do the Maven = Update Project... on all of them at once
2. If that doesn't work closing, deleting and the reimporting all of the OpenDaylight projects seems to help when nothing else does
    - while you have the projects not imported into Eclipse, it doesn't hurt to do a build from the CLI as well

## Alternative way without m2eclipse plugin - for CLI-fans

Sometimes m2eclipse plugin does weird things like compiling in infinite loop, updating dependencies when they need to be frozen and the other way around. If you want to omit m2eclipse plugin completely, you can. Of course the xtend plugin is still needed.

1. pull code from git
2. run in folder with pom:

   ```
   mvn eclipse:eclipse -DdownloadSources=true
   ```

3. start eclipse and import *existing project* (navigate to folder with pom)

## Frequent gotchas

- eclipse needs to setup a classpath variable M2_REPO pointing to local repository. Can be done manually or via

  ```
  mvn -Declipse.workspace=<path-to-eclipse-workspace> eclipse:add-maven-repo
  ```

- some projects miss maven-compiler-plugin configuration and render into java 1.4 compliant settings - you need to change that manually in eclipse or fix corresponding pom
- when running eclipse:eclipse upon parent pom which aggregates modules, then in eclipse these modules will "see" each other in workspace. If you change code on project A, eclipse will recompile it and new stuff is immediately available to project B (having the same parent). Anyway workspace dependencies in eclipse can be set or removed manually anytime.
- when running eclipse:eclipse upon bottom project pom, where no modules resides, eclipse will see it after import as independent project having all dependencies pointed to local repo
- when using pax-exam you need to install touched artifacts into local repo in order for pax-exam to put updated version on classpath
- by default maven updates every SNAPSHOT dependency every 24 hours, so if you need stable environment do not forget to use **-nsu** switch

## Advantages

- if you want to freeze all SNAPSHOT dependencies, use **-nsu** (no snapshot update) switch or **-o** (offline)
- if you want to update all SNAPSHOT dependencies, use **-U** switch
- if you want to build only a subset of projects contained in parent pom, use this (maven will take care of build order)

  ```
  mvn clean install -pl <relative_path_to_subproject1>,<relative_path_to_subproject2>..
  ```

- if you want to continue a failed build (after fixing it) when having many modules, use this

  ```
  mvn clean install -rf <relative_path_to_broken_broject>
  ```

- you can apply remote debug on everything you have source code of (including plugins), just use **mvnDebug**
- to run test from commandline use

```
mvn test
mvn test -Dtest=<testName_without_package>
mvn test -Dtest=<testName_without_package>#<methodName>
```

- if everything fails, use -X to see TRACE output

And as all these are shell commands, you are free to chain them (with ; || && &) and automate your work a bit:

```
mvn clean compile -DskipTests && cd target/longPathToDistribution; aplay <funnyNoise.wav> > /dev/null & ./run.bat -debug; cd -
```