

Building Solr with Gradle

Starting with Solr 9, we have changed our build process to be Gradle, Ant is no longer supported. Versions of Solr through Solr 8x are built only with Ant. This page is intended to help people use Gradle to build Solr, with special attention to tips for people already familiar with the Ant build process.

⚠ Solr versions through 8x must be built with Ant.

I want to especially thank Dawid Weiss for all his work on this, without his help and guidance (ok, his willingness to do most of the heavy lifting), we wouldn't be able to even consider transitioning to Gradle yet or even in the foreseeable future.

Gdub

There's a helpful project at: <https://github.com/dougborg/gdub> that lets you run targets from submodules rather than having to do everything top level with [gradlew](#). Where you might normally do `./gradlew solr:core:test` from the root directory, gdub lets you do `cd solr/core; gw test`. This is not necessary, just a convenience.

Setup

If you get through these steps, you're ready to start using Gradle regularly:



1. Get a current version of Lucene/Solr with the usual "git clone <https://github.com/apache/lucene-solr.git> dest"
2. **cd dest**
3. **./gradlew help**. Gradlew is "gradle wrapper" and should automatically download gradle and start it running.
4. **./gradlew helpAnt** - For people already familiar with building Solr with Ant, this command will show Gradle tasks that are equivalent to selected Ant targets.
5. The Gradle project is here if you have trouble, or ping the Lucene dev list: <https://docs.gradle.org/current/userguide/installation.html>
6. **./help** contains plain-text files you can scan, for instance 'ant.txt' is the output from `./gradlew helpAnt`.


Gradle on trunk/9.0 only

As of late August 2020, Ant support has been removed from trunk/9.0.

Useful commands:

Here are some commands that will get you started (all prefixed by **./gradlew** or just **gw** if you've installed gdub).

- **help** - of course.
- **helpAnt** - lists Gradle commands that are the equivalent of what you may be used to in Ant.
- **assemble** - will create a runnable Solr instance in `./solr/packaging/build/solr-#.#.#-SNAPSHOT`. However, it will delete that directory next time it's run. 
- **dev** - like assemble, but will write the output code to `./solr/packaging/build/dev` and will *not* delete the directory first. Use this if you're developing code, instantiating Solr and recompiling and need to preserve your setup. 
- **tasks** - lists all the tasks you can execute. There are a lot of them...
- **check** - will run all of the checks (ant precommit+) and run all of the tests. **Please get in the habit of running this!** More below for why in "Differences from Ant".
 - Adding **-x test** will run all the validation checks but not run tests (this is the rough equivalent of `ant precommit`)

 The first time you run these commands, all of the dependencies for Solr will be downloaded to `(~nix) ~/.gradle/caches`, and this process may take quite some time depending on your network connectivity.

Differences from Ant:

Some operations that are simply different from Ant. Here are some hints to help with these:

- Gradle runs fastest when it uses a daemon. The first time you use gradle, it'll take some time (usually 10-20 seconds) to start the daemon and build caches. You'll see a message about "starting daemon". Subsequent runs should mostly skip this step
 - You may want to change `org.gradle.daemon.idletimeout=900000` in your gradle.properties file (written the first time you run gradle) to a longer interval (this is in ms) to avoid restarting the daemon as frequently.
 - **gradlew --stop** will stop the daemon if you think it's confused.
- The build **will fail on un-suppressed warnings**. Trunk compiles cleanly, all warnings are suppressed or have been fixed (mostly suppressed). The first option if your new code generates more warnings is to change the code so it doesn't. Failing that add a SuppressWarnings annotation, but please try to fix the code first.
 - If you're working on code and can remove some of these suppressions, please do. It was too big a task to try to fix all the warnings by fixing the code all at once. We should improve from here.
- The validation checks fail on log messages that follow potentially wasteful patterns. **./gradlew helpValidateLogCalls** describes how to avoid these errors. This is part of **./gradlew check**.

- The validation is a bit harsh, if you have a logging call that is flagged but you've inspected and know is correct/efficient, add `//logok` as a comment. But please read through `./gradlew help/validateLogCalls` for guidelines, it's more complex than it appears at first glance.
- It's not necessary to run the **idea** task before opening the project, just "open or import" and choose the root directory of your source tree.
- The output directories have changed for artifacts and test results. Pay close attention to the messages on the screen at the end of a task execution, they'll tell you exactly where to look.

IntelliJ IDE



Opening the project

There is a **gradlew idea** task, but it usually is sufficient to just "open or import" the root directory. IntelliJ may prompt you to "import Gradle project?" and you can answer "yes". This will take a minute or two the first time.

IntelliJ integration is useful. There's nothing special about our support for Idea, it's just the one that's been exercised most. In particular the gradle window lets you easily execute tasks. In the case of failures because of the `-Werror` flag and just click to the source. The **classes** and **testClasses** tasks are particularly useful here as they just compile the Java code without the extra work the **assemble** task does.

Running tests in IntelliJ seems to take longer, we're investigating why.

Please do not let IntelliJ "fix" its inspections by adding the IntelliJ-specific suppression. That said, it's useful to look at the inspections, they may be highlighted. Some of them are not useful, any tips on how to turn off specific ones appreciated. For instance, I personally like **if (something == false)** because I think **if (!something)** is easy to misread, but IntelliJ prefers the latter. There's no intention here to make all our code conform to IntelliJ's inspections. That said, looking at them can be instructive.

Eclipse IDE:

Eclipse is also a popular IDE, and support has been added. However, this IDE has not been exercised nearly as much as IntelliJ. If you prefer Eclipse, please report any shortcomings on the dev mailing list.



Opening the project

After running the **gradlew eclipse** task, don't import it as gradle project, just as a plain java project.

Other IDEs:

Please feel free to add any tips about other IDEs. As mentioned above, IntelliJ was just what has been used most.

Troubleshooting:

We've run into some rough edges that are outlined here:

- When switching back and forth between Gradle on trunk and 8x where you have to use ant, there are some Gradle-specific files that will be on your 8x tree. These should be ignored by the build system and Ant. You can delete them if you want. If you do, be aware that the next time you run Gradle it'll write a new "gradle.properties" file if it's not there so any older edits will be lost.
 - Gradle writes a new "gradle.properties" file at the root of your source tree if it's not there already, sometimes this can surprise you
- I've occasionally seen gradle stop with a message about thrashing memory when running the **check** task. There are two short-term solutions here:
 - **./gradlew --stop** will halt the daemon, releasing memory
 - Increase the memory allocated to Gradle by editing the settings in your gradle.properties file.
 - On the TODO list is to figure out why 3G (the current default which I sometimes bump even higher) and address whatever is causing that.

Related articles

- [Move your PR to the new Solr or Lucene GitHub repo](#)
- [Building Solr with Gradle](#)
- [Tips-n-tricks for avoiding compiler warnings](#)
- [\[Experimental\] Getting Started with Solr Development](#)