Usage

The plugin offers goals for updating the versions of artifacts referenced in a Maven `pom.xml` file.

Basic Usage

Maven 2.0, 2.1, 2.2 and 3.0 do not currently support re-reading modifications of the `pom.xml` within one invocation of Maven.

The following goals:

- versions:set
- versions:lock-snapshots
- versions:resolve-ranges
- versions:unlock-snapshots
- versions:update-child-modules
- versions:update-parent
- versions:update-properties
- versions:use-latest-releases
- versions:use-latest-snapshots
- versions:use-latest-versions
- versions:use-next-releases
- versions:use-next-snapshots
- versions:use-next-versions
- versions:use-releases

modify the `pom.xml` file, you need to run these goals separately from any other goals or life-cycle phases.

Note: The first time any of the goals that modify the `pom.xml` file run, they will create a local backup copy `pom.xml.versionsBackup`. Subsequent modifications will leave this backup unchanged. The versions:commit goal will remove the backup copy, while the versions:revert goal will restore the backup copy. It is best practice to use a Source Code Management system and not rely on the `pom.xml.versionsBackup` files created by the versions-maven-plugin. The versions:commit and versions:revert goals are only a "Poor Man's SCM".

Goals that modify the `pom.xml`

Executing any of the following goals may modify your `pom.xml` file.

Reverting modifications to the `pom.xml` files (Note: modifies `pom.xml` files)

To restore your `pom.xml` files to their initial state, before you started modifying it with the versions-maven-plugin, invoke the `revert` goal. Note that it is best practice to use a Source Code Management system and not rely on the `pom.xml.versionsBackup` files created by the versions-maven-plugin.

```
mvn versions:revert
```

Accepting modifications to the `pom.xml` files

To accept the modifications made to your `pom.xml` files by the versions-maven-plugin invoke the `commit` goal. This will have the effect of removing any `pom.xml.versionsBackup` files. Note that it is best practice to use a Source Code Management system and not rely on the `pom.xml.versionsBackup` files created by the versions-maven-plugin.

```
mvn versions:commit
```

Updating the parent version (Note: modifies `pom.xml` files)

To update the parent version of your POM to the latest available, just invoke the `update-parent` goal.

```
mvn versions:update-parent
```

A more detailed example of the `update-parent` goal.

Fixing a multi-module build (Note: modifies `pom.xml` files)

If you have a multi-module build where the aggregator pom (i.e. the one with packaging of `pom` and the `modules` section) is also the parent referenced by its child modules, and the aggregator version does not match the version specified in the parent section of the child modules, Maven will not let you build the project. To fix all the child modules, use the versions:update-child-modules goal and invoke Maven in non-recursive mode.

```
mvn -N versions:update-child-modules
```

A more detailed example of the `update-child-modules` goal.

Updating versions specified by properties (Note: modifies `pom.xml` files)

This goal helps when you use properties to define versions. Please see the `update-properties` example.

Updating versions of dependencies (Note: modifies `pom.xml` files)

There are a set of goals to help with advancing dependency versions: `use-latest-releases`, `use-latest-snapshots`, `use-latest-versions`, `use-next-releases`, `use-next-snapshots`, and `use-next-versions`.

Please see the advancing dependency versions example.

Resolving version ranges (Note: modifies `pom.xml` files)

If a pom contains version ranges in one or more dependencies, it can be useful to collapse those ranges to the specific versions used during the build, just invoke the `resolve-ranges` goal.

```
mvn versions:resolve-ranges
```

More examples of the `resolve-ranges` goal.

Locking and unlocking -SNAPSHOT versions (Note: modifies `pom.xml` files)

If your pom contains a lot of -SNAPSHOT dependencies and those -SNAPSHOT dependencies are a moving target, it can sometimes be helpful to temporarily replace the -SNAPSHOT with a locked -YYYYMMDD.HHMMSS-NNN snapshot. In the long term, you will need to return to the -SNAPSHOT dependencies and then replace them with their release version, but if you need a short term semi-reproducible build, locked -SNAPSHOTs can sometimes be a useful hack.

To replace -SNAPSHOT dependencies with their current locked snapshot equivalents, just invoke the `lock-snapshots` goal.

```
mvn versions:lock-snapshots
```

More examples of the `lock-snapshots` goal.

To return to regular -SNAPSHOT dependencies, i.e. replace 1.9.6-20090103.152537-3 with 1.9.6-SNAPSHOT, just invoke the `unlock-snapshots` goal.

```
mvn versions:unlock-snapshots
```

More examples of the `unlock-snapshots` goal.

Picking up releases of -SNAPSHOT dependencies (Note: modifies `pom.xml` files)

It is better to depend on a released version of a dependency, rather than the -SNAPSHOT of that version. For example, it is better to depend on version 1.3.4 rather than 1.3.4-SNAPSHOT. Of course if you are waiting for 1.3.4 to be released, you will have had to add 1.3.4-SNAPSHOT as a dependency.

The `use-releases` goal will look at your project dependencies and see if any -SNAPSHOT versions have been released, replacing the -SNAPSHOT version with the corresponding release version.

To replace -SNAPSHOT dependencies with their corresponding release version, just invoke the `use-releases` goal.

```
mvn versions:use-releases
```

More examples of the `lock-snapshots` goal.

Setting the project version

To set the project version to a specific version, just invoke the `set` goal.

```
mvn versions:set -DnewVersion=1.0.1-SNAPSHOT
```

More examples of the `set` goal.

Goals that do not modify the `pom.xml`

Executing any of the following goals will not modify your `pom.xml` file.

Checking for new versions of plugins

To get information about newer versions of plugins that you are using in your build, just invoke the `display-plugin-updates` goal.

```
mvn versions:display-plugin-updates
```

A more detailed example of the `display-plugin-updates` goal.

Checking for new versions of dependencies

To get information about newer versions of dependencies that you are using in your build, just invoke the `display-dependency-updates` goal.

```
mvn versions:display-dependency-updates
```

A more detailed example of the `display-dependency-updates` goal.

Checking for new versions of specified by properties

To get information about newer versions of dependencies that you are using in your build, just invoke the

`display-property-updates` goal.

```
mvn versions:display-property-updates
```

A more detailed example of the `display-property-updates` goal.

Report Usage

The plugin also offers some reporting views; these make no changes to your project, but makes it easy for reviewers to survey available updates without even having to launch Maven themselves. Updates are further categorized as major, minor, and incremental, to make it easier to decide which updates to take.

To add these reports to your project's site, add the following snippet to your POM:

```
<reporting>
  <plugins>
    <plugin>
      <groupId>org.codehaus.mojo</groupId>
      <artifactId>versions-maven-plugin</artifactId>
      <version>2.2</version>
      <reportSets>
        <reportSet>
          <reports>
            <report>dependency-updates-report</report>
            <report>plugin-updates-report</report>
            <report>property-updates-report</report>
          </reports>
        </reportSet>
      </reportSets>
    </plugin>
  </plugins>
</reporting>
```