Intellij IDEA

Your Rating: ☆☆☆☆☆ Results: ★★★☆☆ 152 rates

Rule #1

There's only one shortcut to know by heart: *Find Action...* A. Be sure to check if this is assigned to another keyboard shortcut (under the *Help* menu).
You will learn the other shortcuts over time from this dialog.

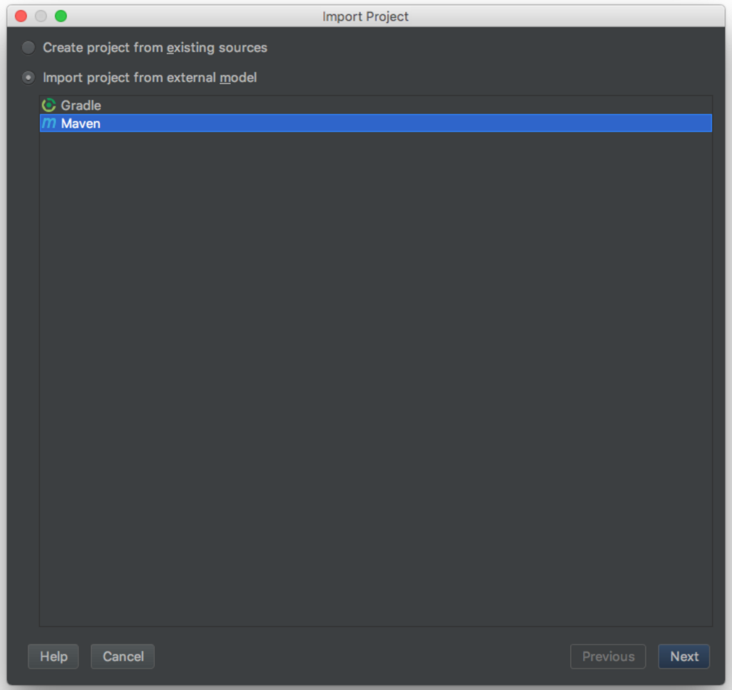
Rule #2

Don't disable the tips on startup. Read them.
One at a time, it will teach you the shortcuts, productivity boosts, as well as new features upon upgrades.

Creating a project

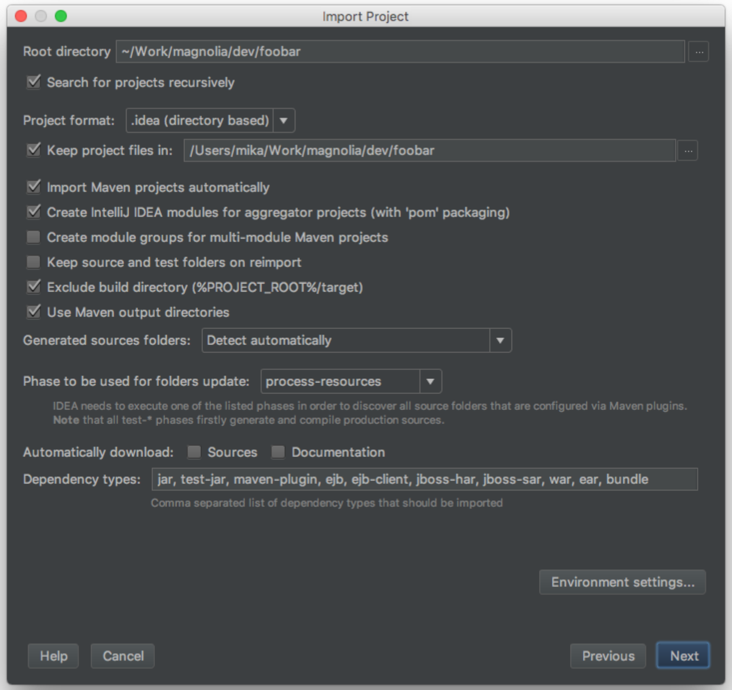
In IntelliJ, hit *File* > *New* > *Project from Existing Sources...*
Pick a directory containing project or module sources from Git.

Choose *Import project from external model*, then select *Maven*.



Import Project

- Create project from existing sources
- [✓ Import project from external model]
  - Gradle
  - [✓ Maven]



Import Project

Root directory: ~/Work/magnolia/dev/foobar

✓ Search for projects recursively

Project format: .idea (directory based)

✓ Keep project files in: /Users/mika/Work/magnolia/dev/foobar

✓ Import Maven projects automatically
✓ Create IntelliJ IDEA modules for aggregator projects (with 'pom' packaging)
☐ Create module groups for multi-module Maven projects
☐ Keep source and test folders on reimport
✓ Exclude build directory (%PROJECT_ROOT%/target)
✓ Use Maven output directories

Generated sources folders: Detect automatically

Phase to be used for folders update: process-resources

IDEA needs to execute one of the listed phases in order to discover all source folders that are configured via Maven plugins.
Note that all test-* phases firstly generate and compile production sources.

Automatically download: ☐ Sources ☐ Documentation

Dependency types: jar, test-jar, maven-plugin, ejb, ejb-client, jboss-har, jboss-sar, war, ear, bundle

Comma separated list of dependency types that should be imported

Couple interesting options in the second *Import Project* dialog:

- *Keep project files in:* Use this if you work with a project referencing modules outside of the project directory.
  This will keep the modules' `.iml` files in the project root directory, instead of polluting your module's Git clone.
  This helps against corrupt projects, when the same module sources are used in two or more projects (basically both projects content over the module's `.iml` file).

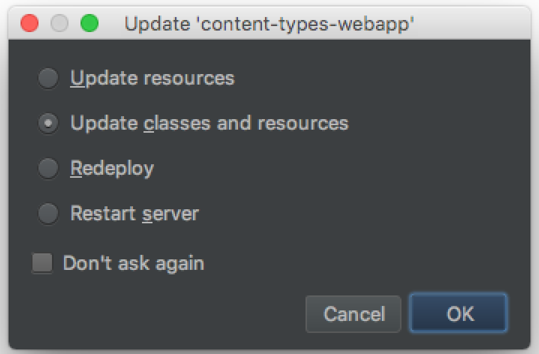
- *Import Maven projects automatically*: keep this on by default, but you may disable it when doing POM changes and working on a sloppy connection.

In the same dialog, check the *Environment settings...* too. If you have a local Maven installation, you may prefer to use this one, instead of the one bundled with IntelliJ.

In the next two dialogs, you are prompted to confirm the active profiles (no action required), and modules to import.
Be careful with module selection, in case your source directory already contains some webapp *overlays*.

Updating a running application

To get hot-swap on classes and resources (including FreeMarker files), you need to *Build Project* `F9`, which in turn updates the application.
You may also *Update* `F10` the application as a standalone step.



Update 'content-types-webapp'

- Update resources
- [✓ Update classes and resources]
- Redeploy
- Restart server

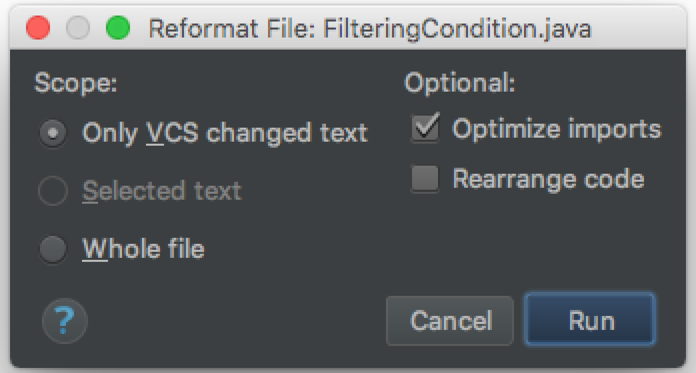
Don't ask again

You may also configure the *On 'Update' action* in each Run/Debug configuration; *Update classes and resources* is a good default.
Keep an eye on the project's *Event Log*, or *balloon notifications*: if the classes cannot be hot-swapped (*e.g.* binary signature changes), a red info message will be displayed.

As opposed to Eclipse, as soon as a webapp module is present in your project, the resources are "packaged" (as opposed to "compiled"), and therefore not copied anymore to the module's own output folder, but rather to the webapp's.
See http://devnet.jetbrains.net/message/5317525#5317525 for updates. Turns out we don't even need to do anything to FreeMarker caching nor template loaders, but as far as experiments go, templates may be refreshed via *Build Project* or *Update*.

Reformatting options

We only reformat the new and modified lines. To turn on this behavior, open the *Reformat Code* dialog with `L`.
Select *Only VCS changed text*, and tick *Optimize imports*.


Reformat File: FilteringCondition.java

Scope:
- (●) Only VCS changed text
- ( ) Selected text
- ( ) Whole file

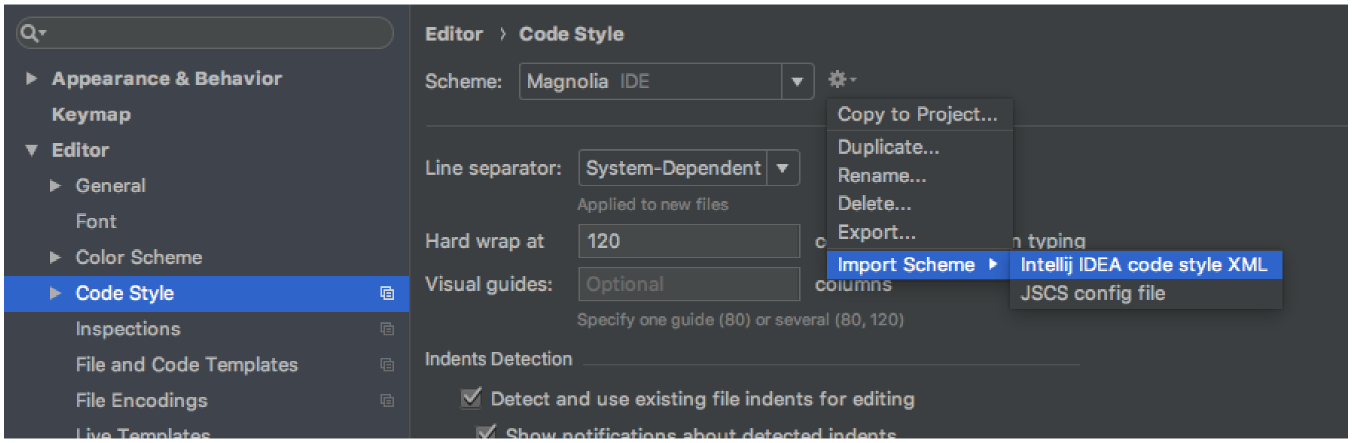
Optional:
- [✓] Optimize imports
- [ ] Rearrange code

After this is done, the regular *Reformat Code* shortcut is L.

Importing Magnolia's Code Style

Upon first setup, import the code style settings attached to this page, as shown in the screenshot below.
This is updated now and then to best reflect our coding conventions. See also the Code Style page.

Go to *IntelliJ IDEA* > *Preferences...* > *Code Style* and *Import Scheme* from *Intellij IDEA code style XML*.


Editor › Code Style

Scheme: Magnolia IDE

Line separator: System-Dependent
Applied to new files

Hard wrap at: 120

Visual guides: Optional
Specify one guide (80) or several (80, 120)

Indents Detection
- [✓] Detect and use existing file indents for editing
- [✓] Show notifications about detected indents

Magnolia-IntelliJ-CodeStyle.xml

License copyright automation

See Copyright years for recent research and guidelines on how to work with copyright years in the license at the beginning of files. There's also IntelliJ setup for adding those licenses automatically at the beginning of files.