Leiningen Documentation

Leiningen User's Guide

User's Guide to Leiningen

Notice

Notice

Topics:

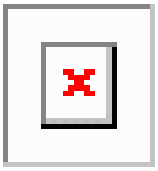
- *Trademarks*

This information was developed for products and services offered in the U.S.A.

This product is meant for educational purposes only. Some of the trademarks mentioned in this product appear for identification purposes only. Not responsible for direct, indirect, incidental or consequential damages resulting from any defect, error or failure to perform. Any resemblance to real persons, living or dead is purely coincidental. Void where prohibited. Some assembly required. Batteries not included. Use only as directed. Do not use while operating a motor vehicle or heavy equipment. Do not fold, spindle or mutilate. Do not stamp. No user-serviceable parts inside. Subject to change without notice. Drop in any mailbox. No postage necessary if mailed in the United States. Postage will be paid by addressee. Post office will not deliver without postage. Some equipment shown is optional. Objects in mirror may be closer than they appear. Not recommended for children. Your mileage may vary.

No other warranty expressed or implied. This supersedes all previous notices.

COPYRIGHT LICENSE:



Trademarks

The following terms are trademarks of the Foobar Corp. in the United States, other countries, or both:

Foobarista ®

The following terms are trademarks of other companies:

Red, Orange, Yellow, Green, Blue, Indigo, and Violot are registered trademarks of Rainbow Corporation and/or its affiliates.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Contents

Abstract

User Guide for Leiningen. See also the Technical Manual.

Preface

Preface

Preface...

Chapter
1

Overview: Configuration, Customization, Coordination

Topics:

"Leiningen is for automating Clojure projects without setting your hair on fire."

Leiningent is a general “CCC” (configuration, customization, and coordination) tool.

Configuration	Use it to "configure" tools, i.e. set options, for various tasks.
Customization	Groups and individuals can “inherit” general configurations and then customize them on a per-task and/or per-project basis.
Coordination	Use it to integrate and coordinate related activities. For example, quality assurance, code check-in, and issue tracking are related activities. Leiningen can help you automate standard procedures such as: first QA check code, then check it in with a reference to the bug it fixes, and then close the bug in the Issues Tracker, annotating it with the commit ID.

The Problem

Leiningent is a general “CCC” (configuration, customization, and coordination) tool.

This section describes the sort of problems Leiningen is designed to solve: configuration, coordination, etc.

etc

The Solution

This section describes in general how Leiningen solves the problems described in the previous topic. I.e. it's more about the strategies Leiningen adopts than the specific implementation techniques; they are the subject of the remainder of this book..

etc

Chapter
2

Getting Started

Topics:

We get started by first exploring a minimal project, and then exploring the Leiningen infrastructure. This should give us a good general idea of what pieces are involved and how the work together.

Before delving into the details we take a brief tour of:

- A minimal project
- The Leiningen infrastructure

Installation

Unix-like systems

If your preferred *package manager* has a relatively recent version of Leiningen, try that first. Otherwise you can install by hand:

Leiningen bootstraps itself using the `lein` shell script; there is no separate install script. It handles installing its own dependencies, which means the first run will take longer.

1. Make sure you have a Java JDK version 6 or later.
2. *Download the script*
3. Place it on your *$PATH*. (`~/bin` is a good choice if it is on your path.)
4. Set it to be executable. (`chmod 755 ~/bin/lein`)

Windows

There is an *installer* which will handle downloading and placing Leiningen and its dependencies.

The manual method of putting the *batch file*. on your *PATH* and running `lein self-install` should still work for most users. If you have Cygwin you should be able to use the shell script above rather than the batch file.

Project Structure and Dynamics

We create a minimal project and explore its structure and configuration.

A minimal Leiningen project involves: a directory structure, a project configuration file (project.clj), and source code files. In addition to these static resources, Leiningen dynamically determines a project map, a dependency tree, and several other structures. This section takes you through some simple steps to exlore these elements.

1. Create a new project by executing: `lein new app my-app`

   where `lein` is the Leiningen command, and the args are:

   new
   : a Leiningen *task*. To see the syntax and semantics of this task, run `lein help new`. To see a list of the tasks that come preinstalled, run `lein help`.

   app
   : is the name of a project template, in this case the default application template. To see a list of preinstalled templates, run `lein help new`. Leiningen's concept of *task* is discussed in detail under the X topic of this guide.

   my-app
   : is the name to be used for the new project

   This will create a hierarchy of directories and populate it with some files generated from templates:

```
.
./.gitignore
./doc
./doc/intro.md
./project.clj
./README.md
./src
./src/my_stuff
```

```
./src/my_stuff/core.clj
./test
./test/my_stuff
./test/my_stuff/core_test.clj
```

2. Take a look at the Leiningen project file buy running `less my-app/project.clj`
   You should see something like:

```
(defproject my-app "0.1.0-SNAPSHOT"
  :description "FIXME: write description"
  :url "http://example.com/FIXME"
  :license {:name "Eclipse Public License"
            :url "http://www.eclipse.org/legal/epl-v10.html"}
  :dependencies [[org.clojure/clojure "1.5.1"]])
```

   👉 Note: This has the form of a function application, where `defproject` is the function, and the args come as a list of pairs. In fact, `defproject` is a macro that expands into the definition of a Clojure map named `project`. Each task - both those preinstalled and those defined by plugins - takes this `project` map as input. The key point is that the key-value pairs of the `project` map are thus available for use by task implementations to control setting of options etc. for the processes they manage.

   👉 Note: The final `project` map is determined by a combination of several sources in addition to project.clj, such as *~/.lein/profiles.clj*. See X for details.

   In this minimal example, the only really functional parameter is `:dependencies`. This is used to specify which libraries/jars the project uses (and thus depends on). One of the best things about Leiningen is its powerful management of dependencies. Using the [?] library under the hood, Leiningen is able to construct the entire dependency graph for the project and arrange for everything needed to be installed. Leiningen's dependency management capabilities are described in X.

   👉 Important: Leiningen dependency specification uses the naming conventions established by Maven: artifact-group/artifact-id. Etc. See X for details.

3. Examine the project map: `lein pprint`
   You should see something like:

```
{:compile-path "/Users/gar/tmp/my-app/target/classes",
 :group "my-app",
 :license
 {:name "Eclipse Public License",
  :url "http://www.eclipse.org/legal/epl-v10.html"},
 :global-vars {},
 :checkout-deps-shares
 [:source-paths
  :test-paths
  :resource-paths
  :compile-path
  #&Var@3e25e2b8:
    #amp;classpath$checkout_deps_paths
leiningen.core.classpath$checkout_deps_paths@89bbe8c>>],
 :dependencies
 ([org.clojure/clojure "1.5.1"]
  [org.clojure/tools.nrepl "0.2.3" :exclusions ([org.clojure/clojure])]
  [clojure-complete/clojure-complete
   "0.2.3"
   :exclusions
   ([org.clojure/clojure])]),
```

```
:plugin-repositories
[["central" {:snapshots false, :url "http://repo1.maven.org/maven2/"}]
 ["clojars" {:url "https://clojars.org/repo/"}]],
:test-selectors {:default (constantly true)},
:target-path "/Users/gar/tmp/my-app/target",
:name "my-app",
:deploy-repositories
[["clojars"
  {:username :gpg,
   :url "https://clojars.org/repo/",
   :password :gpg}]],
:root "/Users/gar/tmp/my-app",
:offline? false,
:source-paths ("/Users/gar/tmp/my-app/src"),
:certificates ["clojars.pem"],
:version "0.1.0-SNAPSHOT",
:jar-exclusions [#"^\."],
:profiles {:uberjar {:aot :all}},
:prep-tasks ["javac" "compile"],
:url "http://example.com/FIXME",
:repositories
[["central" {:snapshots false, :url "http://repo1.maven.org/maven2/"}]
 ["clojars" {:url "https://clojars.org/repo/"}]],
:resource-paths
("/Users/gar/tmp/my-app/dev-resources"
 "/Users/gar/tmp/my-app/resources"),
:uberjar-exclusions [#"(?i)^META-INF/[^/]*\.(SF|RSA|DSA)$"],
:main my-app.core,
:jvm-opts ["-XX:+TieredCompilation" "-XX:TieredStopAtLevel=1"],
:eval-in :subprocess,
:plugins
([lein-localrepo/lein-localrepo "0.4.1"]
 [lein-difftest/lein-difftest "1.3.8"]
 [org.clojure/java.classpath "0.2.0"]
 [lein-marginalia/lein-marginalia "0.7.1"]
 [lein-mustache/lein-mustache "0.2"]
 [lein-pprint/lein-pprint "1.1.1"]),
:native-path "/Users/gar/tmp/my-app/target/native",
:description "FIXME: write description",
:test-paths ("/Users/gar/tmp/my-app/test"),
:clean-targets [:target-path],
:aliases nil}
```

Take some time to look this over. Most of these parameters you will never have to deal with, but it's good to have an idea of what sorts of things Leiningen is interested in.

Note: The predefined paramters are documented in the Technical Reference Manual. Other chapters of this User's Guide explain how to use them.

Note: We should distinguish between the project map that results from Leiningen's work and the defproject map in project.clj that forms the starting point for project map construction.

4. Tell Leiningen to install all dependencies by running `lein deps`
   Since you already have Clojure installed, you probably won't see any output.
5. Check your dependency tree: `lein deps :tree`
   You should see something like the following:

```
[clojure-complete "0.2.3" :exclusions [[org.clojure/clojure]]]
```

```
 [org.clojure/clojure "1.5.1"]
 [org.clojure/tools.nrepl "0.2.3" :exclusions [[org.clojure/clojure]]]
```

This is a complete listing of the jars your project depends on, derived from your project.clj :dependencies parameter plus a set of default maps to be described later. Since this is a tree representation, you can infer that these three libraries are independently specified; none of them has required any of the others as a dependency. In fact, `clojure-complete` and `org.clojure/tools.nrepl` are both specified as dependencies by the default `:base` *profile*, which means that they will always be in the dependency tree for every project (unless overriden). Profiles are named maps that can be used to customize the `project` map in various ways; they are fully described in X.

6. Now let's take a look at the application code installed by the `app` template. Use your editor, or run `cat src/my_app/core.clj`

   The contents of `my-app/src/my_app/core.clj` should look something like:

```
(ns my-app.core
  (:gen-class))

(defn -main
  "I don't do a whole lot ... yet."
  [& args]
  (println "Hello, World!"))
```

Notice: The `:gen-class` option and the definition of `-main`. Remember this was produced by the application template, rather than the library template. So it assumes you want to execute the result, which means you need to generate Java byte code. That's what :gen-class does. You also need a main entry point; thats what `-main` does.

Warning: In case it isn't obvious: to effectively use Leiningen, you need to know Clojure. That is, the better your mastery of Clojure, the more you can do with Leiningen. Leiningen is a Clojure application, after all.

The Leiningen Infrastructure

Leiningen's infrastructure config files, the local repo, remote repos, etc. We should also include default profiles such as :base, since they are just as fundamental. This topic provides a brief overview of these parts and how they fit together.

External

Directories and files used by Leiningen

Exploring: ~/.lein; ~/.m2/repository;

Do this

Internal infrastructure: profiles, etc.

Profiles and other internally defined (but overridable) stuff on which Leiningen's functionality depends.

Part

I

Leiningen in Detail

Topics:

Chapter
1

Parameters and Options

Terminology: in keeping with standard Unix usage, we reserve the term *option* for bits of state that control the behavior of software. For example, many command line tools have a "verbose" option that controls how much information the tool dumps the stdout/stderr as it executes.

In order to specify (or set) an option, the user must supply the appropriate *parameter* as part of the command. Alternatively, software commonly accepts parameters stored in environment variables. Yet a third alternative is to accept parameters specified in a configuration file.

In practice the distinction between options and parameters in this sense is overlooked or not even noticed in the first place, with no ill effect. But for the sake of clarity, it is useful to respect the distinction when it comes to Leiningen. There are two reasons for this.

First, Leiningen project files may contain parameters for any number of programs, each of which support different options. So although it makes sense to think of the project map as a kind of configuration file, it is not like the configuration files specific to particular software packages. For example, Jetty depends heavily on configuration files that are specific to Jetty; there is no reason to put anything in a Jetty configuration file that Jetty would not understand. A Leiningen project map, by contrast, ordinarily will contain parameters for several different software components.

Second, Leiningen places no constraints on the representation of any option. Since Leiningen is extensible via plugins, the most Leiningen can do is provide a means of associating a value with a key - a Clojure map. It then becomes the responsibility of the plugin to correctly interpret the key-value pair (which we call a *parameter*). This means in particular that a plugin designer can choose key and value names for configuration parameters that are completely different than their corresponding software options.

A simple example: the `java` command understands a "max heap size" option, whose command line parameter is `-Xmx<size>`. Leiningen does not directly support this as a configuration parameter, however; instead it has a `:jvm-opts` parameter, whose value can be anything `java` understands to be an option-setting parameter. So Leiningen must *construct* the appropriate command line from the project map parameters.

Furthermore, it is possible to choose completely different names; for example, Leiningen could have used “semantic” parameters; in this example, it could have supported a `:jvm-max-heap` parameter that takes a number in MB as a value; or, it could have used a nested map, e.g. `:jvm-opts {:heap {:max 512 :min 64}}`.

So for the sake of clarity we will always refer to the key-value pairs in the project map as *parameters*, and refer to the software state controlled by such parameter A given parameter is not necessarily specific to a particular software component, whereas options are always software-specific.

Chapter
2

Tasks

Topics:

Overview ...

Preinstalled tasks

Overview ...

Leiningen 2.3.2 Standard Tasks	
check	Check syntax and warn on reflection.
classpath	Write the classpath of the current project to output-file.
clean	Remove all files from paths in project's clean-targets.
compile	Compile Clojure source into .class files.
deploy	Deploy jar and pom to remote repository.
deps	Show details about dependencies.
difftest	
do	Higher-order task to perform other tasks in succession.
help	Display a list of tasks or help for a given task or subtask.
install	Install current project to the local repository.
jar	Package up all the project's files into a jar file.
javac	Compile Java source files.
localrepo	Work with local Maven repository
marg	Run Marginalia against your project source files.
mustache	Evaluate a Mustache template.
new	Generate scaffolding for a new project based on a template.
plugin	DEPRECATED. Please use the :user profile instead.
pom	Write a pom.xml file to disk for Maven interoperability.
pprint	Pretty-print a representation of the project map.
repl	Start a repl session either with the current project or standalone.
retest	Run only the test namespaces which failed last time around.
run	Run the project's -main function.
search	Search remote maven repositories for matching jars.
show-profiles	List all available profiles or display one if given an argument.
swank	obsolete??
test	Run the project's tests.
trampoline	Run a task without nesting the project's JVM inside Leiningen's.
uberjar	Package up the project files and all dependencies into a jar file.
update-in	Perform arbitrary transformations on your project map.
upgrade	Upgrade Leiningen to specified version or latest stable.
version	Print version for Leiningen and the current JVM.
with-profile	Apply the given task with the profile(s) specified.

User-defined Tasks: Plugins

Overview ...

Chapter
3

Dependency Management

Overview ...

Chapter
4

Profiles

Overview ...

Chapter
5

Templates

All you really need to know is `lein new`.

A word here about how Leiningen finds templates, and about clojars, so the user can troubleshoot in case something goes wrong.

See the Developer's Manual for information on how to write your own templates.

Chapter
6

REPL Workflow

Overview ...

Chapter
7

Testing

Overview ...

Chapter
8

Deployment

Overview ...

Appendix
A

Contributing to this document

This is a DITA document. The original is available from the *github project*.

Please use the *Issues Tracker* to register bugs, corrections, enhancement requests, etc. Alternatively, you can clone the repository and edit the text directly. If you do please credit yourself in the metadata.

Intellectual property

DITA

DITA (Darwin Information Typing Architecture) is an *OASIS Standard* for writing, managing, and publishing information. It supports "topic-based" authoring.

The *DITA Wiki Knowledgebase* contains lots of information on DITA, from introductions and overviews to detailed documentation.

Another good overview is *DITA for the Impatient* from *XMLMind*.

The *DITA Language Specification* documents the elements and atributes of the DITA Schema.

Tools

There are two open source DITA transformation tools available:

- *DITA Open Toolkit*
- *XMLMind DITA Converter*

Index