Subversion Import Wizard User Manual

Edition 1 (draft), 2013-03-14

Ralph Schleicher

This is edition 1 (draft), last updated 2013-03-14, of the *Subversion Import Wizard User Manual*, for Subversion Import Wizard version 1.0.

Please report any errors in this manual to `rs@ralph-schleicher.de`.

Table of Contents

1 Introduction

The utility program `subversion-import-wizard`, helps the user to import a project history from a file system into a Subversion repository. To facilitate this, the `subversion-import-wizard` program takes one or more *command files* as input and evaluates the commands specified in them.

A command file is coded up by the user and its syntax is described in Chapter 2 [Command File], page 3.

The resulting Subversion repository is always created as a local repository. When finished, you can move this repository to your Subversion server for general availability.

1.1 An Illustrative Example

Before we explain the gory details, let's have a look at a simple example. Suppose you are working on the Foo project since some time and you now notice that it is time to manage your project files with Subversion. You have already released two versions which you have saved in separate directories in the file system and you have another directory containing the current status of the project.

You can of course use Subversion's import feature to copy the current status of the project from the file system into the repository but that means that you loose the history of the project. The alternative is to import the project files of the oldest available version into the repository and then commit successive versions by hand. This preserves the history of the project but the procedure is booring and error prone.

And now is the time to use the Subversion Import Wizard utility. Therefore, you fire up your favorite text editor and create a file with the following contents.

```
;; Import history of the Foo project.

(import "foo-20120401" trunk)
(tag trunk "foo-1.0" :message "Version 1.0")

(import "foo-20120401-new" trunk :message "Fix unit conversion bug")
(tag trunk "foo-1.1" :message "Version 1.1")

(import "foo" trunk :message "Import current status")
```

A quick word on the syntax of this file. Comments start with a semi-colon character and last until the end of the line. Subversion Import Wizard commands are written between a pair of parenthesis. The first word of a command is the command name and the remaining words are the command arguments. If this syntax remind you of a Lisp program, then you are right. It actually is a Lisp program. To be true, a Subversion Import Wizard command file is read[1] and evaluated by a Common Lisp system. The reason to provide a fully fledged Common Lisp system as the shell for Subversion Import Wizard commands is to make simple tasks easy and complex tasks possible.

Let's go back and have a look what these commands are about. The first `import` command copies the contents of the directory `foo-20120401` from the file system into the *trunk* directory of the repository. When the *trunk* directory is empty, this is equal to the Subversion command

```
svn import foo-20120401 ^/trunk
```

After that the `tag` command creates a label for this revision. Again, this is equal to the Subversion command

[1] The full truth is that commands are read in the `subversion-import-wizard-user` package which defines the Subversion Import Wizard symbols but also shadows Common Lisp symbols like `import`.

```
svn copy ^/trunk ^/tags/foo-1.0
```

The second `import` command copies the contents of the directory `foo-20120401-new` from the file system into the *trunk* directory of the repository. Although it has the same form as the first `import` command, what happens behind the scene is quite different. Files and directories are automatically added or deleted so that, after the `import` command finished, the *trunk* directory resembles the `foo-20120401-new` directory. Again, the second `tag` command creates a label for this revision.

Finally, the third `import` command copies the contents of the current working directory `foo` from the file system into the *trunk* directory of the repository.

To actually execute these commands you have to save the command file as, for example, `foo-import.txt` and invoke the `subversion-import-wizard` program on it.

```
subversion-import-wizard
 foo-import.txt
```

2 Command File

A command file is a text file coded up by the user.

Comments

Comments start with a semi-colon character. The semi-colon and all characters up to and including the next end-of-line character or the end of the file are ignored.

See also Semicolon.

Numbers

`13`, `-13` Integral numbers.

`1/3`, `-1/3` Rational numbers.

`1.2`, `-1.2`, `1E+3`, `1E-3`
Floating-point numbers.

See also Numbers as Tokens.

Strings

Strings are delimited by double-quote characters. The backslash character \ is the escape character.

`""` The empty string.

`"foo"` A non-empty string.

`"Say \"No\"!"`
A string containing double-quote characters.

`"foo\\bar"`
A string containing a backslash character.

See also Double-Quote.

Symbols

`foo`, `bar` Simple symbols.

`foo-bar`, `foo*`, `%bar`
Symbol containing a minus sign.

`:foo` A keyword.

2.1 Symbol Reference

The predefined Subversion Import Wizard variables and functions are described in this section. For simple tasks, that's all you need to know. The rest, and that's a lot, is documented in the Common Lisp HyperSpec.

`~` *arg* . . . [Function]
Concatenate parts of a file system path.

The `~` function takes any number of arguments (which have to be strings) and constructs a file system path of it.

The return value is a string. If no arguments are given, the return value is the empty string.

The result of the `~` function depends on your operating system's file system path separator.

On Windows, you get

```
(~ "foo" "bar" "baz")
 ⇒ "foo\bar\baz"
```

whereas on Unix, you get

```
(~ "foo" "bar" "baz")
 ⇒ "foo/bar/baz"
```

`^` *arg ...* [Function]

Concatenate parts of a repository path.

The `^` function takes any number of arguments (which have to be strings) and constructs a repository path of it.

The return value is a string. If no arguments are given, the return value is the empty string.

Contrary to the `~` function, the result of the `^` function does not depend on your operating system because it refers to a path in a Subversion repository.

```
(^ "foo" "bar" "baz")
 ⇒ "foo/bar/baz"
```

`trunk` [Special Variable]

The repository path of the trunk directory. Default value is `"trunk"`.

For example,

```
(^ trunk "foo")
 ⇒ "trunk/foo"
```

`branches` [Special Variable]

The repository path of the branches directory. Default value is `"branches"`.

For example,

```
(^ branches "bar")
 ⇒ "branches/bar"
```

`tags` [Special Variable]

The repository path of the tags directory. Default value is `"tags"`.

For example,

```
(^ tags "baz")
 ⇒ "tags/baz"
```

`mkdir` *dir* `&key` *message* [Subversion Command]

Create a directory in the repository, if it does not exist. Intermediate directories are made as needed.

- First argument *dir* is the repository path of the directory.
- Keyword argument *message* is the commit message. Default is `"Create directory"`.

For example,

```
(mkdir "branches/users/john" :message "Create user directory for John Doe")
```

`copy` *from to* `&key` *message* [Subversion Command]

Duplicate a directory in the repository, remembering history.

- First argument *from* is the repository path of the source directory.
- Second argument *to* is the repository path of the destination directory.
- Keyword argument *message* is the commit message. Default is `"Create copy"`.

For example,

```
(copy "trunk" "branches/foo" :message "Create foo branch")
```

`import` *from to* `&key` *ignore message* [Subversion Command]

Copy an unversioned directory from the file system into the repository.

- First argument *from* is the file system path of the source directory.
- Second argument *to* is the repository path of the destination directory.
- If keyword argument *ignore* is true, omit ignored files and directories. Default is to import all files and directories.
- Keyword argument *message* is the commit message.

The source directory has to exist. If the destination directory does not exist, it is created. If the destination directory is not empty, files and directories are added, deleted, or updated as needed to match the contents of the source directory.

For example,

```
(import "foo" "trunk" :ignore t :message "Initial import")
```

3 Invoking subversion-import-wizard

The subversion-import-wizard command reads commands from input files and evaluates them. Its arguments are as follows:

```
subversion-import-wizard
 [option...] filename...
```

The subversion-import-wizard program writes diagnostic messages to standard error. If non-recoverable errors occur while reading an input file, subversion-import-wizard terminates with an exit status indicating failure. Diagnostic messages have the form

```
filename:linenumber: message
```

where *filename* and *linenumber* point to the location of the error.

An exit status of 0 means success and 1 means trouble.

3.1 subversion-import-wizard Options

Below is a summary of all of the options that subversion-import-wizard accepts. Some options have two equivalent names, one of which is a single letter preceded by a single hyphen '-', and the other of which is a long name preceded by two hyphens '--'. Multiple single letter options (unless they take an argument) can be combined into a single command line word. Long named options can be abbreviated to any unique prefix of their name.

`--repository=LOCATION`
: Location of the Subversion repository. Default is to create a local repository.

`--working-copy=DIRNAME`
: Directory name of the working copy. Default is to create a temporary directory.

`-n`
`--dry-run`
: Do not execute any command. This option is useful for testing and debugging.

`-w`
`--window-system`
: Interact with the user via a graphical user interface.

  Subversion Import Wizard attempts to guess if it has been invoked from the command line or from a window system. You can use the `-w` option to force the later.

`--help` Print a short help text and exit successfully.

`--version`
: Print the version number and exit successfully.

Concept Index

Command Index