Tin: A Tcl Package Manager

Version 1.1

Alex Baker

https://github.com/ambaker1/Tin

October 19, 2023

Abstract

Tin is a Tcl package that installs Tcl packages and their dependencies directly from GitHub. Additionally, Tin provides various utilities for Tcl package ecosystem development.

The Tin List

Tin installs packages from GitHub repositories based on entries in the "Tin List". Tin comes pre-packaged with a list of packages available for download, and additional entries can be added to in the current session with the commands *tin add*.

```
tin add <-tin> $name $version $repo $tag $file
```

-tin	Option to add an entry to the Tin List. Default.
$name	Package name.
$version	Package version.
$repo	Github repository URL.
$tag	Github release tag for version.
$file	Installer file path in repo.

Example 1: Adding entries to the Tin List

Code:

```
tin add foo 1.0 https://github.com/username/foo v1.0 install_foo.tcl
tin add foo 1.1 https://github.com/username/foo v1.1 install_foo.tcl
tin add foo 1.2 https://github.com/username/foo v1.2 install_foo.tcl
tin add foo 1.2.1 https://github.com/username/foo v1.2.1 install_foo.tcl
tin add foo 1.2.2 https://github.com/username/foo v1.2.2 install_foo.tcl
tin add foo 2a0 https://github.com/username/foo v2a0 install_foo.tcl
tin add foo 2.0 https://github.com/username/foo v2.0 install_foo.tcl
tin add foo 2.0.1 https://github.com/username/foo v2.0.1 install_foo.tcl
```

Auto-Tin Configuration

To streamline the process of populating the Tin List, simply use the command *tin add -auto* to specify an Auto-Tin configuration. An Auto-Tin configuration specifies a template for Tin entries, and this template is used in conjunction with GitHub release tags to automatically populate the Tin List.

```
tin add -auto $name $repo $file <<-exact> $version> <$reqs ...>
```

-auto	Option to add an Auto-Tin configuration to the Tin List.
$name	Package name.
$repo	Github repository URL.
$file	Installer file path in repo.
$version	Package version (-exact specifies exact version).
$reqs ...	Package version requirements, mutually exclusive with -exact option.

Auto-Tin Tag Pattern

In order for an Auto-Tin configuration to work, the GitHub repository must have release tags corresponding directly with the package versions, such as "v1.2.3".

To be specific, version release tags must match the Auto-Tin tag pattern:

```
^v(0|[1-9]\d*)(\.(0|[1-9]\d*))*([ab](0|[1-9]\d*)(\.(0|[1-9]\d*))*)?$
```

Note that this is not the same as "SemVer" (https://semver.org/), which is the standard version number format. This is because the Auto-Tin tag pattern specifically matches all version numbers compatible with Tcl, with a prefix of "v". Most notably, the format for alpha and beta versions is different in Tcl, where "a" and "b" replace one of the periods (e.g. "v2a0"), and effectively represent ".-2." and ".-1.", respectively.

Fetching and Auto-Fetching

If a package is configured as an Auto-Tin package, the Tin List can be automatically populated with versions available for installation with the command *tin fetch*. If the -all option is specified, it will return a dictionary of package names and versions added. Otherwise, it will simply return a list of the versions added for the package.

```
tin fetch $name <$pattern>
tin fetch -all <$names>
```

$name	Package name.
$pattern	Version number glob pattern for *git ls-remote*. Default "*", or all versions.
-all	Option to fetch all available versions.
$names	List of package names. Default all Auto-Tin packages.

Additionally, by default, Tin auto-fetches on the fly when it cannot find Tin entries satisfying version requirements. This can be toggled on/off with *tin auto*, which also returns the current auto-fetch status.

```
tin auto <$toggle>
```

$toggle	Boolean, whether to auto-fetch. Default true.

Example 2: Adding an Auto-Tin configuration and fetching Tin entries

Code:

```
tin add -auto foo https://github.com/username/foo install_foo.tcl
tin fetch foo; # fetches all entries from GitHub
```

Removing from the Tin List

Both Tin entries and Auto-Tin configurations can be removed from the Tin List in a session with the command *tin remove*.

```
tin remove $name ...
tin remove -tin $name <$version> <$repo>
tin remove -auto $name <$repo> <$file>
```

`$name`	Package name.
`-tin`	Option to only remove from Tin.
`-auto`	Option to only remove from Auto-Tin.
`$version`	Package version to remove (optional, default all versions).
`$repo`	Repository to remove (optional, default all repositories).
`$file`	Installer file path to remove (optional, default all installer files).

Clearing the Tin List

The command *tin clear* removes all entries and Auto-Tin configurations from the Tin List.

```
tin clear
```

Saving and Resetting the Tin List

The state of the Tin List can be saved for future sessions with *tin save*, and reset to default or factory settings with *tin reset*. Note that *tin save* saves to a hidden user-config file located in the user's home directory.

```
tin save
```

```
tin reset <-soft>
tin reset -hard
```

-soft	Option to reset to user settings (default).
-hard	Option to reset to factory settings.

Example 3: Saving changes to the Tin List

Code:

```
tin reset -hard
tin add foo 1.0 https://github.com/username/foo v1.0 install_foo.tcl
tin save
```

Output:
"∼/.tinlist.tcl" :

```
tin add foo 1.0 https://github.com/username/foo v1.0 install_foo.tcl
```

Available Packages

The available packages in the Tin List can be queried with the command *tin packages*.

```
tin packages <$pattern>
tin packages -tin <$pattern>
tin packages -auto <$pattern>
```

$pattern	Optional "glob" pattern, default "*", or all packages.
-tin	Option to search only for Tin packages.
-auto	Option to search only for Auto-Tin packages.

Example 4: Getting available packages

Code:

```
set allPackages [tin packages]
set loadedPackages [tin packages -tin]
set autoPackages [tin packages -auto]
```

Available Package Versions

A list of available versions for a Tin package that satisfy version requirements can be queried with the command *tin versions*. Similarly, the command *tin available* returns the version that would be installed with *tin install*, or blank if none can be found.

```
tin versions $name <<-exact> $version> <$reqs ...>
```

```
tin available $name <<-exact> $version> <$reqs ...>
```

$name	Package name.
$version	Package version (-exact specifies exact version).
$reqs ...	Package version requirements, mutually exclusive with -exact option.

Accessing the Tin List

The command *tin get* queries basic information about Tin, and returns blank if the requested entry does exist. Similar to *tin add* and *tin remove*, it has two forms, one for querying Tin packages and one for querying Auto-Tin packages. Returns a dictionary associated with the supplied arguments.

```
tin get <-tin> $name <$version> <$repo>
tin get -auto $name <$repo> <$file>
```

$name	Package name.
-tin	Option to query Tin packages (default).
-auto	Option to query Auto-Tin packages.
$version	Package version.
$repo	Github repository URL.
$file	Installer file path in repo.

Example 5: Getting info from the Tin List

Code:

```
package require tin
tin add foo 1.0 https://github.com/username/foo v1.0 install_foo.tcl
puts [tin get foo]
```

Output:

```
1.0 {https://github.com/username/foo {v1.0 install_foo.tcl}}
```

Installing and Uninstalling Packages

The command *tin install* installs packages directly from GitHub, and returns the version installed.

The command *tin depend* installs packages only if they are not installed, and returns the version number installed (useful for dependencies in installation scripts).

The command *tin uninstall* uninstalls packages (as long as they are in the Tin List), and returns blank if successful, or error if it was unsuccessful in uninstalling the package.

```
tin install $name <<-exact> $version> <$reqs ...>
```

```
tin depend $name <<-exact> $version> <$reqs ...>
```

```
tin uninstall $name <<-exact> $version> <$reqs ...>
```

$name	Package name.
$version	Package version (-exact specifies exact version).
$reqs ...	Package version requirements, mutually exclusive with -exact option.

By default, uninstalling a package simply deletes the library folder associated with the package. However, if a "pkgUninstall.tcl" file is located within the package folder, it will run that file instead, with the variable `$dir` set to the package library folder path, similar to how "pkgIndex.tcl" files work.

Example 6: Complex uninstall file "pkgUninstall.tcl"

Code:

```
set bindir [file dirname [info nameofexecutable]]
file delete [file join $bindir foo.bat]; # delete file in the bin directory
file delete -force $dir; # Clean up package
```

Upgrading Packages

Upgrading packages involves first installing the new version with *tin install*, then uninstalling the old version with *tin uninstall*. The command *tin check* returns an upgrade list of available minor and patch upgrades for packages. The command *tin upgrade* checks for available upgrades with *tin check*, and then upgrades the packages, returning the upgrade list. If there is no upgrade available, the upgrade list will be empty. If the -all option is specified, the format of the upgrade list is "`name {old new ...} ...`". Otherwise, the format of the upgrade list is "`old new`".

```
tin check $name <<-exact> $version> <$reqs ...>
tin check -all <$names>
```

```
tin upgrade $name <<-exact> $version> <$reqs ...>
tin upgrade -all <$names>
```

`-all`	Option to look for upgrades in all installed major versions of the packages.
`$names`	Package names. Default searches all packages in the Tin List.
`$name`	Package name.
`$version`	Package version (-exact specifies exact version).
`$reqs ...`	Package version requirements, mutually exclusive with -exact option.

Example 7: Upgrading Tin

Code:

```
# Upgrade Tin
package require tin
tin upgrade tin
```

Loading and Importing Packages

The command *tin require* is similar to the Tcl command *package require*, but with the added feature that if the package is missing, it will try to install it with *tin install*.

The command *tin import* additionally handles most use-cases of *namespace import*. Both *tin require* and *tin import* return the version number of the package imported.

```
tin require $name <<-exact> $version> <$reqs ...>
```

$name	Package name.
$version	Package version (-exact specifies exact version).
$reqs ...	Package version requirements, mutually exclusive with -exact option.

```
tin import <-force> <$patterns from> $name <<-exact> $version> <$reqs ...> <as $ns>
```

-force	Option to overwrite existing commands.
$patterns	Commands to import, or "glob" patterns, default "*", or all commands.
$name	Package name.
$version	Package version (-exact specifies exact version).
$reqs ...	Package version requirements, mutually exclusive with -exact option.
$ns	Namespace to import into. Default global namespace, or "::".

Example 8: Importing all commands package "foo"

Code:

```
package require tin
tin import foo 1.0
```

Generic Package Utilities

Tin works alongside the Tcl *package* commands, and provides a few package utility commands that do not interface with the Tin List.

Check if a Package is Installed

The command *tin installed* returns the package version number that would be loaded with *package require*, or blank if the package is not installed. If there is no package version in the Tcl package database satisfying the requirements, it will call the *package unknown* script to load *package ifneeded* statements from "pkgIndex.tcl" files, just like what *package require* does, but without loading the package.

```
tin installed $name <<-exact> $version> <$reqs ...>
```

$name	Package name.
$version	Package version (-exact specifies exact version).
$reqs ...	Package version requirements, mutually exclusive with -exact option.

Unload a Package/Namespace

The command *tin forget* is short-hand for both *package forget* and *namespace delete*, but it will not throw an error if there is no namespace corresponding with the package name. It is especially useful for reloading packages within an instance of Tcl.

```
tin forget $name ...
```

$name	Package name.

Example 9: Loading and reloading a package

Code:

```
package require foo
tin forget foo
package require foo
```

Utilities for Package Development

In addition to commands for installing and loading packages, Tin provides a few commands intended to help in writing installation and build files for your packages.

Creating Package Directories

The command *tin mkdir* creates a library directory to install a package in, with a normalized naming convention that allows it to be uninstalled easily with *tin uninstall*.

```
tin mkdir <-force> <$basedir> $name $version
```

-force	Option to create fresh library directory (deletes existing folder).
$basedir	Base directory, default one folder up from the Tcl library folder.
$name	Package name.
$version	Package version.

See the example installation file for a package "foo" that requires the package "bar 1.2", and installs in library folder "foo-1.0".

Example 10: Example file "install_foo.tcl"

Code:

```
package require tin
tin depend bar 1.2
set dir [tin mkdir -force foo 1.0]
file copy README.md $dir
file copy LICENSE $dir
file copy lib/bar.tcl $dir
file copy lib/pkgIndex.pdf $dir
```

Building Library Files from Source with Configuration Variable Substitution

The command *tin bake* takes an input text file, and writes an output text file after substitution of configuration variables such as `@VERSION@`. This is especially helpful for ensuring that the package version is consistent across the entire project. If a source directory is used as input, it will batch bake all ".tin" files.

```
tin bake $src $target $config
tin bake $src $target $varName $value ...
```

`$src`	Source file, or directory with ".tin" files.
`$target`	Target file, or directory to write ".tcl" files to.
`$config`	Dictionary of config variable names and values. Config variables must be uppercase alphanumeric.
`$varName $value ...`	Config variable names and values. Mutually exclusive with **`$config`**.

See below for an example of how *tin bake* can be used to automatically update a "pkgIndex.tcl" file:

Example 11: Building a "pkgIndex.tcl" file

Code:

```
package require tin
tin bake pkgIndex.tin pkgIndex.tcl {VERSION 1.0}
```

Output:

"pkgIndex.tin" :

```
package ifneeded foo @VERSION@ [list source [file join $dir foo.tcl]]
```

"pkgIndex.tcl" :

```
package ifneeded foo 1.0 [list source [file join $dir foo.tcl]]
```

Basic Unit Testing

The command *assert* can be used for basic unit testing of Tcl scripts. It throws an error if the statement is false. Otherwise, it simply returns nothing and the script continues.

There are two forms of this command, one which passes input through the Tcl *expr* command, and the other which does value comparison.

```
tin assert $expr <$message>
tin assert $value $op $expected <$message>
```

$expr	Tcl math expression to evaluate as boolean.
$value	Value to compare.
$op	Comparison operator, or "is" to assert type.
$expected	Comparison value.
$message	Optional error message to add context to assertion error. Default blank.

Example 12: Asserting values and types

Code:

```
tin assert {2 + 2 == 4}; # Asserts that math works
tin assert 5.0 is double; # Asserts that 5.0 is indeed a number
tin assert {hello world} is integer; # This is false
```

Output:

```
expected integer value but got "hello world"
```

Example 13: Provide context to invalid procedure input

Code:

```
proc subtract {x y} {
    tin assert $x > $y {x must be greater than y}
    expr {$x - $y}
}
subtract 2.0 3.0
```

Output:

```
x must be greater than y
assert 2.0 > 3.0 failed
    while executing
"subtract 2.0 3.0"
```

For more advanced unit testing, the built-in *tcltest* package is recommended.