Stow 2.4.0

Managing the installation of software packages

Bob Glickstein, Zanshin Software, Inc.
Kahlil Hodgson, RMIT University, Australia.
Guillaume Morin
Adam Spiers

This manual describes GNU Stow version 2.4.0 (7 April 2024), a program for managing farms of symbolic links.

Table of Contents

1 Introduction

GNU Stow is a symlink farm manager which takes distinct sets of software and/or data located in separate directories on the filesystem, and makes them all appear to be installed in a single directory tree.

Originally Stow was born to address the need to administer, upgrade, install, and remove files in independent software packages without confusing them with other files sharing the same file system space. For instance, many years ago it used to be common to compile programs such as Perl and Emacs from source and install them in `/usr/local`. When one does so, one winds up with the following files[1] in `/usr/local/man/man1`:

```
a2p.1
ctags.1
emacs.1
etags.1
h2ph.1
perl.1
s2p.1
```

Now suppose it's time to uninstall Perl. Which man pages get removed? Obviously `perl.1` is one of them, but it should not be the administrator's responsibility to memorize the ownership of individual files by separate packages.

The approach used by Stow is to install each package into its own tree, then use symbolic links to make it appear as though the files are installed in the common tree. Administration can be performed in the package's private tree in isolation from clutter from other packages. Stow can then be used to update the symbolic links. The structure of each private tree should reflect the desired structure in the common tree; i.e. (in the typical case) there should be a `bin` directory containing executables, a `man/man1` directory containing section 1 man pages, and so on.

While this is useful for keeping track of system-wide and per-user installations of software built from source, in more recent times software packages are often managed by more sophisticated package management software such as `rpm` (`https://en.wikipedia.org/wiki/Rpm_(software)`), `dpkg` (`https://en.wikipedia.org/wiki/Dpkg`), and Nix (`https://en.wikipedia.org/wiki/Nix_package_manager`) / GNU Guix (`https://en.wikipedia.org/wiki/GNU_Guix`), or language-native package managers such as Ruby's `gem` (`https://en.wikipedia.org/wiki/RubyGems`), Python's `pip` (`https://en.wikipedia.org/wiki/Pip_(package_manager)`), Javascript's `npm` (`https://en.wikipedia.org/wiki/Npm_(software)`), and so on.

However Stow is still used not only for software package management, but also for other purposes, such as facilitating a more controlled approach to management of configuration files in the user's home directory[2], especially when coupled with version control systems[3].

Stow was inspired by Carnegie Mellon's Depot program, but is substantially simpler and safer. Whereas Depot required database files to keep things in sync, Stow stores no extra

[1] As of Perl 4.036 and Emacs 19.22. These are now ancient releases but the example still holds valid.

[2] `http://brandon.invergo.net/news/2012-05-26-using-gnu-stow-to-manage-your-dotfiles.html`

[3] `http://lists.gnu.org/archive/html/info-stow/2011-12/msg00000.html`

state between runs, so there's no danger (as there was in Depot) of mangling directories when file hierarchies don't match the database. Also unlike Depot, Stow will never delete any files, directories, or links that appear in a Stow directory (e.g., `/usr/local/stow/emacs`), so it's always possible to rebuild the target tree (e.g., `/usr/local`).

Stow is implemented as a combination of a Perl script providing a CLI interface, and a backend Perl module which does most of the work.

For information about the latest version of Stow, you can refer to `http://www.gnu.org/software/stow/`.

2 Terminology

A *package* is a related collection of files and directories that you wish to administer as a unit — e.g., Perl or Emacs — and that needs to be installed in a particular directory structure — e.g., with `bin`, `lib`, and `man` subdirectories.

A *target directory* is the root of a tree in which one or more packages wish to *appear* to be installed. `/usr/local` is a common choice for this, but by no means the only such location. Another common choice is `~` (i.e. the user's `$HOME` directory) in the case where Stow is being used to manage the user's configuration ("dotfiles") and other files in their `$HOME`. The examples in this manual will use `/usr/local` as the target directory.

A *stow directory* is the root of a tree containing separate packages in private subtrees. When Stow runs, it uses the current directory as the default stow directory. The examples in this manual will use `/usr/local/stow` as the stow directory, so that individual packages will be, for example, `/usr/local/stow/perl` and `/usr/local/stow/emacs`.

An *installation image* is the layout of files and directories required by a package, relative to the target directory. Thus, the installation image for Perl includes: a `bin` directory containing `perl` and `a2p` (among others); an `info` directory containing Texinfo documentation; a `lib/perl` directory containing Perl libraries; and a `man/man1` directory containing man pages.

> **Note:** This is a *pre*-installation image which exists even before Stow has installed any symlinks into the target directory which point to it.

A *package directory* is the root of a tree containing the installation image for a particular package. Each package directory must reside in a stow directory — e.g., the package directory `/usr/local/stow/perl` must reside in the stow directory `/usr/local/stow`. The *name* of a package is the name of its directory within the stow directory — e.g., `perl`.

Thus, the Perl executable might reside in `/usr/local/stow/perl/bin/perl`, where `/usr/local` is the target directory, `/usr/local/stow` is the stow directory, `/usr/local/stow/perl` is the package directory, and `bin/perl` within is part of the installation image.

A *symlink* is a symbolic link, i.e. an entry on the filesystem whose path is sometimes called the *symlink source*, which points to another location on the filesystem called the *symlink destination*. There is no guarantee that the destination actually exists.

In general, symlinks can be *relative* or *absolute*. A symlink is absolute when the destination names a full path; that is, one starting from `/`. A symlink is relative when the destination names a relative path; that is, one not starting from `/`. The destination of a relative symlink is computed starting from the symlink's own directory, i.e. the directory containing the symlink source.

> **Note:** Stow only creates symlinks within the target directory which point to locations *outside* the target directory and inside the stow directory.
>
> Consequently, we avoid referring to symlink destinations as symlink *targets*, since this would result in the word "target" having two different meanings:
>
> 1. the target directory, i.e. the directory into which Stow targets installation, where symlinks are managed by Stow, and

2. the destinations of those symlinks.

If we did not avoid the second meaning of "target", then it would lead to confusing language, such as describing Stow as installing symlinks into the target directory which point to targets *outside* the target directory.

Similarly, the word "source" can have two different meanings in this context:

1. the installation image, or some of its contents, and
2. the location of symlinks (the "source" of the link, vs. its destination).

Therefore it should also be avoided, or at least care taken to ensure that the meaning is not ambiguous.

3 Invoking Stow

The syntax of the `stow` command is:

```
stow [options] [action flag] package ...
```

Each *package* is the name of a package (e.g., '`perl`') in the stow directory that we wish to install into (or delete from) the target directory. The default action is to install the given packages, although alternate actions may be specified by preceding the package name(s) with an *action flag*.

The following options are supported:

'`-d dir`'
'`--dir=dir`'
: Set the stow directory to *dir*. Defaults to the value of the environment variable `STOW_DIR` if set, or the current directory otherwise.

'`-t dir`'
'`--target=dir`'
: Set the target directory to *dir* instead of the parent of the stow directory. Defaults to the parent of the stow directory, so it is typical to execute `stow` from the directory `/usr/local/stow`.

'`--ignore=regexp`'
: This (repeatable) option lets you suppress acting on files that match the given Perl regular expression. For example, using the options

  ```
  --ignore='.*\.orig' --ignore='.*\.dist'
  ```

  will cause stow to ignore files ending in `.orig` or `.dist`.

  Note that the regular expression is anchored to the end of the filename, because this is what you will want to do most of the time.

  Also note that by default Stow automatically ignores a "sensible" built-in list of files and directories such as `CVS`, editor backup files, and so on. See Chapter 4 [Ignore Lists], page 8, for more details.

'`--defer=regexp`'
: This (repeatable) option avoids stowing a file matching the given regular expression, if that file is already stowed by another package. This is effectively the opposite of `--override`.

  (N.B. the name `--defer` was chosen in the sense that the package currently being stowed is treated with lower precedence than any already installed package, not in the sense that the operation is being postponed to be run at a later point in time; do not confuse this nomenclature with the wording used in [Deferred Operation], page 14.)

  For example, the following options

  ```
  --defer=man --defer=info
  ```

  will cause stow to skip over pre-existing man and info pages.

  Equivalently, you could use '`--defer='man|info''` since the argument is just a Perl regular expression.

Note that the regular expression is anchored to the beginning of the path relative to the target directory, because this is what you will want to do most of the time.

'`--override=`*regexp*'
: This (repeatable) option forces any file matching the regular expression to be stowed, even if the file is already stowed to another package. For example, the following options

```
--override=man --override=info
```

will permit stow to overwrite links that point to pre-existing man and info pages that are owned by stow and would otherwise cause a conflict.

The regular expression is anchored to the beginning of the path relative to the target directory, because this is what you will want to do most of the time.

'`--dotfiles`'
: Enable special handling for *dotfiles* (files or folders whose name begins with a period) in the package directory. If this option is enabled, Stow will add a preprocessing step for each file or folder whose name begins with '`dot-`', and replace the '`dot-`' prefix in the name by a period '`.`'. This is useful when Stow is used to manage collections of dotfiles, to avoid having a package directory full of hidden files.

For example, suppose we have a package containing two files, `stow/dot-bashrc` and `stow/dot-emacs.d/init.el`. With this option, Stow will create symlinks from `.bashrc` to `stow/dot-bashrc` and from `.emacs.d/init.el` to `stow/dot-emacs.d/init.el`. Any other files, whose name does not begin with '`dot-`', will be processed as usual.

'`--no-folding`'
: This disables any further tree folding (see [tree folding], page 11) or refolding (see [tree refolding], page 13). If a new subdirectory is encountered whilst stowing a new package, the subdirectory is created within the target, and its contents are symlinked, rather than just creating a symlink for the directory. If removal of symlinks whilst unstowing a package causes a subtree to be foldable (i.e. only containing symlinks to a single package), that subtree will not be removed and replaced with a symlink.

'`--adopt`'
: **Warning!** This behaviour is specifically intended to alter the contents of your stow directory. If you do not want that, this option is not for you.

When stowing, if a target is encountered which already exists but is a plain file (and hence not owned by any existing stow package), then normally Stow will register this as a conflict and refuse to proceed. This option changes that behaviour so that the file is moved to the same relative place within the package's installation image within the stow directory, and then stowing proceeds as before. So effectively, the file becomes adopted by the stow package, without its contents changing.

This is particularly useful when the stow package is under the control of a version control system, because it allows files in the target tree, with potentially different contents to the equivalent versions in the stow package's installation

image, to be adopted into the package, then compared by running something like '`git diff ...`' inside the stow package, and finally either kept (e.g. via '`git commit ...`') or discarded ('`git checkout HEAD ...`').

'`-n`'
'`--no`'
'`--simulate`'
: Do not perform any operations that modify the file system; in combination with `-v` can be used to merely show what would happen.

'`-v`'
'`--verbose[=n]`'
: Send verbose output to standard error describing what Stow is doing. Verbosity levels are from 0 to 5; 0 is the default. Using `-v` or `--verbose` increases the verbosity by one; using '`--verbose=n`' sets it to *n*.

'`-p`'
'`--compat`'
: Scan the whole target tree when unstowing. By default, only directories specified in the *installation image* are scanned during an unstow operation. Previously Stow scanned the whole tree, which can be prohibitive if your target tree is very large, but on the other hand has the advantage of unstowing previously stowed links which are no longer present in the installation image and therefore orphaned. This option restores the legacy behaviour; however, the `--badlinks` option to the `chkstow` utility may be a better way of ensuring that your installation does not have any dangling symlinks (see Chapter 10 [Target Maintenance], page 17).

'`-V`'
'`--version`'
: Show Stow version number, and exit.

'`-h`'
'`--help`' Show Stow command syntax, and exit.

The following *action flags* are supported:

'`-D`'
'`--delete`'
: Delete (unstow) the package name(s) that follow this option from the *target directory*. This option may be repeated any number of times.

'`-R`'
'`--restow`'
: Restow (first unstow, then stow again) the package names that follow this option. This is useful for pruning obsolete symlinks from the target tree after updating the software in a package. This option may be repeated any number of times.

'`-S`'
: explictly stow the package name(s) that follow this option. May be omitted if you are not using the `-D` or `-R` options in the same invocation. See Chapter 8 [Mixing Operations], page 15, for details of when you might like to use this feature. This option may be repeated any number of times.

4 Ignore Lists

4.1 Motivation For Ignore Lists

In many situations, there will exist files under the package directories which it would be undesirable to stow into the target directory. For example, files related version control such as `.gitignore`, `CVS`, `*,v` (RCS files) should typically not have symlinks from the target tree pointing to them. Also there may be files or directories relating to the build of the package which are not needed at run-time.

In these cases, it can be rather cumbersome to specify a `--ignore` parameter for each file or directory to be ignored. This could be worked around by ensuring the existence of `~/.stowrc` containing multiple `--ignore` lines, or if a different set of files/directories should be ignored depending on which stow package is involved, a `.stowrc` file for each stow package, but this would require the user to ensure that they were in the correct directory before invoking stow, which would be tedious and error-prone. Furthermore, since Stow shifts parameters from `.stowrc` onto ARGV at run-time, it could clutter up the process table with excessively long parameter lists, or even worse, exceed the operating system's limit for process arguments.

Therefore in addition to `--ignore` parameters, Stow provides a way to specify lists of files and directories to ignore.

4.2 Types And Syntax Of Ignore Lists

If you put Perl regular expressions, one per line, in a `.stow-local-ignore` file within any top level package directory, in which case any file or directory within that package matching any of these regular expressions will be ignored. In the absence of this package-specific ignore list, Stow will instead use the contents of `~/.stow-global-ignore`, if it exists. If neither the package-local or global ignore list exist, Stow will use its own built-in default ignore list, which serves as a useful example of the format of these ignore list files:

```
# Comments and blank lines are allowed.

RCS
.+,v

CVS
\.\#.+       # CVS conflict files / emacs lock files
\.cvsignore

\.svn
_darcs
\.hg

\.git
\.gitignore
\.gitmodules
```

```
.+~          # emacs backup files
\#.*\#       # emacs autosave files

^/README.*
^/LICENSE.*
^/COPYING
```

Stow first iterates through the chosen ignore list (built-in, global, or package-local) as per above, stripping out comments (if you want to include the '`#`' symbol in a regular expression, escape it with a blackslash) and blank lines, placing each regular expressions into one of two sets depending on whether it contains the '`/`' forward slash symbol.

Then in order to determine whether a file or directory should be ignored:

1. Stow calculates its path relative to the top-level package directory, prefixing that with '`/`'. If any of the regular expressions containing a '`/`' *exactly*[1] match a subpath[2] of this relative path, then the file or directory will be ignored.
2. If none of the regular expressions containing a '`/`' match in the manner described above, Stow checks whether the *basename*[3] of the file or directory matches *exactly* against the remaining regular expressions which do not contain a '`/`', and if so, ignores the file or directory.
3. Otherwise, the file or directory is not ignored.

For example, if a file `bazqux` is in the `foo/bar` subdirectory of the package directory, Stow would use '`/foo/bar/bazqux`' as the text for matching against regular expressions which contain '`/`', and '`bazqux`' as the text for matching against regular expressions which don't contain '`/`'. Then regular expressions '`bazqux`', '`baz.*`', '`.*qux`', '`bar/.*x`', and '`^/foo/.*qux`' would all match (causing the file to be ignored), whereas '`bar`', '`baz`', '`qux`', and '`o/bar/b`' would not (although '`bar`' would cause its parent directory to be ignored and prevent Stow from recursing into that anyway, in which case the file `bazqux` would not even be considered for stowing).

As a special exception to the above algorithm, any `.stow-local-ignore` present in the top-level package directory is *always* ignored, regardless of the contents of any ignore list, because this file serves no purpose outside the stow directory.

4.3 Justification For Yet Another Set Of Ignore Files

The reader may note that this format is very similar to existing ignore list file formats, such as those for `cvs`, `git`, `rsync` etc., and wonder if another set of ignore lists is justified. However there are good reasons why Stow does not simply check for the presence of say, `.cvsignore`, and use that if it exists. Firstly, there is no guarantee that a stow package would contain any version control meta-data, or permit introducing this if it didn't already exist.

---

[1] Exact matching means the regular expression is anchored at the beginning and end, in contrast to unanchored regular expressions which will match a substring.

[2] In this context, "subpath" means a contiguous subset of path segments; e.g for the relative path `one/two/three`, there are six valid subpaths: `one`, `two`, `three`, `one/two`, `two/three`, `one/two/three`.

[3] The "basename" is the name of the file or directory itself, excluding any directory path prefix - as returned by the `basename` command.

Secondly even if it did, version control system ignore lists generally reflect *build-time* ignores rather than *install-time*, and there may be some intermediate or temporary files on those ignore lists generated during development or at build-time which it would be inappropriate to stow, even though many files generated at build-time (binaries, libraries, documentation etc.) certainly do need to be stowed. Similarly, if a file is *not* in the version control system's ignore list, there is no way of knowing whether the file is intended for end use, let alone whether the version control system is tracking it or not.

Therefore it seems clear that ignore lists provided by version control systems do not provide sufficient information for Stow to determine which files and directories to stow, and so it makes sense for Stow to support independent ignore lists.

5 Installing Packages

The default action of Stow is to install a package. This means creating symlinks in the target tree that point into the package tree. Stow attempts to do this with as few symlinks as possible; in other words, if Stow can create a single symlink that points to an entire subtree within the package tree, it will choose to do that rather than create a directory in the target tree and populate it with symlinks.

5.1 Tree folding

For example, suppose that no packages have yet been installed in `/usr/local`; it's completely empty (except for the `stow` subdirectory, of course). Now suppose the Perl package is installed. Recall that it includes the following directories in its installation image: `bin`; `info`; `lib/perl`; `man/man1`. Rather than creating the directory `/usr/local/bin` and populating it with symlinks to `../stow/perl/bin/perl` and `../stow/perl/bin/a2p` (and so on), Stow will create a single symlink, `/usr/local/bin`, which points to `stow/perl/bin`. In this way, it still works to refer to `/usr/local/bin/perl` and `/usr/local/bin/a2p`, and fewer symlinks have been created. This is called *tree folding*, since an entire subtree is "folded" into a single symlink.

To complete this example, Stow will also create the symlink `/usr/local/info` pointing to `stow/perl/info`; the symlink `/usr/local/lib` pointing to `stow/perl/lib`; and the symlink `/usr/local/man` pointing to `stow/perl/man`.

Now suppose that instead of installing the Perl package into an empty target tree, the target tree is not empty to begin with. Instead, it contains several files and directories installed under a different system-administration philosophy. In particular, `/usr/local/bin` already exists and is a directory, as are `/usr/local/lib` and `/usr/local/man/man1`. In this case, Stow will descend into `/usr/local/bin` and create symlinks to `../stow/perl/bin/perl` and `../stow/perl/bin/a2p` (etc.), and it will descend into `/usr/local/lib` and create the tree-folding symlink `perl` pointing to `../stow/perl/lib/perl`, and so on. As a rule, Stow only descends as far as necessary into the target tree when it can create a tree-folding symlink. However, this behaviour can be changed via the `--no-folding` option; see Chapter 3 [Invoking Stow], page 5.

5.2 Tree unfolding

The time often comes when a tree-folding symlink has to be undone because another package uses one or more of the folded subdirectories in its installation image. This operation is called *splitting open* or *unfolding* a folded tree. It involves removing the original symlink from the target tree, creating a true directory in its place, and then populating the new directory with symlinks to the newly-installed package *and* to the old package that used the old symlink. For example, suppose that after installing Perl into an empty `/usr/local`, we wish to install Emacs. Emacs's installation image includes a `bin` directory containing the `emacs` and `etags` executables, among others. Stow must make these files appear to be installed in `/usr/local/bin`, but presently `/usr/local/bin` is a symlink to `stow/perl/bin`. Stow therefore takes the following steps: the symlink `/usr/local/bin` is deleted; the directory `/usr/local/bin` is created; links are made from `/usr/local/bin`

to `../stow/emacs/bin/emacs` and `../stow/emacs/bin/etags`; and links are made from `/usr/local/bin` to `../stow/perl/bin/perl` and `../stow/perl/bin/a2p`.

5.3 Ownership

When splitting open a folded tree, Stow makes sure that the symlink it is about to remove points inside a valid package in the current stow directory. *Stow will never delete anything that it doesn't own.* Stow "owns" everything living in the target tree that points into a package in the stow directory. Anything Stow owns, it can recompute if lost: symlinks that point into a package in the stow directory, or directories that only contain symlinks that stow "owns". Note that by this definition, Stow doesn't "own" anything *in* the stow directory or in any of the packages.

5.4 Conflicts during installation

If Stow needs to create a directory or a symlink in the target tree and it cannot because that name is already in use and is not owned by Stow, then a *conflict* has arisen. See Chapter 7 [Conflicts], page 14.

6 Deleting Packages

When the `-D` option is given, the action of Stow is to delete a package from the target tree. Note that Stow will not delete anything it doesn't "own". Deleting a package does *not* mean removing it from the stow directory or discarding the package tree.

To delete a package, Stow recursively scans the target tree, skipping over any directory that is not included in the installation image.[1] For example, if the target directory is `/usr/local` and the installation image for the package being deleted has only a `bin` directory and a `man` directory at the top level, then we only scan `/usr/local/bin` and `/usr/local/man`, and not `/usr/local/lib` or `/usr/local/share`, or for that matter `/usr/local/stow`. Any symlink it finds that points into the package being deleted is removed. Any directory that contained only symlinks to the package being deleted is removed.

6.1 Refolding "foldable" trees.

After removing symlinks and empty subdirectories, any directory that contains only symlinks to a single other package is considered to be a previously "folded" tree that was "split open." Stow will refold the tree by removing the symlinks to the surviving package, removing the directory, then linking the directory back to the surviving package. However, this behaviour can be prevented via the `--no-folding` option; see Chapter 3 [Invoking Stow], page 5.

[1] This approach was introduced in version 2 of GNU Stow. Previously, the whole target tree was scanned and stow directories were explicitly omitted. This became problematic when dealing with very large installations. The only situation where this is useful is if you accidentally delete a directory in the package tree, leaving you with a whole bunch of dangling links. Note that you can enable the old approach with the `-p` option. Alternatively, you can use the `--badlinks` option get stow to search for dangling links in your target tree and remove the offenders manually.

7 Conflicts

If, during installation, a file or symlink exists in the target tree and has the same name as something Stow needs to create, and if the existing name is not a folded tree that can be split open, then a *conflict* has arisen. A conflict also occurs if a directory exists where Stow needs to place a symlink to a non-directory. On the other hand, if the existing name is merely a symlink that already points where Stow needs it to, then no conflict has occurred. (Thus it is harmless to install a package that has already been installed.)

For complex packages, scanning the stow and target trees in tandem, and deciding whether to make directories or links, split-open or fold directories, can actually take a long time (a number of seconds). Moreover, an accurate analysis of potential conflicts requires us to take into account all of these operations.

7.1 Deferred Operation

Since version 2.0, Stow now adopts a two-phase algorithm, first scanning for any potential conflicts before any stowing or unstowing operations are performed. If any conflicts are found, they are displayed and then Stow terminates without making any modifications to the filesystem. This means that there is much less risk of a package being partially stowed or unstowed due to conflicts.

Prior to version 2.0, if a conflict was discovered, the stow or unstow operation could be aborted mid-flow, leaving the target tree in an inconsistent state.

8 Mixing Operations

Since version 2.0, multiple distinct actions can be specified in a single invocation of GNU Stow. For example, to update an installation of Emacs from version 21.3 to 21.4a you can now do the following:

```
stow -D emacs-21.3 -S emacs-21.4a
```

which will replace emacs-21.3 with emacs-21.4a using a single invocation.

This is much faster and cleaner than performing two separate invocations of stow, because redundant folding/unfolding operations can be factored out. In addition, all the operations are calculated and merged before being executed (see [Deferred Operation], page 14), so the amount of time in which GNU Emacs is unavailable is minimised.

You can mix and match any number of actions, for example,

```
stow -S pkg1 pkg2 -D pkg3 pkg4 -S pkg5 -R pkg6
```

will unstow pkg3, pkg4 and pkg6, then stow pkg1, pkg2, pkg5 and pkg6.

9 Multiple Stow Directories

If there are two or more system administrators who wish to maintain software separately, or if there is any other reason to want two or more stow directories, it can be done by creating a file named `.stow` in each stow directory. The presence of `/usr/local/foo/.stow` informs Stow that, though `foo` is not the current stow directory, even if it is a subdirectory of the target directory, nevertheless it is *a* stow directory and as such Stow doesn't "own" anything in it (see Chapter 5 [Installing Packages], page 11). This will protect the contents of `foo` from a '`stow -D`', for instance.

When multiple stow directories share a target tree, if a tree-folding symlink is encountered and needs to be split open during an installation, as long as the top-level stow directory into which the existing symlink points contains `.stow`, Stow knows how to split open the tree in the correct manner.

10 Target Maintenance

From time to time you will need to clean up your target tree. Since version 2, Stow provides a new utility `chkstow` to help with this. It includes three operational modes which performs checks that would generally be too expensive to be performed during normal stow execution.

The syntax of the `chkstow` command is:

```
chkstow [options]
```

The following options are supported:

'`-t dir`'
'`--target=dir`'
: Set the target directory to *dir* instead of the parent of the stow directory. Defaults to the parent of the stow directory, so it is typical to execute `stow` from the directory `/usr/local/stow`.

'`-b`'
'`--badlinks`'
: Checks target directory for bogus symbolic links. That is, links that point to non-existent files.

'`-a`'
'`--aliens`'
: Checks for files in the target directory that are not symbolic links. The target directory should be managed by stow alone, except for directories that contain a `.stow` file.

'`-l`'
'`--list`'
: Will display the target package for every symbolic link in the stow target directory.

11 Resource Files

Default command line options may be set in `.stowrc` (current directory) or `~/.stowrc` (home directory). These are parsed in that order, and are appended together if they both exist. The effect of the options in the resource file is similar to simply prepending the options to the command line. This feature can be used for some interesting effects.

For example, suppose your site uses more than one stow directory, perhaps in order to share around responsibilities with a number of systems administrators. One of the administrators might have the following in their `~/.stowrc` file:

```
--dir=/usr/local/stow2
--target=/usr/local
--ignore='~'
--ignore='^CVS'
```

so that the `stow` command will default to operating on the `/usr/local/stow2` directory, with `/usr/local` as the target, and ignoring vi backup files and CVS directories.

If you had a stow directory `/usr/local/stow/perl-extras` that was only used for Perl modules, then you might place the following in `/usr/local/stow/perl-extras/.stowrc`:

```
--dir=/usr/local/stow/perl-extras
--target=/usr/local
--override=bin
--override=man
--ignore='perllocal\.pod'
--ignore='\.packlist'
--ignore='\.bs'
```

so that when you are in the `/usr/local/stow/perl-extras` directory, `stow` will regard any subdirectories as stow packages, with `/usr/local` as the target (rather than the immediate parent directory `/usr/local/stow`), overriding any pre-existing links to bin files or man pages, and ignoring some cruft that gets installed by default.

If an option is provided both on the command line and in a resource file, the command line option takes precedence. For options that provide a single value, such as `--target` or `--dir`, the command line option will overwrite any options in the resource file. For options that can be given more than once, `--ignore` for example, command line options and resource options are appended together.

For options that take a file path, environment variables and the tilde character (`~`) are expanded. An environment variable can be given in either the `$VAR` or `${VAR}` form. To prevent expansion, escape the `$` or `~` with a backslash.

The options `-D`, `-S`, and `-R` are ignored in resource files. This is also true of any package names given in the resource file.

12 Compile-time vs. Install-time

Software whose installation is managed with Stow needs to be installed in one place (the package directory, e.g. `/usr/local/stow/perl`) but needs to appear to run in another place (the target tree, e.g., `/usr/local`). Why is this important? What's wrong with Perl, for instance, looking for its files in `/usr/local/stow/perl` instead of in `/usr/local`?

The answer is that there may be another package, e.g., `/usr/local/stow/perl-extras`, stowed under `/usr/local`. If Perl is configured to find its files in `/usr/local/stow/perl`, it will never find the extra files in the '`perl-extras`' package, even though they're intended to be found by Perl. On the other hand, if Perl looks for its files in `/usr/local`, then it will find the intermingled Perl and '`perl-extras`' files.

This means that when you compile a package, you must tell it the location of the run-time, or target tree; but when you install it, you must place it in the stow tree.

12.1 Advice on changing compilation and installation parameters

Some software packages allow you to specify, at compile-time, separate locations for installation and for run-time. Perl is one such package; see Section 12.5 [Perl and Perl 5 Modules], page 21. Others allow you to compile the package, then give a different destination in the '`make install`' step without causing the binaries or other files to get rebuilt. Most GNU software falls into this category; Emacs is a notable exception. See Section 12.2 [GNU Emacs], page 20, and Section 12.3 [Other FSF Software], page 20.

Still other software packages cannot abide the idea of separate installation and run-time locations at all. If you try to '`make install prefix=/usr/local/stow/`*`foo`*', then first the whole package will be recompiled to hardwire the `/usr/local/stow/`*`foo`* path. With these packages, it is best to compile normally, then run '`make -n install`', which should report all the steps needed to install the just-built software. Place this output into a file, edit the commands in the file to remove recompilation steps and to reflect the Stow-based installation location, and execute the edited file as a shell script in place of '`make install`'. Be sure to execute the script using the same shell that '`make install`' would have used.

(If you use GNU Make and a shell [such as GNU bash] that understands `pushd` and `popd`, you can do the following:

1. Replace all lines matching '`make[`*`n`*`]: Entering directory `*`dir`*'` with '`pushd `*`dir`*'.
2. Replace all lines matching '`make[`*`n`*`]: Leaving directory `*`dir`*'` with '`popd`'.
3. Delete all lines matching '`make[`*`n`*`]: Nothing to be done for `*`rule`*'.

Then find other lines in the output containing `cd` or `make` commands and rewrite or delete them. In particular, you should be able to delete sections of the script that resemble this:

```
for i in dir_1 dir_2 ...; do \
  (cd $i; make args ...) \
done
```

Note, that's "should be able to," not "can." Be sure to modulate these guidelines with plenty of your own intelligence.

The details of stowing some specific packages are described in the following sections.

12.2 GNU Emacs

Although the Free Software Foundation has many enlightened practices regarding Makefiles and software installation (see see Section 12.3 [Other FSF Software], page 20), Emacs, its flagship program, doesn't quite follow the rules. In particular, most GNU software allows you to write:

```
make
make install prefix=/usr/local/stow/package
```

If you try this with Emacs, then the new value for *prefix* in the '`make install`' step will cause some files to get recompiled with the new value of *prefix* wired into them. In Emacs 19.23 and later,[1] the way to work around this problem is:

```
make
make install-arch-dep install-arch-indep prefix=/usr/local/stow/emacs
```

In 19.22 and some prior versions of Emacs, the workaround was:

```
make
make do-install prefix=/usr/local/stow/emacs
```

12.3 Other FSF Software

The Free Software Foundation, the organization behind the GNU project, has been unifying the build procedure for its tools for some time. Thanks to its tools '`autoconf`' and '`automake`', most packages now respond well to these simple steps, with no other intervention necessary:

```
./configure options
make
make install prefix=/usr/local/stow/package
```

Hopefully, these tools can evolve to be aware of Stow-managed packages, such that providing an option to '`configure`' can allow '`make`' and '`make install`' steps to work correctly without needing to "fool" the build process.

12.4 Cygnus Software

Cygnus is a commercial supplier and supporter of GNU software. It has also written several of its own packages, released under the terms of the GNU General Public License; and it has taken over the maintenance of other packages. Among the packages released by Cygnus are '`gdb`', '`gnats`', and '`dejagnu`'.

Cygnus packages have the peculiarity that each one unpacks into a directory tree with a generic top-level Makefile, which is set up to compile *all* of Cygnus' packages, any number of which may reside under the top-level directory. In other words, even if you're only building '`gnats`', the top-level Makefile will look for, and try to build, `gdb` and `dejagnu` subdirectories, among many others.

The result is that if you try '`make -n install prefix=/usr/local/stow/package`' at the top level of a Cygnus package, you'll get a bewildering amount of output. It will then be very difficult to visually scan the output to see whether the install will proceed correctly. Unfortunately, it's not always clear how to invoke an install from the subdirectory of interest.

[1] As I write this, the current version of Emacs is 19.31.

In cases like this, the best approach is to run your '`make install prefix=...`', but be ready to interrupt it if you detect that it is recompiling files. Usually it will work just fine; otherwise, install manually.

12.5 Perl and Perl 5 Modules

Perl 4.036 allows you to specify different locations for installation and for run-time. It is the only widely-used package in this author's experience that allows this, though hopefully more packages will adopt this model.

Unfortunately, the authors of Perl believed that only AFS sites need this ability. The configuration instructions for Perl 4 misleadingly state that some occult means are used under AFS to transport files from their installation tree to their run-time tree. In fact, that confusion arises from the fact that Depot, Stow's predecessor, originated at Carnegie Mellon University, which was also the birthplace of AFS. CMU's need to separate install-time and run-time trees stemmed from its use of Depot, not from AFS.

The result of this confusion is that Perl 5's configuration script doesn't even offer the option of separating install-time and run-time trees *unless* you're running AFS. Fortunately, after you've entered all the configuration settings, Perl's setup script gives you the opportunity to edit those settings in a file called `config.sh`. When prompted, you should edit this file and replace occurrences of

```
inst.../usr/local...
```

with

```
inst.../usr/local/stow/perl...
```

You can do this with the following Unix command:

```
sed 's,^\(inst.*/usr/local\),\1/stow/perl,' config.sh > config.sh.new
mv config.sh.new config.sh
```

Hopefully, the Perl authors will correct this deficiency in Perl 5's configuration mechanism.

Perl 5 modules—i.e., extensions to Perl 5—generally conform to a set of standards for building and installing them. The standard says that the package comes with a top-level `Makefile.PL`, which is a Perl script. When it runs, it generates a `Makefile`.

If you followed the instructions above for editing `config.sh` when Perl was built, then when you create a `Makefile` from a `Makefile.PL`, it will contain separate locations for run-time (`/usr/local`) and install-time (`/usr/local/stow/perl`). Thus you can do

```
perl Makefile.PL
make
make install
```

and the files will be installed into `/usr/local/stow/perl`. However, you might prefer each Perl module to be stowed separately. In that case, you must edit the resulting Makefile, replacing `/usr/local/stow/perl` with `/usr/local/stow/`*`module`*. The best way to do this is:

```
perl Makefile.PL
find . -name Makefile -print | \
  xargs perl -pi~ -e 's,^(INST.*/stow)/perl,$1/module,;'
```

```
make
make install
```

(The use of '`find`' and '`xargs`' ensures that all Makefiles in the module's source tree, even those in subdirectories, get edited.) A good convention to follow is to name the stow directory for a Perl *module* `cpan.`*`module`*, where '`cpan`' stands for Comprehensive Perl Archive Network, a collection of FTP sites that is the source of most Perl 5 extensions. This way, it's easy to tell at a glance which of the subdirectories of `/usr/local/stow` are Perl 5 extensions.

When you stow separate Perl 5 modules separately, you are likely to encounter conflicts (see Chapter 7 [Conflicts], page 14) with files named `.exists` and `perllocal.pod`. One way to work around this is to remove those files before stowing the module. If you use the `cpan.`*`module`* naming convention, you can simply do this:

```
cd /usr/local/stow
find cpan.* \( -name .exists -o -name perllocal.pod \) -print | \
  xargs rm
```

13 Bootstrapping

Suppose you have a stow directory all set up and ready to go: `/usr/local/stow/perl` contains the Perl installation, `/usr/local/stow/stow` contains Stow itself, and perhaps you have other packages waiting to be stowed. You'd like to be able to do this:

```
cd /usr/local/stow
stow -vv *
```

but `stow` is not yet in your `PATH`. Nor can you do this:

```
cd /usr/local/stow
stow/bin/stow -vv *
```

because the '`#!`' line at the beginning of `stow` tries to locate Perl (usually in `/usr/local/bin/perl`), and that won't be found. The solution you must use is:

```
cd /usr/local/stow
perl/bin/perl stow/bin/stow -vv *
```

14 Reporting Bugs

You can report bugs to the current maintainers in one of three ways:

1. Send e-mail to `bug-stow@gnu.org`.
2. File an issue in the Savannah bug tracker (`https://savannah.gnu.org/bugs/?group=stow`).
3. File an issue in the GitHub project (`https://github.com/aspiers/stow/issues/`).

While GitHub is arguably the most convenient of these three options, it is not the most ethical or freedom-preserving way to host software projects (`https://www.gnu.org/software/repo-criteria-evaluation.html#GitHub`). Therefore the GitHub project may be moved to a more ethical hosting service (`https://github.com/aspiers/stow/issues/43`) in the future.

Before reporting a bug, it is recommended to check whether it is already known, so please first see Chapter 15 [Known Bugs], page 25.

When reporting a new bug, please include:

- the version number of Stow (‘`stow --version`’);
- the version number of Perl (‘`perl -v`’);
- the system information, which can often be obtained with ‘`uname -a`’;
- a description of the bug;
- the precise command you gave;
- the output from the command (preferably verbose output, obtained by adding ‘`--verbose=5`’ to the Stow command line).

If you are really keen, consider developing a minimal test case and creating a new test. See the `t/` directory in the source for lots of examples, and the `CONTRIBUTING.md` file for a guide on how to contribute.

Before reporting a bug, please read the manual carefully, especially Chapter 15 [Known Bugs], page 25, to see whether you’re encountering something that doesn’t need reporting. (see Chapter 7 [Conflicts], page 14).

15 Known Bugs

Known bugs can be found in the following locations:

- the GitHub issue tracker (`https://github.com/aspiers/stow/issues/`)
- the Savannah bug tracker (`https://savannah.gnu.org/bugs/?group=stow`)
- the bug-stow list archives (`https://lists.gnu.org/archive/html/bug-stow/`)

If you think you have found a new bug, please see Chapter 14 [Reporting Bugs], page 24.

GNU General Public License

Version 3, 29 June 2007

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program—to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

   "This License" refers to version 3 of the GNU General Public License.

   "Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

   "The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

   To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

   A "covered work" means either the unmodified Program or a work based on the Program.

   To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

   To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

   An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

   The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

   A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

   All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

   You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

   Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

   No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

   You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

   You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

   You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

   a. The work must carry prominent notices stating that you modified it, and giving a relevant date.

   b. The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".

   c. You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

   d. If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

   A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

   You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a. Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

b. Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c. Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d. Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e. Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source.

The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

   a. Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or
   b. Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or
   c. Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

d. Limiting the use for publicity purposes of names of licensors or authors of the material; or

e. Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

f. Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

   You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

   However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

   Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

   Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

   You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance.

However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so

available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

    If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

    Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

    THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

    IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

    If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
one line to give the program's name and a brief idea of what it does.
Copyright (C) year name of author

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or (at
your option) any later version.

This program is distributed in the hope that it will be useful, but
WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see https://www.gnu.org/licenses/.
```

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

```
program Copyright (C) year name of author
This program comes with ABSOLUTELY NO WARRANTY; for details type ‘show w’.
This is free software, and you are welcome to redistribute it
under certain conditions; type ‘show c’ for details.
```

The hypothetical commands ‘`show w`’ and ‘`show c`’ should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an "about box".

You should also get your employer (if you work as a programmer) or school, if any, to sign a "copyright disclaimer" for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see `https://www.gnu.org/licenses/`.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read `https://www.gnu.org/licenses/why-not-lgpl.html`.

Index