Stow 2.1.0

Managing the installation of software packages

Bob Glickstein, Zanshin Software, Inc.
Kahlil Hodgson, RMIT Univerity, Australia.

This manual describes GNU Stow version 2.1.0, a program for managing the installation of software packages.

Software and documentation is copyrighted by the following:

© 1993, 1994, 1995, 1996 Bob Glickstein <bobg+stow@zanshin.com>
© 2000, 2001 Guillaume Morin <gmorin@gnu.org>
© 2007 Kahlil (Kal) Hodgson <kahlil@internode.on.net>
© 2011 Adam Spiers <stow@adamspiers.org>

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided also that the section entitled "GNU General Public License" is included with the modified manual, and provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this manual into another language, under the above conditions for modified versions, except that this permission notice may be stated in a translation approved by the Free Software Foundation.

1 Introduction

Stow is a tool for managing the installation of multiple software packages in the same run-time directory tree. One historical difficulty of this task has been the need to administer, upgrade, install, and remove files in independent packages without confusing them with other files sharing the same file system space. For instance, it is common to install Perl and Emacs 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.

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 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`').

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

---

[1] As of Perl 4.036 and Emacs 19.22.

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. A common, but by no means the only such location is ‘`/usr/local`’. 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.

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. A symlink can be *relative* or *absolute*. An absolute symlink names a full path; that is, one starting from ‘`/`’. A relative symlink names a relative path; that is, one not starting from ‘`/`’. The target of a relative symlink is computed starting from the symlink’s own directory. Stow only creates relative symlinks.

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='<regex>'`’

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 7, for more details.

‘`--defer='<regex>'`’

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 13.)

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 regex.

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=’<regex>’`’

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.

‘`-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 0, 1, 2, and 3; 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. Scanning the whole tree can be prohibitive if your target tree is very large. 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 16).

‘`-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`’

‘`--stow`’ 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 14, 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

.+~          # emacs backup files
```

```
    \#.*\#          # emacs autosave files
```

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`, and `qux` 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.

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

---

[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/four`', the following are examples of valid subpaths: '`one`', '`two`', '`two/three`', '`two/three/four`'.

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

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.

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.

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`'.

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.

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 13.

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/bin/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. Any directory that, after removing symlinks and empty subdirectories, contains only symlinks to a single other package, is considered to be a previously "folded" tree that was "split open." Stow will re-fold the tree by removing the symlinks to the surviving package, removing the directory, then linking the directory back to the surviving package.

---

[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 13), so the amount of 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 10). 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 effectively prepended to you 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 there '~/.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 the when your 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 directoy '`/usr/local/stow`'), overriding any pre-existing links to bin files or man pages, and ignoring some cruft that gets installed by default.

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.

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 12.4 [Perl and Perl 5 Modules], page 20. 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 12.1 [GNU Emacs], page 19, and Section 12.2 [Other FSF Software], page 19.

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.

12.1 GNU Emacs

Although the Free Software Foundation has many enlightened practices regarding Makefiles and software installation (see see Section 12.2 [Other FSF Software], page 19), 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
```

12.2 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.

12.3 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.

12.4 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 13) 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

Please send bug reports to the current maintainers by electronic mail. The address to use is ‘`<bug-stow@gnu.org>`’. 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=3`’ to the Stow command line).

If you are really keen, consider developing a minimal test case and creating a new test. See the ‘`t/`’ for lots of examples.

Before reporting a bug, please read the manual carefully, especially Chapter 15 [Known Bugs], page 24, to see whether you’re encountering something that doesn’t need reporting. (see Chapter 7 [Conflicts], page 13).

15 Known Bugs

- When using multiple stow directories (see Chapter 9 [Multiple Stow Directories], page 15), Stow fails to "split open" tree-folding symlinks (see Chapter 5 [Installing Packages], page 10) that point into a stow directory which is not the one in use by the current Stow command. Before failing, it should search the target of the link to see whether any element of the path contains a '`.stow`' file. If it finds one, it can "learn" about the cooperating stow directory to short-circuit the '`.stow`' search the next time it encounters a tree-folding symlink.

GNU General Public License

Version 2, June 1991

Copyright © 1989, 1991 Free Software Foundation, Inc.
675 Mass Ave, Cambridge, MA 02139, USA

Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) 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 this service 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 make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. 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.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

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

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute 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 and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   a. You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

   b. You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

   c. If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:
   a. Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
   b. Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
   c. Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

   The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

   If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.
7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), 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 distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

   If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

   It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

   This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.
8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.
9. The Free Software Foundation may publish revised and/or new versions of the 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 a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.
10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software

which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, 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.
12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE 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.

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 convey 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 an idea of what it does.
Copyright (C) 19yy  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 2
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, write to the Free Software
Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
```

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

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

```
Gnomovision version 69, Copyright (C) 19yy name of author
Gnomovision 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, the commands you use may be called something other than '`show w`' and '`show c`'; they could even be mouse-clicks or menu items—whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright
interest in the program `Gnomovision'
(which makes passes at compilers) written
by James Hacker.

signature of Ty Coon, 1 April 1989
Ty Coon, President of Vice
```

This 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 Library General Public License instead of this License.

Index

A

absolute symlink 3

C

conflict 11

D

deletion 12
directory folding 10

F

folding trees 10

I

ignore lists 7
ignoring files and directories 7
installation 10
installation image 3

O

ownership 10

P

package 3
package directory 3
package name 3

R

relative symlink 3

S

splitting open folded trees 10
stow directory 3
symlink 3

T

target directory 3
tree folding 10

U

unfolding trees 10

Table of Contents

. . . . 1

1 Introduction 2

2 Terminology 3

3 Invoking Stow 4

4 Ignore Lists 7
  4.1 Motivation For Ignore Lists 7
  4.2 Types And Syntax Of Ignore Lists 7
  4.3 Justification For Yet Another Set Of Ignore Files 8

5 Installing Packages 10

6 Deleting Packages 12

7 Conflicts 13
  7.1 Deferred Operation 13

8 Mixing Operations 14

9 Multiple Stow Directories 15

10 Target Maintenance 16

11 Resource Files 17

12 Compile-time vs Install-time 18
  12.1 GNU Emacs 19
  12.2 Other FSF Software 19
  12.3 Cygnus Software 19
  12.4 Perl and Perl 5 Modules 20

13 Bootstrapping 22

14 Reporting Bugs 23

15 Known Bugs 24

GNU General Public License 25

Preamble 25

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION 26

How to Apply These Terms to Your New Programs 30

Index 31