Graft - a package management utility

Prepared by Peter Samuel

$Revision: 2.6 $

$Date: 2015/07/11 16:09:43 $

> ***graft:*** *To insert (a graft) in a branch or stem of another tree; to propagate by insertion in another stock; also, to insert a graft upon. To implant a portion of (living flesh or skin) in a lesion so as to form an organic union. To join (one thing) to another as if by grafting, so as to bring about a close union.*

Contents

- Introduction
- Rationale
- Research
- Design
- History
- Installation
- Grafting *Graft* and *Perl* - the bootstrap problem
- Using *Graft*
    - Compiling Packages
    - *Graft* command line options
        1. Install
        2. Delete
        3. Prune
    - Testing the *Graft* Installation
    - Installing Packages
    - Bypassing package directories
    - Including specific files and/or directories
    - Excluding specific files and/or directories
    - Grafting part of a package
    - Deleting and/or Upgrading Packages
    - Transitioning a package to *Graft* control
    - Conflict Processing
    - Exit Status
- Using *Graft* with other package management tools
- Availability
- License

Introduction

Graft provides a mechanism for managing multiple packages under a single directory hierarchy. It was inspired by both *Depot* (Carnegie Mellon University) and *Stow* (Bob Glickstein).

For the purposes of this discussion a *package* is defined as a suite of programs and files that make up an individual product. For example, the *package* known as *gcc* consists of the compiler and preprocessor programs, include files, manual pages and any other associated file or program. The concept of a *package* should not be confused with some vendor's definitions that are - by this definition - actually collections of *packages*.

Special thanks to Gordon Rowell, Charles Butcher, Charlie Brady and Robert Maldon for design suggestions and contributions.

Rationale

In any reasonably large environment, many software packages will be installed. The installation location for these packages usually follows one of three rationales - each with its own advantages and drawbacks:

1. Each package is isolated from all other packages by installing it into a self contained directory tree. All binaries, manual pages, library and configuration files are stored under a single directory tree. This directory tree contains NO other files which are not the exclusive domain of the package in question.

   This method makes package demarcation obvious. As each package is self contained, identification of any file within a package is immediately apparent.

   Multiple versions of packages can be installed fairly easily to accommodate acceptance testing of new versions and/or legacy systems.

   However, the use of individual package directories can lead to VERY long **$PATH** and **$MANPATH** environment variables. Some shells may not be able to handle such long variables. Whenever a new package is added, each user MUST update their **$PATH** and **$MANPATH** to make the package available.

2. Packages are installed under a common directory tree. Binaries for all packages are grouped in a single directory, manual pages for all packages in another directory and so on.

   This method eliminates the need for continually updating long **$PATH** variables for each user. As soon as a package is placed into the common `'bin'` directory it is immediately available to all users (after a shell rehash if necessary).

   However, when a package is to be updated it is often very difficult to isolate all

the files related to a particular package if they are intermingled with unrelated files.

3. A combination of methods (1) and (2).

In an effort to maximise the advantages and minimise the disadvantages, *Depot*, *Stow* and *Graft* adopt a similar philosophy:

> Packages are installed in self contained directory trees and symbolic links from a common area are made to the package files.

This approach allows multiple versions of the same package to co-exist on the one system. One version is the commonly available version and symbolic links will be made to this version. New versions can be tested and once acceptable can replace the current commonly available version. Older versions can still be used for legacy systems by using the '*real*' path name to the package instead of the '*common*' path name.

The size and complexity of environment variables such as **$PATH** and **$MANPATH** is minimised because only the common area is required. Any special cases can also be accommodated but these will usually be in the minority when compared with the number of commonly available packages.

Research

Note: Development of *Graft* began in late 1996. The comments regarding the packages listed below reflect their functionality and behaviour at that time and may not necessarily reflect their current functionality and behaviour.

As stated earlier, *Graft* was inspired by *Depot* and *Stow*. Both these systems were examined and finally rejected for the following reasons:

Depot

> *Depot* is very flexible yet cumbersome.
>
> It requires a database file to be created which provides a snapshot of the current state of both the package repository and the *Depot* target. It is possible to inadvertently destroy the package repository if the database is damaged.
>
> *Depot* assumes "*ownership*" of the target area, making it almost impossible to accommodate packages that are not under the control of *Depot*. ("*Ownership*" in this case means that *Depot* assumes ALL files in the target area will be under the control of *Depot*. It does not imply that *Depot* modifies Unix file permissions).
>
> Because of *Depot*'s assumed *ownership* it is difficult for other packages not under the control of *Depot* to be placed in the same target area.
>
> *Depot* attempts to impose a fixed package repository relative to the package

target. It assumes that all packages will be stored under '*dir*/depot/*package*' and the target will be '*dir*'. This can be overridden on the command line but the internals of *Depot* make this mechanism cumbersome.

Depot is written in C and there are many source files in its distribution. Local modifications would be difficult to quickly implement and test.

Stow

Stow is a stateless system. It requires no database or configuration information.

Like *Depot*, it assumes that the package repository will be stored under '*dir*/stow/*package*' and the target will be '*dir*'. This can be overridden on the command line and works well during the install phase.

Stow assumes "*ownership*" of the target area, making it difficult to accommodate packages that are not under the control of *Stow*. ("*Ownership*" in this case means that *Stow* assumes ALL files in the target area will be under the control of *Stow*. It does not imply that *Stow* modifies Unix file permissions).

Because of *Stow*'s assumed *ownership* it is difficult for other packages not under the control of *Stow* to be placed in the same target area. When deleting packages, *Stow* examines everything in the target directory - whether it is associated with the package it is trying to delete or not. This can be time consuming and potentially dangerous as empty directories are also removed - even empty directories that do not belong to the package being removed.

Stow has a clever feature of *folding* and *unfolding* directories. It attempts to optimise the number of symbolic links by making links to directories if the directory is only associated with a single package. If at a later date *Stow* discovers another package that needs that directory it will *unfold* that directory into a collection of symbolic links to files rather than a single symbolic link to the directory. *Stow* will *fold* the directory when removing packages if the remainder of the directory is only concerned with a single package. While clever, this feature is probably a waste of time and effort. It means that the entire package target must be scanned to determine package ownership of links and as packages will usually be replaced by newer versions a directory *fold* will probably be short lived.

Stow will sometimes miss potential conflicts when run in *show only* mode. The conflicts may occur when a directory is unfolded and will not show up in *show only* mode.

Stow's author suggests that packages be compiled such that they refer to files in the target location rather than the actual package installation directory. This approach precludes the use of multiple versions of packages with different configuration and/or library files.

Stow is written in *Perl* and is only a few hundred lines of code so local modifications can be accommodated. However there are very few comments in

the code which makes the process of modification difficult.

Since the release of *Graft* 1.6, the existence of yet another packaging program has been brought to the author's attention.

Encap

> *Encap* grew out of work begun at the University of Illinois Champaign-Urbana. It has the same underlying philosophy as *Depot*, *Stow* and *Graft* - encapsulate packages into self contained directories and use symbolic links to make them visible in a common location.
>
> *Encap* uses a combination of a *csh* wrapper and a *Perl* program to accomplish its work. Like both *Depot* and *Stow*, *Encap* assumes that all compiled packages will live under a single directory hierarchy - by default '*dir*/encap/*package*'. It then attempts to create a symbolic link tree for ALL the packages under this area. There doesn't appear to be any easy way to support the quick addition or removal of a single package.
>
> A new release of *Encap* incorporating many new features was expected to be available in early 1997, however no release greater than version 1.2 has been forthcoming.
>
> One good feature of *Encap* is the ability to exclude specific files from the package tree. This concept has been incorporated into *Graft* 1.7 and above.

Since the release of *Graft* 2.3, the existence of several another packaging programs have been brought to the author's attention. Rather than outline their features and whether or not the author feels they are superior (or inferior) to *Graft*, a reference to each package and a brief description is given and further research is left as an exercise for the reader:

stowES

> http://freshmeat.net/projects/stowes/
>
> "*stowES (stow Enhancement Script) is a Perl script which tries to ease the use of the "stow" packaging program and software which can be compiled and installed with autoconf. It automates the compilation and installation of software packages and provides some useful functions to maintain your stow packages (e.g., list packages, check packages for integrity, etc.).*"

opt_depot

> http://www.arlut.utexas.edu/csd/opt_depot/
>
> "*opt_depot is a suite of Perl scripts which makes it easy to manage installed software across a wide range of client systems. opt_depot makes it possible to keep all files associated with a program together in one directory, so installation and de-installation is simple. opt_depot is easy to manage, and provides a scheme for installing software in a truly portable fashion; packages may be*

installed locally on client systems, or kept in a central package archive for NFS access. "

This site also has links to several other package management utilities, including *Graft*.

relink

http://sourceforge.net/projects/relink/

"*relink is a package management tool for organization and management of software packages. It should run on any UNIX platform that runs PERL. Similar tools include: rpm(REDHAT/Mandrake), pkgadd(Slackware/SUN), stow(GNU) and depot(CMU)*"

univSrcPkg

http://www.sistema.it/univSrcPkg/

Bud Bruegger has written a brief paper outlining his thoughts on a "*Universal Source Package*" solution.

This site also has links to other package management programs and similar items of interest. **Note:** Some of the links on this site are broken.

Design

This brings us to *Graft*. *Graft* has been designed to use the best features of *Depot*, *Stow* and *Encap* while maintaining as simple a mechanism as possible. The principles of *Graft* are:

- *Graft* will allow packages to be *grafted* from any directory to any other directory. Default `installation` and `target` directories will be used but can easily be overridden on the command line.
- *Graft* will log its actions to a log file. The log file can be specified on the command line. If not specified on the command line a default log file will be used.
- *Graft* will NOT create symbolic links to directories. If a directory does not exist in the target tree it will be created (with the same ownership and permissions as the original if desired).
- *Graft* will create symbolic links with full pathnames rather than relative pathnames. This allows easy identification of true package locations and facilitates block movement of a target tree without the need for *Graft* deletion and re-installation.
- *Graft* will cease installation of a package if a conflict arises. A conflict is defined as one of the following conditions:

    - If the package object is a directory and the target object exists but is not a directory.
    - If the package object is not a directory and the target object exists and is not a symbolic link.
    - If the package object is not a directory and the target object exists and is a symbolic link to something other than the package object.

- Installation conflicts will always be reported. Conflicts will be reported to standard error.

- *Graft* will attempt to display all possible operations when asked, even when asked not to perform the operations.

- *Graft* will not delete directories when uninstalling. *Graft* will print an appropriate message if an empty directory results and leave the deletion for the operator to perform outside the scope of *Graft*'s operations. This ensures that *place holder* directories that may be used by other packages are not inadvertently removed. This feature can be permanently disabled by setting a flag in the `Makefile`. It can also be temporarily disabled using a command line option.

- *Graft* will continue to delete the remainder of a package after a conflict arises. This maximises the amount of deletion that can be performed.

- Deletion conflicts will always be reported. Conflicts will be reported to standard error.

- *Graft* will only concern itself with files relating to the package at hand. This will allow other packages to be placed in the target area without fear of intervention by *Graft*.

- *Graft* will only allow the superuser to install or delete packages. This feature can be permanently disabled by setting a flag in the `Makefile` or it may be overridden by a command line option.

- If the file `.nograft` exists in any package directory, *Graft* will bypass that directory and any subdirectories during installation. The name of this file is specified in the `Makefile`.

- When installing a directory tree, if the file `.graft-exclude` exists in any package directory, *Graft* will assume that the file contains a list of file and/or directory names - one per line - which correspond to files and/or directories in the directory containing the `.graft-exclude` file. These files and/or directories will NOT be *grafted*. The name of this file is specified in the `Makefile`.

  The `.nograft` file takes priority over the `.graft-exclude` file.

- When installing a directory tree, if the file `.graft-include` exists in any package directory, *Graft* will assume that the file contains a list of file and/or directory names - one per line - which correspond to files and/or directories in the directory containing the `.graft-include` file. ONLY the files and/or directories listed

in the `.graft-include` will be *grafted*. The name of this file is specified in the `Makefile`.

The `.graft-exclude` file takes priority over the `.graft-include` file.

- If the file `.nograft` exists in any package directory, it will be ignored and *Graft* will continue processing the directory and any subdirectories during deletion.
- If the file `.graft-exclude` exists in any package directory, its contents will be ignored and *Graft* will continue processing the directory and any subdirectories during deletion.
- If the file `.graft-include` exists in any package directory, its contents will be ignored and *Graft* will continue processing the directory and any subdirectories during deletion.
- As an aid to transitioning systems to *Graft*, *Graft* will allow conflicting files to be pruned. This pruning can take the form of a file rename or a file removal depending on either a `Makefile` flag or a command line option. If file removal is selected and the file is a non-empty directory, it will be renamed instead.
- If the file `.nograft` exists in any package directory, it will be ignored and *Graft* will continue processing the directory and any subdirectories during pruning.
- If the file `.graft-exclude` exists in any package directory, its contents will be ignored and *Graft* will continue processing the directory and any subdirectories during pruning.
- If the file `.graft-include` exists in any package directory, its contents will be ignored and *Graft* will continue processing the directory and any subdirectories during pruning.

History

Development on *Graft* began in October 1996. The initial design used a configuration file to map the installed location of each package to its target directory (that is the directory in which the symbolic links would be created). Work proceeded at a regular pace and by November 1997 *Graft* version *2.1* was released. In this, and all subsequent versions, the configuration file had been removed in favour of using default source and target directories.

No further work was performed until September 2000 when the concept of bypassing or including files and directories using `.nograft` or `.graft-include` files was introduced in *Graft* version *2.3*.

Again nothing changed until February 2002 when Rod Whitby identified a bug in the handling of `.graft-include` files. Several other users (Peter Bray, Robert Maldon and others) also reported some deprecation warnings when using *Graft* with *Perl* version

5.6.0. *Graft* version *2.4* was the end of *Graft* development for over a decade.

In May 2015 Matias A. Fonzo contacted the author wishing to use *Graft* in the Dragora Linux distribution. Matias' usage of *Graft* lead to *Graft* version *2.5* in June 2015 whereby the `-P` command line option was silently ignored if the effective user was not root.

Since the release of *Graft* version *2.4* the author's *Perl* code had improved somewhat so *Graft* version *2.6* released in July 2015 represented a major clean up of coding style and internals. No new behaviours or features were added to the *2.6* release.

Installation

Before installing *Graft* you'll need *Perl* 5.*x*. *Graft* version 2.*x* requires features only available with *Perl* 5.*x* and will not run with *Perl* 4.*x*.

Your operating system and its file system(s) should also support symbolic links. If you can't make symbolic links then you can't use *Graft*! *Graft* will exit gracefully if your version of *Perl* does not support symbolic links. It will also exit gracefully if you attempt to *graft* a package into a file system that does not support symbolic links - from a Linux `ext4` file system into an `vfat` file system for example.

Follow these instructions to install *Graft*:

1. Unpack the gzipped *Graft* distribution:

   ```
   gunzip -c graft-2.6.tar.gz | tar xvf -
   ```

2. change directories to the *Graft* distribution directory:

   ```
   cd graft-2.6
   ```

3. Create an writable version of the `Makefile` by running the command

   ```
   make -f Makefile.dist
   ```

   You'll see output similar to

   ```
   cp Makefile.dist Makefile
   chmod 644 Makefile

   ########################################################
   #                                                      #
   #       You'll now need to modify the Makefile       #
   #      variables to suit your local conditions.      #
   #                                                      #
   ########################################################

   make: *** [Makefile] Error 1
   ```

   You can ignore the error message. That is just there to prevent the creation of the *graft* executable before you've made your site specific configurations to the

`Makefile`.

4. Edit the `Makefile`. The following variables should be modified to suit your local requirements:

```
PACKAGEDIR      = /pkgs
TARGETDIR       = /pkgs
```

   These two variables control your default package `installation` and `target` directories. Most sites will probably choose to install packages under a common `installation` directory and then *graft* them into a common `target` directory.

   If no specific `target` directory is given on the command line, *Graft* will use the default value specified by **TARGETDIR**. If a `target` directory is given on the command line but is not fully qualified, the value specified by **TARGETDIR** will be prepended to the command line argument.

   Package names provided to *Graft* that are not fully qualified will have the value specified by **PACKAGEDIR** prepended to the command line arguments.

```
TOP             = $(PACKAGEDIR)/graft-$(VERSION)
BIN             = $(TOP)/bin
MAN             = $(TOP)/man
DOC             = $(TOP)/doc
```

   There should be no need to modify these values unless you wish to install *Graft* into a directory that is different from your default package installation directory. If you do modify **TOP** you should not change the values of **BIN**, **MAN** and **DOC**. If you feel you must change these values then perhaps you've misunderstood the concept behind *Graft* so a re-read of this document may be in order.

```
PERL            = /usr/bin/perl
```

   This variable refers to the location of the *Perl* 5.*x* that will be used by the *Graft* executable. If you plan on *grafting Perl* then this value should be the *grafted* location of *Perl* rather than the installation location of *Perl*. If you are using an operating system that comes with *Perl* 5.*x* - such as RedHat or Ubuntu Linux - then you don't need to worry about *grafting Perl* so the value of **PERL** should reflect its installed location.

   Most Unix, Linux and other Unix like operating systems ship with *Perl* these days so modifying this value is probably unnecessary.

```
BUILDPERL       = $(PERL)
```

   Perl is required during the `make`. You'll only need to change this if the current installed location of *Perl* is different to the future *grafted* location of *Perl*.

```
LOGFILE         = /var/log/graft
```

Graft logs all of its actions to a log file. Modify the value of **LOGFILE** to suit your local needs. An alternative name can be specified on the command line.

If you want logging disabled by default, set the value of **LOGFILE** to `/dev/null`.

GRAFT-IGNORE = `.nograft`
GRAFT-EXCLUDE = `.graft-exclude`
GRAFT-INCLUDE = `.graft-include`

These variables hold the names of the special *Graft* files that control whether or not subdirectories or files are *grafted*. If you change these values, try to choose obvious names. If you want the files to appear in a simple directory listing, do not use file names that begin with a dot "." character.

GRAFT-NEVER =

This variable holds the names of the files and/or directories that should never be *grafted*. Typically these may be source code repositories as used by systems such as *CVS*, or perhaps lockfiles. The default value is empty but if you wish to specify values, simply add them to the variable using only whitespace as a separator. For example:

```
GRAFT-NEVER     = CVS RCS SCCS .lock
```

NEVERGRAFT = 0

If this variable is set to **1**, the files and/or directories specified by **GRAFT-NEVER** will be automatically excluded from the *grafted* directory.

If this variable is set to **0**, the files and/or directories specified by **GRAFT-NEVER** will be not be excluded from the *grafted* directory.

The sense of this value is reversed by use of the **-C** command line option.

The automatic exclusion is bypassed completely if the *grafted* directory contains either a `.nograft` or `.graft-include` file.

PRUNED-SUFFIX = `.pruned`

This variable sets the suffix name of *pruned* files. *Pruned* files will be renamed *filename*`.pruned`.

SUPERUSER = 1

If this variable is set to **1** only the superuser can *install*, *delete* or *prune* packages. This can be overridden by a command line option. If this variable is set to **0**, superuser privileges are not required and the override command line option is disabled.

If you are installing a private copy of *Graft* to manage packages in your

home directory you should set **SUPERUSER** to **0**. If you're using *Graft* to manage a global set of packages you should set **SUPERUSER** to **1**.

PRESERVEPERMS = 0

When *grafting* packages, *Graft* will create new directories as required. By setting **PRESERVEPERMS** to **1**, the original user id, group id and file modes will be carried over to the new directory. This variable is used only if **SUPERUSER** is set to **1**. The sense of this variable can be reversed using a command line option.

DELETEOBJECTS = 0

When deleting *grafted* packages, *Graft* may leave empty directories. Setting **DELETEOBJECTS** to **1** will allow *Graft* to delete these directories. If **DELETEOBJECTS** is **0** then *Graft* will display an appropriate message reminding the user that a directory has been emptied. The sense of this variable can be reversed using a command line option.

It's probably not good practise to set this value to **1** as some directories may be used as place holders by a number of different packages. If the value is set to **0** deletion of directories can be forced via a command line option.

When pruning packages, *graft* can either remove conflicting files or rename them. If **DELETEOBJECTS** is set to **1** the default prune action will be to delete conflicting objects. If **DELETEOBJECTS** is set to **0** the default prune action will be to rename conflicting objects. The sense of this variable can be reversed using a command line option.

Save your changes and exit from the editor.

5. Remove any existing executables by running:

```
make clean
```

You should see output similar to:

```
rm -f graft
```

6. Create the *Graft* executable by running:

```
make
```

You should see output similar to:

```
/usr/bin/perl -wc graft.pl
graft.pl syntax OK
sed                                          \
    -e 's#xDELETEOBJECTSx#0#g'               \
    -e 's#xGRAFT-EXCLUDEx#.graft-exclude#g' \
    -e 's#xGRAFT-IGNOREx#.nograft#g'         \
    -e 's#xGRAFT-INCLUDEx#.graft-include#g' \
    -e 's#xGRAFT-NEVERx##g'                  \
```

```
    -e 's#xLOGFILEx#/var/log/graft#g'       \
    -e 's#xNEVERGRAFTx#0#g'                 \
    -e 's#xPACKAGEDIRx#/pkgs#g'             \
    -e 's#xPERLx#/usr/bin/perl#g'           \
    -e 's#xPRESERVEPERMSx#0#g'              \
    -e 's#xPRUNED-SUFFIXx#.pruned#g'        \
    -e 's#xSUPERUSERx#1#g'                  \
    -e 's#xTARGETDIRx#/pkgs#g'              \
    < graft.pl > graft
chmod +x graft
/usr/bin/perl -wc graft
graft syntax OK

manpage=`basename graft.man`;               \
man=`expr graft.man : '.*\.\(.\)'`;         \
sed                                         \
    -e 's#xDELETEOBJECTSx#0#g'              \
    -e 's#xGRAFT-EXCLUDEx#.graft-exclude#g' \
    -e 's#xGRAFT-IGNOREx#.nograft#g'        \
    -e 's#xGRAFT-INCLUDEx#.graft-include#g' \
    -e 's#xGRAFT-NEVERx##g'                 \
    -e 's#xLOGFILEx#/var/log/graft#g'       \
    -e 's#xNEVERGRAFTx#0#g'                 \
    -e 's#xPACKAGEDIRx#/pkgs#g'             \
    -e 's#xPRESERVEPERMSx#0#g'              \
    -e 's#xPRUNED-SUFFIXx#.pruned#g'        \
    -e 's#xSUPERUSERx#1#g'                  \
    -e 's#xTARGETDIRx#/pkgs#g'              \
    -e 's#xVERSIONx#2.6#g'                  \
    < graft.man > graft.1
```

7. If you're using the *automounter* under Solaris 2.*x*, the installation process may not be able to directly create the directory specified by **TOP**. If this is the case then manually create this directory using whatever procedures are appropriate for your operating system.

   For example, if the `/pkgs` mount point is under the control of the *automounter* via an entry in the `auto_pkgs` map:

   ```
   *   nfshost:/export/sparc-SunOS-5.5.1/pkgs/&
   ```

   you'll need to create the *Graft* installation directory by executing the following command on the machine *nfshost*:

   ```
   mkdir /export/sparc-SunOS-5.5.1/pkgs/graft-2.6
   ```

8. Install the *Graft* executable, manual page and documentation by executing:

   ```
   make install
   ```

   You should see output similar to:

   ```
   mkdir -p /pkgs/graft-2.6/bin
   cp graft /pkgs/graft-2.6/bin

   for i in graft.1;                                   \
   do                                                  \
   ```

```
    manpage=`basename $i`;                          \
    man=`expr $i : '.*\.\(.\)'`;                    \
    mkdir -p /pkgs/graft-2.6/man/man$man;           \
    cp $i /pkgs/graft-2.6/man/man$man/$manpage;     \
    chmod 644 /pkgs/graft-2.6/man/man$man/$manpage; \
done

for i in graft.html graft.pdf graft.ps graft.txt;   \
do                                                  \
    mkdir -p /pkgs/graft-2.6/doc;                   \
    cp doc/$i /pkgs/graft-2.6/doc;                  \
    chmod 644 /pkgs/graft-2.6/doc/$i;               \
    touch /pkgs/graft-2.6/doc/.nograft;             \
done
```

Graft is now installed and ready to be used.

NOTE: If you make changes to your *Graft* installation at a later date, please run the following commands:

```
make clean
make install
```

Failure to do this may result in a *Graft* manual page that does NOT reflect your current configuration.

Grafting *Graft* and *Perl* - the bootstrap problem

If you are using an operating system that comes with *Perl* 5.*x* - such as RedHat or Ubuntu Linux - then you don't need to worry about *grafting Perl*, so this section can be ignored.

Embedded into the *Graft* executable is the location of the *Perl* executable. If you've understood the concept behind *Graft* then this location may be the *grafted* location of *Perl* rather than the true location of *Perl*.

This presents a dilemma when you come to *graft* both *Graft* and *Perl*. You can't run the *grafted* location of the *Graft* executable because it doesn't exist yet, and you can't run the real location of the *Graft* executable because *Perl* hasn't been *grafted* yet.

Assuming that *Graft* and *Perl* are installed in

```
/pkgs/graft-2.6
/pkgs/perl-5.18.2
```

you can resolve this dilemma by executing the following commands:

```
/pkgs/perl-5.18.2/bin/perl /pkgs/graft-2.6/bin/graft -i graft-2.6
/pkgs/perl-5.18.2/bin/perl /pkgs/graft-2.6/bin/graft -i perl-5.18.2
```

This will *graft* both *Graft* and *Perl* from the default package installation directory (as specified by **PACKAGEDIR** in the `Makefile`) into your default target directory (as

specified by **TARGETDIR** in the `Makefile`).

If you don't wish to use the default directories you can use the following commands instead:

```
/pkgs/perl-5.18.2/bin/perl /pkgs/graft-2.6/bin/graft -i -t /pkgs /pkgs/graft-2.6
/pkgs/perl-5.18.2/bin/perl /pkgs/graft-2.6/bin/graft -i -t /pkgs /pkgs/perl-5.18.2
```

Now both *Graft* and *Perl* have been *grafted* and any other package can be *grafted* by executing the simpler command:

```
graft -i package
```

The *Graft* 2.6 distribution includes a program called `graftBootStrap.sh` which allows you to easily *graft* both *Graft* and *Perl*. It can be found in the *contrib* directory of the distribution.

Using *Graft*

Compiling Packages

Any packages you wish to place under the control of *Graft* should be compiled and installed in such a way that any package dependent files are referenced with the ACTUAL package installation directory rather than the common area in which *Graft* will be creating symbolic links. For example, ensure that *Perl* version *5.18.2* is looking for its library files in `/pkgs/perl-5.18.2/lib/perl5` instead of `/pkgs/lib/perl5`. This approach will allow you to easily separate multiple versions of the same package without any problems.

Graft command line options

All of the details concerning actions, package locations and target directories are passed to *Graft* on the command line. (*Graft* 1.*x* used a configuration file. This has now been deprecated in favour of a log file).

Graft's command line options can be summarised as:

```
graft -i [-P|u] [-l log] [-n] [-v|V] [-s|-t target] package(s)
graft -d [-D] [-u] [-l log] [-n] [-v|V] [-s|-t target] package(s)
graft -p [-D] [-u] [-l log] [-n] [-v|V] [-s|-t target] package(s)
```

Graft has three basic actions:

1. **Install**

   ```
   graft -i [-C] [-P|u] [-l log] [-n] [-v|V] [-s|-t target] package(s)
   ```

   -i

Install symbolic links from the package installation directory to the target directory. Requires superuser privileges if **SUPERUSER** was set to **1** in the `Makefile`.

-C

If **NEVERGRAFT** was set to **1** in the `Makefile`, disable the automatic exclusion of files and/or directories whose names exactly match the values specified by **GRAFT-NEVER** in the `Makefile`.

If **NEVERGRAFT** was set to **0** in the `Makefile`, force the automatic exclusion of files and/or directories whose names exactly match the values specified by **GRAFT-NEVER** in the `Makefile`.

Can only be used with the -i option.

This option is ignored for each *grafted* directory, if the directory contains a `.nograft` or `.graft-include` file.

The *Graft* manual page will correctly reflect the status of this option based on the values specified in the `Makefile`.

-P

Preserve modes and ownerships when creating new directories if **PRESERVEPERMS** was set to **0** in the `Makefile`. Do not preserve modes and ownerships if the option is not provided on the command line.

Do not preserve modes and ownerships when creating new directories if **PRESERVEPERMS** was set to **1** in the `Makefile`. Preserve modes and ownerships if the option is not provided on the command line.

Cannot be used with the **-u** option.

This option will be silently ignored if the effective user of *Graft* is not root.

This option does not apply if **SUPERUSER** was set to **0** in the `Makefile`.

The *Graft* manual page will correctly reflect the status of this option based on the values specified in the `Makefile`.

-u

Superuser privileges are not required when installing packages.

Cannot be used with the **-P** option.

This option is only available if **SUPERUSER** was set to **1** in the `Makefile`.

The *Graft* manual page will correctly reflect the status of this option based on the values specified in the `Makefile`.

-l *log*

Specify an alternate log file instead of the default specified by **LOGFILE** in the `Makefile`. No logging is performed if the **-n** option is used.

Log entries have the form:

```
878790215   1.10+   I    /pkgs/cpio-2.4.2           /pkgs
878888916   2.1     I    /pkgs/gzip-1.2.4           /pkgs
878888916   2.1     IC   /pkgs/gzip-1.2.4/bin/gzip   invalid symlink
```

This shows that a development version of *graft* (1.10+) was used to install symbolic links from `/pkgs/cpio-2.4.2` to `/pkgs`. A new version of *graft* (2.1) was used to install symbolic links from `/pkgs/gzip-1.2.4` to `/pkgs`. The `IC` entry indicates that a conflict occurred during this installation - the file `/pkgs/bin/gzip` was a symbolic link to something other than `/pkgs/gzip-1.2.4/bin/gzip`.

-n

List actions but do not perform them. Implies the very verbose option. Does not require superuser privileges regardless of the value of **SUPERUSER** in the `Makefile`.

-v

Be verbose.

-V

Be very verbose.

-s

Stow/*Depot* compatibility mode. Infer the *Graft* target directory from each package installation directory in the manner of *Stow* and *Depot*.

Target directory is the `dirname` of the `dirname` of the package installation directory. (Yes that really is two `dirname`s). So if the package installation directory is

```
/usr/local/depot/gzip-1.2.4
```

the package will be *grafted* into

```
/usr/local
```

Cannot be used with the **-t** option.

-t *target*

Override the default *graft* target directory with ***target***. The value of ***target***

must be a fully qualified directory and it must exist.

Cannot be used with the **-s** option.

package

Install the named package. If **package** is a fully qualified directory, use it as the package installation directory. If **package** is not a fully qualified directory, prepend it with the value of **PACKAGEDIR** as specified in the `Makefile`.

---

2. **Delete**

```
graft -d [-D] [-u] [-l log] [-n] [-v|V] [-s|-t target] package(s)
```

-d

Delete symbolic links from the package target directory to the package installation directory. Requires superuser privileges if **SUPERUSER** was set to **1** in the `Makefile`.

-D

Delete empty directories if **DELETEOBJECTS** was set to **0** in the `Makefile`. If the option is not provided on the command line, notify the user that a directory has been emptied.

Do not delete empty directories if **DELETEOBJECTS** was set to **1** in the `Makefile`. Notify the user that a directory has been emptied. If the option is not provided on the command line, delete empty directories.

The *Graft* manual page will correctly reflect the status of this option based on the values specified in the `Makefile`.

-u

Superuser privileges are not required when deleting packages.

This option is only available if **SUPERUSER** was set to **1** in the `Makefile`.

The *Graft* manual page will correctly reflect the status of this option based on the values specified in the `Makefile`.

-l *log*

Specify an alternate log file instead of the default specified by **LOGFILE** in the `Makefile`. No logging is performed if the **-n** option is used.

Log entries have the form:

```
    879126278       1.10+   D       /pkgs/weblint-1.017     /pkgs
```

```
879126278       1.10+   DC      /pkgs/weblint-1.017/bin/weblint  file exists
879126278       1.10+   DC      /pkgs/weblint-1.017/man/man1/weblint.1  file exists
```

This shows that a development version of *graft* (1.10+) was used to delete symbolic links from `/pkgs` to `/pkgs/weblint-1.017`. The `DC` entries indicate that conflicts occurred during this action - the files `/pkgs/bin/weblint` and `/pkgs/man/man1/weblint.1` already exist.

-n

List actions but do not perform them. Implies the very verbose option. Does not require superuser privileges regardless of the value of **SUPERUSER** in the `Makefile`.

-v

Be verbose.

-V

Be very verbose.

-s

Stow/*Depot* compatibility mode. Infer the *Graft* target directory from each package installation directory in the manner of *Stow* and *Depot*.

Target directory is the `dirname` of the `dirname` of the package installation directory. (Yes that really is two `dirname`s). So if the package installation directory is

```
/usr/local/depot/gzip-1.2.4
```

the package will be *grafted* into

```
/usr/local
```

Cannot be used with the **-t** option.

-t *target*

Override the default *graft* target directory with ***target***. The value of ***target*** must be a fully qualified directory and it must exist.

Cannot be used with the **-s** option.

package

Delete the named package. If **package** is a fully qualified directory, use it as the package installation directory. If **package** is not a fully qualified directory, prepend it with the value of **PACKAGEDIR** as specified in the `Makefile`.

3. **Prune**

```
graft -p [-D] [-u] [-l log] [-n] [-v|V] [-s|-t target] package(s)
```

-p

Prune objects (files, links or directories) from the package target directory that are in conflict with the package installation directory. Requires superuser privileges if **SUPERUSER** was set to **1** in the `Makefile`.

-D

Remove conflicting objects if **DELETEOBJECTS** was set to **0** in the `Makefile`. Rename conflicting objects as *`object`*`.pruned` if the option is not provided on the command line.

Rename conflicting objects to *`object`*`.pruned` if **DELETEOBJECTS** was set to **1** in the `Makefile`. Remove conflicting objects if the option is not provided in the command line.

If a directory is to be removed and it is not empty, it will be renamed as *`dir`*`.pruned` and a suitable warning message will be given regardless of the sense of this flag.

The *Graft* manual page will correctly reflect the status of this option based on the values specified in the `Makefile`.

-u

Superuser privileges are not required when pruning packages.

This option is only available if **SUPERUSER** was set to **1** in the `Makefile`.

The *Graft* manual page will correctly reflect the status of this option based on the values specified in the `Makefile`.

-l *log*

Specify an alternate log file instead of the default specified by **LOGFILE** in the `Makefile`. No logging is performed if the **-n** option is used.

Log entries have the form:

```
879126283       1.10+   P       /pkgs/weblint-1.017     /pkgs
```

This shows that a development version of *graft* (1.10+) was used to delete objects from `/pkgs` that were in conflict with `/pkgs/weblint-1.017`.

-n

List actions but do not perform them. Implies the very verbose option. Does

not require superuser privileges regardless of the value of **SUPERUSER** in the `Makefile`.

-v

Be verbose.

-V

Be very verbose.

-s

Stow/*Depot* compatibility mode. Infer the *Graft* target directory from each package installation directory in the manner of *Stow* and *Depot*.

Target directory is the `dirname` of the `dirname` of the package installation directory. (Yes that really is two `dirname`s). So if the package installation directory is

```
/usr/local/depot/gzip-1.2.4
```

the package will be *grafted* into

```
/usr/local
```

Cannot be used with the **-t** option.

-t *target*

Override the default *graft* target directory with ***target***. The value of ***target*** must be a fully qualified directory and it must exist.

Cannot be used with the **-s** option.

package

Prune the named package. If **package** is a fully qualified directory, use it as the package installation directory. If **package** is not a fully qualified directory, prepend it with the value of **PACKAGEDIR** as specified in the `Makefile`.

Testing the *Graft* Installation

Before creating the symbolic links from the target directory to the package directory, you may wish to see what actions *Graft* will perform. Execute the following command:

```
graft -i -n package-name
```

The `-i` option tells *Graft* to install the package and the `-n` option tells *Graft* to report on its actions without actually performing them. The default *Graft* target directory

will be used and the package installation directory will be taken from the fully qualified package argument or the default value will be prepended to the package argument if it is not fully qualified.

Graft will report on the following actions:

- Installing links to *package-location* in *package-target*

  Indicates the real package location and its *grafted* target.

- Processing *package-directory*

  Indicates which package directory is being processed.

- MKDIR *dirname*

  This destination directory will be created.

- SYMLINK *dest-package-file* -> *package-file*

  This symbolic link will be created.

- NOP *string*

  No action was necessary for this package object.

- BYPASS *dirname* - .nograft file found

  This directory contains a file called `.nograft` so its contents and any subdirectories will be bypassed by *Graft*.

- READING include file *package-dir*`/.graft-include`

  The directory currently being processed by *Graft* contains a file called `.graft-include` which contains a list of file and/or directory names from the directory that should only be *grafted*. The contents of this file are being read by *Graft*.

- INCLUDE file *package-file* - listed in *package-dir*`/.graft-include`

  The file name mentioned in this message appears in the *.graft-include* file and the file exists in the directory currently being processed. It will be `grafted`.

- IGNORE file *package-file* - not listed in *package-dir*`/.graft-include`

  The file name mentioned in this message does not appear in the *.graft-include* file and the file exists in the directory currently being processed. It will not be `grafted`.

- INCLUDE directory *package-directory* - listed in *package-dir*`/.graft-include`

  The directory name mentioned in this message appears in the *.graft-include* file and the directory exists in the directory currently being processed. It will be

grafted.

- IGNORE directory *package-file* - not listed in *package-dir*/.graft-include

  The directory name mentioned in this message does not appear in the *.graft-include* file and the directory exists in the directory currently being processed. It will not be grafted.

- READING exclude file *package-dir*/.graft-exclude

  The directory currently being processed by *Graft* contains a file called .graft-exclude which contains a list of file and/or directory names from the directory that should not be *grafted*. The contents of this file are being read by *Graft*.

- IGNORE include file *package-dir*/.graft-include, overridden by exclude file *package-dir*/.graft-exclude

  The directory currently being processed by *Graft* contains a file called .graft-exclude as well as a file called .graft-include. The .graft-exclude file takes precedence over the .graft-include file, so the latter file will be ignored.

- EXCLUDE file *package-file* - listed in *package-dir*/.graft-exclude

  The file name mentioned in this message appears in the *.graft-exclude* file and the file exists in the directory currently being processed. It will not be grafted.

- EXCLUDE directory *package-directory* - listed in *package-dir*/.graft-exclude

  The directory name mentioned in this message appears in the *.graft-exclude* file and the directory exists in the directory currently being processed. It will not be grafted.

- CONFLICT *message*

  Graft could not successfully process a package object. One of the following conditions was encountered:

  - The package object is a directory and the target object exists but it not a directory.
  - The package object is not a directory and the target object exists and is not a symbolic link.
  - The package object is not a directory and the target object exists and is a symbolic link to something other than the package object.

  Conflicts are ALWAYS reported on standard error. If you wish to see if the installation of a package will have any conflicts, you can execute:

  ```
  graft -i -n package-name > /dev/null
  ```

  Only CONFLICT messages will be displayed. If nothing is displayed then you can safely conclude that this package can be installed using *Graft* without any

conflicts.

If you were to test the installation of the *kermit-5A190* package you would execute the command:

```
graft -i -n kermit-5A190
```

You should see output resembling:

```
Installing   links to /pkgs/kermit-5A190 in /pkgs
Processing   /pkgs/kermit-5A190
SYMLINK      /pkgs/README -> /pkgs/kermit-5A190/README
NOP          /pkgs/kermit-5A190/bin and /pkgs/bin are both directories
Processing   /pkgs/kermit-5A190/bin
SYMLINK      /pkgs/bin/kermit -> /pkgs/kermit-5A190/bin/kermit
SYMLINK      /pkgs/bin/wart -> /pkgs/kermit-5A190/bin/wart
NOP          /pkgs/kermit-5A190/man and /pkgs/man are both directories
Processing   /pkgs/kermit-5A190/man
NOP          /pkgs/kermit-5A190/man/man1 and /pkgs/man/man1 are both directories
Processing   /pkgs/kermit-5A190/man/man1
SYMLINK      /pkgs/man/man1/kermit.1 -> /pkgs/kermit-5A190/man/man1/kermit.1
MKDIR        /pkgs/doc
Processing   /pkgs/kermit-5A190/doc
SYMLINK      /pkgs/doc/ckccfg.doc -> /pkgs/kermit-5A190/doc/ckccfg.doc
SYMLINK      /pkgs/doc/ckuins.doc -> /pkgs/kermit-5A190/doc/ckuins.doc
SYMLINK      /pkgs/doc/ckc190.upd -> /pkgs/kermit-5A190/doc/ckc190.upd
SYMLINK      /pkgs/doc/ckcker.upd -> /pkgs/kermit-5A190/doc/ckcker.upd
SYMLINK      /pkgs/doc/ckaaaa.hlp -> /pkgs/kermit-5A190/doc/ckaaaa.hlp
SYMLINK      /pkgs/doc/ckuaaa.hlp -> /pkgs/kermit-5A190/doc/ckuaaa.hlp
NOP          /pkgs/kermit-5A190/lib and /pkgs/lib are both directories
Processing   /pkgs/kermit-5A190/lib
SYMLINK      /pkgs/lib/ckedemo.ini -> /pkgs/kermit-5A190/lib/ckedemo.ini
SYMLINK      /pkgs/lib/ckeracu.ini -> /pkgs/kermit-5A190/lib/ckeracu.ini
SYMLINK      /pkgs/lib/ckermit.ini -> /pkgs/kermit-5A190/lib/ckermit.ini
SYMLINK      /pkgs/lib/ckermod.ini -> /pkgs/kermit-5A190/lib/ckermod.ini
SYMLINK      /pkgs/lib/cketest.ini -> /pkgs/kermit-5A190/lib/cketest.ini
SYMLINK      /pkgs/lib/ckevt.ini -> /pkgs/kermit-5A190/lib/ckevt.ini
SYMLINK      /pkgs/lib/ckurzsz.ini -> /pkgs/kermit-5A190/lib/ckurzsz.ini
```

This output shows you that most of the directories already exist (indicated by the `NOP` flags). A symbolic link will be created in the relevant target directory to each of the files in the *kermit-5A190* package. One directory exists in the *kermit-5A190* package that does not exist in the target - `doc`. This directory will be created by *Graft*.

NOTE: If you are using the *automounter* you may not be able to create the directory `/pkgs/doc`. You'll have to create the directory on the NFS server under the file system in which it really lives. You should be familiar with the peculiarities of the *automounter* and your specific site configuration before creating any directories directly under mount points used by the *automounter*.

Installing Packages

Once you have ensured that *Graft* will perform the correct actions, you can execute:

```
graft -i package-name
```

So to install *kermit* you would execute:

```
graft -i kermit-5A190
```

There will be no output from *Graft* unless it encounters a conflict. If you wish to see more information you can specify one of the verbose flags. For a minimum of output you can execute:

```
graft -i -v kermit-5A190
```

You should see the following output:

```
Processing   /pkgs/kermit-5A190
Processing   /pkgs/kermit-5A190/bin
Processing   /pkgs/kermit-5A190/man
Processing   /pkgs/kermit-5A190/man/man1
Processing   /pkgs/kermit-5A190/doc
Processing   /pkgs/kermit-5A190/lib
```

If you choose the very verbose option by executing:

```
graft -i -V kermit-5A190
```

the output will be the same as that when the `-n` option was used, however this time *Graft* will actually create the symbolic links.

```
Installing   links to /pkgs/kermit-5A190 in /pkgs
Processing   /pkgs/kermit-5A190
SYMLINK      /pkgs/README -> /pkgs/kermit-5A190/README
NOP          /pkgs/kermit-5A190/bin and /pkgs/bin are both directories
Processing   /pkgs/kermit-5A190/bin
SYMLINK      /pkgs/bin/kermit -> /pkgs/kermit-5A190/bin/kermit
SYMLINK      /pkgs/bin/wart -> /pkgs/kermit-5A190/bin/wart
NOP          /pkgs/kermit-5A190/man and /pkgs/man are both directories
Processing   /pkgs/kermit-5A190/man
NOP          /pkgs/kermit-5A190/man/man1 and /pkgs/man/man1 are both directories
Processing   /pkgs/kermit-5A190/man/man1
SYMLINK      /pkgs/man/man1/kermit.1 -> /pkgs/kermit-5A190/man/man1/kermit.1
NOP          /pkgs/kermit-5A190/doc and /pkgs/doc are both directories
Processing   /pkgs/kermit-5A190/doc
SYMLINK      /pkgs/doc/ckccfg.doc -> /pkgs/kermit-5A190/doc/ckccfg.doc
SYMLINK      /pkgs/doc/ckuins.doc -> /pkgs/kermit-5A190/doc/ckuins.doc
SYMLINK      /pkgs/doc/ckc190.upd -> /pkgs/kermit-5A190/doc/ckc190.upd
SYMLINK      /pkgs/doc/ckcker.upd -> /pkgs/kermit-5A190/doc/ckcker.upd
SYMLINK      /pkgs/doc/ckaaaa.hlp -> /pkgs/kermit-5A190/doc/ckaaaa.hlp
SYMLINK      /pkgs/doc/ckuaaa.hlp -> /pkgs/kermit-5A190/doc/ckuaaa.hlp
NOP          /pkgs/kermit-5A190/lib and /pkgs/lib are both directories
Processing   /pkgs/kermit-5A190/lib
SYMLINK      /pkgs/lib/ckedemo.ini -> /pkgs/kermit-5A190/lib/ckedemo.ini
SYMLINK      /pkgs/lib/ckeracu.ini -> /pkgs/kermit-5A190/lib/ckeracu.ini
SYMLINK      /pkgs/lib/ckermit.ini -> /pkgs/kermit-5A190/lib/ckermit.ini
SYMLINK      /pkgs/lib/ckermod.ini -> /pkgs/kermit-5A190/lib/ckermod.ini
SYMLINK      /pkgs/lib/cketest.ini -> /pkgs/kermit-5A190/lib/cketest.ini
SYMLINK      /pkgs/lib/ckevt.ini -> /pkgs/kermit-5A190/lib/ckevt.ini
SYMLINK      /pkgs/lib/ckurzsz.ini -> /pkgs/kermit-5A190/lib/ckurzsz.ini
```

NOTE: In this case the `/pkgs/doc` directory was not created by *Graft* because `/pkgs` is a mount point controlled by the *automounter*. The `doc` directory was created manually prior to executing *Graft*.

Bypassing package directories

You may have the need to place only part of a package under the control of *Graft*. Examples of such occasions may be:

- The contents of one package conflict with another package. For example `/pkgs/gcc-2.7.2.1/lib/libiberty.a` and `/pkgs/gdb-4.16/lib/libiberty.a`.
- A package directory is obviously the exclusive domain of the package and no benefit will be gained by creating symbolic links to its files. For example `/pkgs/perl-5.18.2/lib/perl5`.

  NOTE: This will ONLY work if you originally compiled and installed the package such that it refers to its files by their '*real*' pathnames and NOT by the virtual pathnames provided by *Graft*.

You can force *Graft* to bypass a directory by creating the file

```
package-name/dir/dir/.nograft
```

Using the second example above, if you were to create the file:

```
/pkgs/perl-5.18.2/lib/perl5/.nograft
```

Graft would create directories and symbolic links for every file and directory down to `/pkgs/perl-5.18.2/lib`. The `perl5` directory and anything below it would not be created.

Including specific files and/or directories

There may be the occasional need to include specific files and/or directories in a directory, rather than the entire directory tree itself. An example of such an occurrence would be the case where a package contains a number of subdirectories, only one of which is required to be *grafted*.

You can force *Graft* to only include any number of files and/or directories in a package directory by creating the file

```
.graft-include
```

in the same directory.

`.graft-include` will contain a list of file and/or directory names - one per line - of the files and/or directories you wish to include.

Consider the *a2ps* package for example. When installed it contains the following

directories:

```
/pkgs/a2ps-4.13b/bin
/pkgs/a2ps-4.13b/etc
/pkgs/a2ps-4.13b/include
/pkgs/a2ps-4.13b/info
/pkgs/a2ps-4.13b/lib
/pkgs/a2ps-4.13b/man
/pkgs/a2ps-4.13b/share
```

The only directory you wish to *graft* is the `bin` directory. You could place a `.nograft` file in each of the other directories, **OR** you could create a single `.graft-include` file in `/pkgs/a2ps-4.13b/.graft-include`. This file would contain

```
bin
```

Now only the `bin` directory will be *grafted*.

Excluding specific files and/or directories

There may be the occasional need to exclude specific files and/or directories from a directory, rather than the entire directory itself. An example of such an occurrence would be the case where files from different packages have the same name. *Emacs* and *Xemacs* use the same names for a number of their configuration files for example.

You can force *Graft* to exclude any number of files and/or directories from a package directory by creating the file

```
.graft-exclude
```

in the same directory.

`.graft-exclude` will contain a list of file and/or directory names - one per line - of the files and/or directories you wish to exclude.

For example, if you did not wish the file

```
/pkgs/sudo-1.5.3/etc/sudoers
```

to be *grafted* as

```
/pkgs/etc/sudoers
```

but you did want

```
/pkgs/sudo-1.5.3/etc/visudo
```

to be *grafted* as

```
/pkgs/etc/visudo
```

you would create the file

```
/pkgs/sudo-1.5.3/etc/.graft-exclude
```

and ensure its contents contained the line:

```
sudoers
```

NOTE: Any entries made in a `.graft-exclude` file will override the same entries made in a `.graft-include` file. That is, if a file or directory name is listen in both a `.graft-exclude` and a `.graft-include` file, it will be **excluded** from the *graft*.

Grafting part of a package

Some packages can be successfully used when only part of their installation directory is *grafted*. Other packages are recalcitrant and need some special hand holding which can only be solved by *grafting* each section of the package separately.

The first scenario can be handled by either `.nograft` files or partial *grafts*. Consider *Perl* version *5.18.2*. When installed in its own directory

```
/pkgs/perl-5.18.2
```

there are three subdirectories

```
drwxr-sr-x   2 psamuel  bisg         512 Oct 30  1996 bin
drwxr-sr-x   3 psamuel  bisg         512 Oct 30  1996 lib
drwxr-sr-x   4 psamuel  bisg         512 Oct 30  1996 man
```

Everything in the `lib` directory is exclusive to *Perl* and does not require *grafting*. Therefore, *perl-5.18.2* can be *grafted* using either of the following two methods:

```
touch /pkgs/perl-5.18.2/lib/.nograft
graft -i perl-5.18.2
```

or

```
graft -it /pkgs/bin perl-5.18.2/bin
graft -it /pkgs/man perl-5.18.2/man
```

Now let's consider a recalcitrant package - *ObjectStore* version 4.0.2.a.0. When installed in

```
/pkgs/ostore-4.0.2.a.0
```

the following files and directories are available:

```
-rwxrwxr-x   1 pauln    one3        1089 Oct 31  1996 Copyright
drwxrwxrwx   8 pauln    one3         512 Oct  2  1996 common
drwxrwxrwx   6 pauln    one3         512 Oct 31  1996 sunpro
-rw-r-----   1 root     one3     1900544 Apr 29  1997 txn.log
```

The executable programs that need to be *grafted* are in `sunpro/bin` and the manual pages that need to be *grafted* are in `common/man`. Everything else in the package does not need to be *grafted*. If the entire package was to be *grafted* the result would be

two directories that are not in the regular **$PATH** and **$MANPATH** environment variables - namely `/pkgs/common/man` and `/pkgs/sunpro/bin`, plus a host of other directories that are not relevant for *grafting*. No amount of `.nograft` and `.graft-exclude` juggling will solve this problem.

The solution is to use two partial *grafts*:

```
graft -it /pkgs/bin ostore-4.0.2.a.0/sunpro/bin
graft -it /pkgs/man ostore-4.0.2.a.0/common/bin
```

Using this approach, the correct executables and manual pages are available without the need to *graft* unnecessary files and directories.

Deleting and/or Upgrading Packages

If you wish to upgrade a package - let's assume you wish to upgrade *kermit* from version 5A190 to version 6.0.192 - you'd follow these steps.

Firstly, you'd compile and install *kermit-6.0.192* in

```
/pkgs/kermit-6.0.192
```

Once you'd tested it to your satisfaction, you'd need to delete the symbolic links to the current *grafted* version. You can check which actions *Graft* will perform by executing:

```
graft -d -n kermit-5A190
```

You'll see output similar to

```
Uninstalling links from /pkgs to /pkgs/kermit-5A190
Processing   /pkgs/kermit-5A190
Processing   /pkgs/kermit-5A190/bin
UNLINK       /pkgs/bin/kermit
UNLINK       /pkgs/bin/wart
Processing   /pkgs/kermit-5A190/man
Processing   /pkgs/kermit-5A190/man/man1
UNLINK       /pkgs/man/man1/kermit.1
Processing   /pkgs/kermit-5A190/doc
UNLINK       /pkgs/doc/ckccfg.doc
UNLINK       /pkgs/doc/ckuins.doc
UNLINK       /pkgs/doc/ckc190.upd
UNLINK       /pkgs/doc/ckcker.upd
UNLINK       /pkgs/doc/ckaaaa.hlp
UNLINK       /pkgs/doc/ckuaaa.hlp
Processing   /pkgs/kermit-5A190/lib
UNLINK       /pkgs/lib/ckedemo.ini
UNLINK       /pkgs/lib/ckeracu.ini
UNLINK       /pkgs/lib/ckermit.ini
UNLINK       /pkgs/lib/ckermod.ini
UNLINK       /pkgs/lib/cketest.ini
UNLINK       /pkgs/lib/ckevt.ini
UNLINK       /pkgs/lib/ckurzsz.ini
UNLINK       /pkgs/lib/.testing
```

If you're happy with the output from the test deletion you can delete the *grafted* package. Once again, you'll only see output if a failure occurs unless you use one of the verbose options.

If you execute:

```
graft -dV kermit-5A190
```

you'll see:

```
Uninstalling links from /pkgs to /pkgs/kermit-5A190
Processing   /pkgs/kermit-5A190
Processing   /pkgs/kermit-5A190/bin
UNLINK       /pkgs/bin/kermit
UNLINK       /pkgs/bin/wart
Processing   /pkgs/kermit-5A190/man
Processing   /pkgs/kermit-5A190/man/man1
UNLINK       /pkgs/man/man1/kermit.1
Processing   /pkgs/kermit-5A190/doc
UNLINK       /pkgs/doc/ckccfg.doc
UNLINK       /pkgs/doc/ckuins.doc
UNLINK       /pkgs/doc/ckc190.upd
UNLINK       /pkgs/doc/ckcker.upd
UNLINK       /pkgs/doc/ckaaaa.hlp
UNLINK       /pkgs/doc/ckuaaa.hlp
EMPTY        /pkgs/doc is now empty. Delete manually if necessary.
Processing   /pkgs/kermit-5A190/lib
UNLINK       /pkgs/lib/ckedemo.ini
UNLINK       /pkgs/lib/ckeracu.ini
UNLINK       /pkgs/lib/ckermit.ini
UNLINK       /pkgs/lib/ckermod.ini
UNLINK       /pkgs/lib/cketest.ini
UNLINK       /pkgs/lib/ckevt.ini
UNLINK       /pkgs/lib/ckurzsz.ini
```

NOTE: In this case the existence of an empty directory has been discovered. If *Graft* empties a directory during a package deletion, it will either notify you or delete the directory depending on the combination of variables in the `Makefile` and command line options. It's probably better practise not to automatically delete empty directories as they may be used by other packages - such as lock file directories for example.

Now you can remove the *real* package contents. (You may not wish to do this immediately as some legacy systems may depend on features provided by the older version or you may feel the need for further testing before feeling confident that the old version can be removed):

```
rm -rf /pkgs/kermit-5A190
```

Now you can *graft* the new version of *kermit*. Execute:

```
graft -i -n kermit-6.0.192
```

to ensure that the *grafting* will proceed without error. Once you are satisfied that this is the case you can *graft* the new package by executing:

```
graft -i kermit-6.0.192
```

Transitioning a package to *Graft* control

Graft can be used to easily transition a package from its current installation in your target directory to a *grafted* installation.

As an example, let's consider the package *weblint* version 1.017. It consists of three files installed in:

```
/usr/local/bin/weblint
/usr/local/lib/global.weblintrc
/usr/local/man/man1/weblint.1
```

The first step is to create a new copy of the package in its own directory:

```
/pkgs/weblint-1.017
```

Ensure that any references to library files are now made to `/pkgs/weblint-1.017/lib` instead of `/usr/local/lib`.

Test the new installation to ensure it behaves as expected.

Then prune the old files from `/usr/local/*` using:

```
graft -pVt /usr/local weblint-1.017
```

You'd expect to see output similar to:

```
Pruning      files in /usr/local which conflict with /pkgs/weblint-1.017
Processing   /pkgs/weblint-1.017
Processing   /pkgs/weblint-1.017/man
Processing   /pkgs/weblint-1.017/man/man1
RENAME       /usr/local/man/man1/weblint.1
Processing   /pkgs/weblint-1.017/bin
RENAME       /usr/local/bin/weblint
Processing   /pkgs/weblint-1.017/lib
RENAME       /usr/local/lib/global.weblintrc
```

If you elected to delete conflicting files instead of renaming them you'd use:

```
graft -pDVt /usr/local weblint-1.017
```

and you'd see output similar to:

```
Pruning      files in /usr/local which conflict with /pkgs/weblint-1.017
Processing   /pkgs/weblint-1.017
Processing   /pkgs/weblint-1.017/man
Processing   /pkgs/weblint-1.017/man/man1
UNLINK       /usr/local/man/man1/weblint.1
Processing   /pkgs/weblint-1.017/bin
UNLINK       /usr/local/bin/weblint
Processing   /pkgs/weblint-1.017/lib
UNLINK       /usr/local/lib/global.weblintrc
```

Now the new version of *weblint* 1.017 can be *grafted* in place:

```
graft -it /usr/local weblint-1.017
```

The *grafted* version of *weblint* can now be tested.

If we renamed conflicting files, they can be removed once the *grafted weblint* has been satisfactorily tested:

```
rm /usr/local/man/man1/weblint.1.pruned
rm /usr/local/bin/weblint.pruned
rm /usr/local/lib/global.weblintrc.pruned
```

Conflict Processing

Occasionally *Graft* will fail to completely install a package. This occurs because *Graft* encounters a conflict. A conflict is defined as one of the following possibilities:

Package Object	Target Object
directory	not a directory
file	directory
file	file
file	symbolic link to something other than the package object

If *Graft* encounters such a conflict during the installation of a package it will report the conflict and exit.

Resolving the conflict depends on the nature of the conflict and is beyond the scope of this discussion - however most conflicts will either be the result of attempting to *graft* a package on top of the same package actually installed in the target directory or a file name clash between two (or more) different packages.

Conflicts arising from the pre-existence of a package in the target directory can be resolved using *graft*'s prune mechanism described above in "Transitioning a package to *Graft* control".

File name clash conflicts can be resolved by the use of either a `.nograft` or `.graft-exclude` file or by *grafting* only part of a package as described above in "Grafting part of a package".

If *Graft* encounters a conflict while deleting a package, it will report the conflict and continue deleting the remainder of the package. In this way *Graft* will delete as much of the package as possible. Conflicts that arise during deletion will probably be the result of an incorrectly installed package or the installation of other components of the same package without the use of *Graft*.

Conflict messages are written to standard error. All other messages are written to

standard output. To quickly determine if a package will have any conflicts when *grafted*, redirect standard output to standard error:

```
graft -i -n package > /dev/null
```

If you don't see any output then you can safely assume that there will be no conflicts when *grafting* this package.

Exit Status

Graft will terminate with an exit status of either 0, 1, 2 or 3 under the following conditions:

Exit Status	Condition
0	All operations succeeded.
1	A conflict occurred during installation.
2	Command line syntax was incorrect.
3	One or more packages listed on the command line does not exist. Other valid packages listed on the command line were processed correctly.

Using *Graft* with other package management tools

Most Unix vendors have released their own package management tools with their operating systems. Examples of this are Solaris 2.*x* with its *SVR4 Package Manager* `pkgadd`, RedHat Linux with its *RedHat Package Manager* `rpm`, Ubuntu Linux (and other Debian Linux derivatives) with its `dpkg` system and HP-UX 10.*x* with its `swinstall` suite. *Graft* has been designed as an adjunct to these package managers rather than a competitor. The author has used *Graft* successfully with all of the operating systems mentioned here.

- Many useful packages available in the public domain and from other commercial sources are not shipped with most flavours of Unix. *Graft* can be used to maintain a rich package environment beyond the set of packages provided by your vendor. Vendor based packages can still be maintained using the vendor's tools and *Graft* can be used to maintain your own packages.

- The vendor based management tools are usually used to maintain single instances of a package on each machine. It is often difficult to have multiple versions of the same package coexisting on the same machine. *Graft* can be used to maintain multiple versions of a package to support legacy, production and development requirements simultaneously.

- Another common problem with vendor supplied software is the speed at which

upgrades are available. The large vendors are not known for providing quick fixes to many of their packages. (Notable exceptions to this are the vendors of operating systems based on open source software who can draw on the enormous number of users who submit patches because the source code is available). Using *Graft* you can obtain a working public domain version of a package (if one exists of course) and install it in a different location to the vendor copy. When the vendor releases a new version of the package, it can be installed using the vendor's package management tool and your *grafted* copy can be removed (only if the vendor's version of the package is better than the public domain version).

- Sometimes, a vendor's package doesn't quite perform in the manner you'd like. It may be making assumptions about your file system(s) that are incorrect for your environment or it may not have all the features you'd like. If an alternative package is available - either in the public domain or from other commercial sources - it can be installed and *grafted* accordingly.

Availability

The latest version of *Graft* should always be available from:

http://peters.gormand.com.au/Home/tools/graft

License
