Resurrecting “cruft”

A tool for cleaning up the system and finding packaging bugs

Marcin Owsiany

May 5, 2007

1 Introduction

“cruft” is, in a nutshell, a tool for finding things on a system which should not be there but are, or aren’t but should be. This paper was prepared for a BoF session about this tool, to take place during Debian Conference 7, in Edinburgh. Its aim is to present cruft history, current status, and possible future developments.

2 History

“cruft” was written in early 1998 by Anthony Towns. The initial package consisted of a few small programs in C, and a dozen or so shell scripts. The most important of these shell scripts was called “cruft”, whose function was to prepare two lists:

- of files *exist* on the system,
- of files *should exist* on a system

and present a report on the difference between the actual and required contents of the filesystem. The first list was simply obtained by running `find`. The bulk of this second list was generated directly from the dpkg’s installed files database.

However even after taking care of files created by the local administrator, there were still many files that cruft did not know about, such as:

- files created during installation by the maintainer scripts, rather than being installed directly by dpkg. Examples include `/etc/passwd`, `/etc/fstab`, `/var/www/index.html`
- files created during normal system usage, for example PID files, log files, cache and spool files, database files, etc

In order to take care of these files, a policy change to register such files with dpkg was proposed [2] during a discussion [1] on the debian-policy mailing list in April 1998. Apart from letting cruft know which files should be on the system, this would have an additional significant advantage of teaching `dpkg --search` about such files. Everybody in the discussion agreed that it is a good idea, and what basic semantics such mechanism should have. However consensus was not reached on exact syntax of patterns specifying the extra files. Anthony proposed a format similar to the one currently used by Apache Ant for specifying “filesets” (supporting the special expression `**`, meaning any number of any path elements), while Ian Jackson preferred standard shell globbing patterns “a la `fnmatch(3)`”, with `*` matching `/`.

Possibly because of the lack of consensus on the pattern format, and/or lack of determined effort, the proposed feature was never added to dpkg, and policy was not amended, despite the issue being raised on at least two more occasions on debian-devel, in 2001 [3] and 2004 [4]. Thus, the last only real non-NMU happened in November 1999 and cruft has gradually been bit-rotting since then (over 50 bugs in the BTS).

During DebConf 5, in Helsinki in 2005 Anthony granted permission to upload fixes to cruft to the author of this paper. Bit by bit, most bugs were fixed in subsequent NMUs (2 non-wishlist bugs left at the time of writing of this paper).

Some of the code has been rewritten, because it was not possible to deal with a few problems with the original cruft — the inability for the system administrator to ignore some broken symlinks being the most prominent one. Some features were added, and code was considerably cleaned up and simplified. Current status can be summed up as "does not report false-positives on a fresh base installation".

Currently cruft source code is kept in a Subversion repository, hosted on alioth [6]. This hopefully makes it easier to contribute changes to the numerous filters and explain scripts which the package contains.

3 How cruft works

Cruft does its work in a few stages, described in the following subsections. They are also presented visually in figure 1 on page 3.

3.1 Existing files

In the first stage, a list of files **actually present** on the system is created. It is produced by running `find` on each mounted filesystem in turn, except for:

- filesystems of type such as nfs, proc, etc
- directories (and their contents) specified by `--ignore`

This list is put into `/var/spool/cruft/file_*` (one file per filesystem scanned).

As "find" scans the filesystem, it also creates lists of files of certain type, which are then inspected at a later stage (see section 3.5).

3.2 Explain scripts

In the second stage, cruft runs a selection of so called "explain" scripts, which are located in two directories:

- `/usr/lib/cruft/explain` — explain scripts installed by the "cruft" package,
- `/etc/cruft/explain` — meant for local explain scripts created by the administrator

The role of these scripts is to "explain" to cruft what files should or should not be present. Selecting which of the explain scripts to run is based on two rules:

- any file in `/etc/cruft/explain` overrides a file in `/usr/lib/cruft/explain` with the same name. So if you want to override `/usr/lib/cruft/explain/`*foo*, just create an executable file called `/etc/cruft/explain/`*foo*.

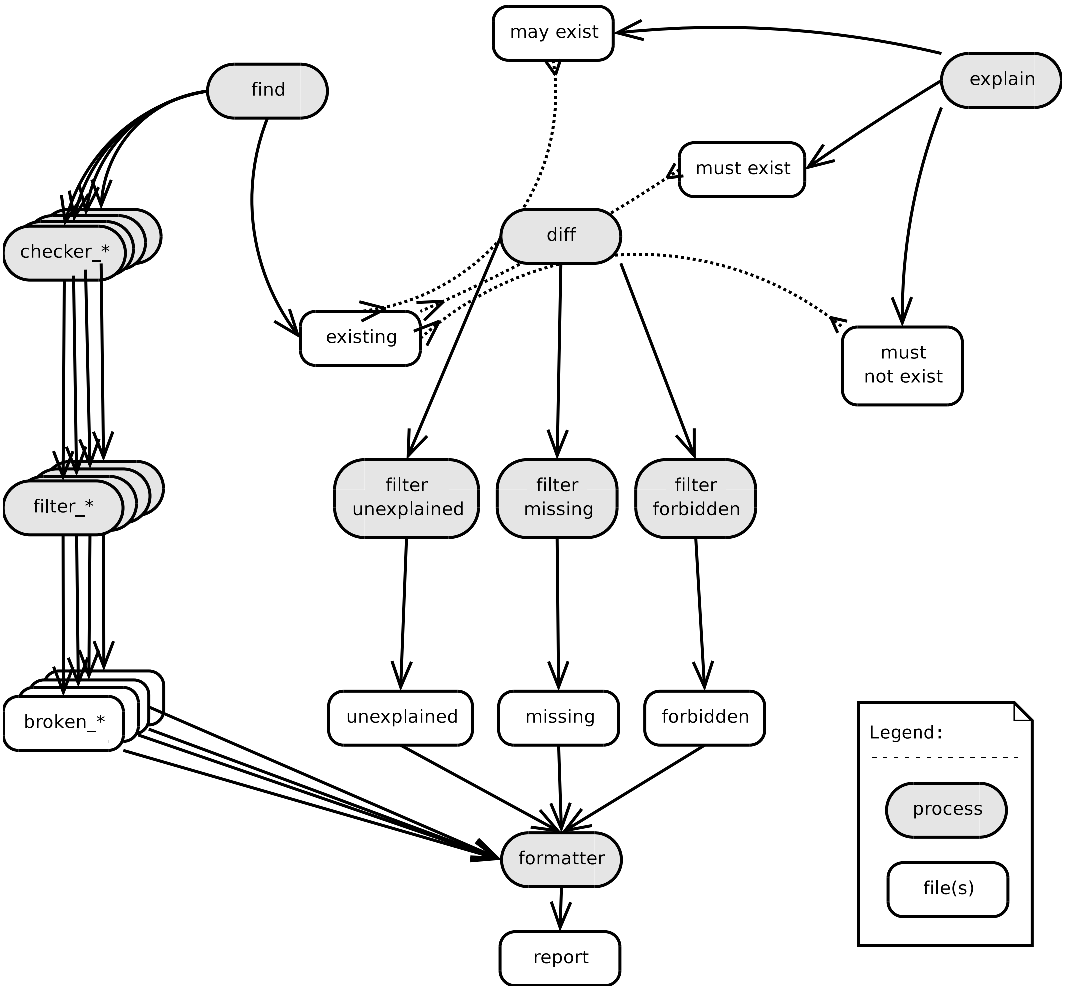

Figure 1: Cruft design

- any explain script will be run only if *any* of the following is true:
  - a package with the same name as the file is currently installed
  - the name of the file is in `UPPER-CASE`.

  In other words, only the scripts which are upper-cased or named after currently installed packages are run.

Each of the scripts is run in turn, and its output from file descriptors 1, 3 and 4 is redirected to the following files in `/var/spool/cruft/`, respectively:

`expl_`*script_name* — list of files which **must** be on the system, according to *script_name*.
: This for example includes the list of files dpkg knows about, diversions, `lost+found` directories and so on.

`mayx_`*script_name* — list of files which **may** be on the system, according to *script_name*.
: If a file is on this list, it means that it is OK for it to be present, but it is not a problem if the file is missing, either. Examples are some log files, spool and cache files.

`msnt_`*script_name* — list of files which **must not** be on the system, according to *script_name*.
: An example could be locale support files which were removed by "localepurge".

3.3 Calculating differences

After producing the lists, the list of existing files is compared to each of the three other ones using a modified "mergesort" algorithm. This produces three new sets of file lists:

unexplained that is files which are on the system, but they are not mentioned by neither the "may" nor the "must" list.
: It is placed into `/var/spool/cruft/unex_*`

missing that is files which are in the "must" list but are not in the "actually present" list.
: It is placed into `/var/spool/cruft/miss_*`

forbidden that is files which are in the "must not" list, and are in the "actually present" list.
: It is placed into `/var/spool/cruft/frbn_*`

3.4 Filtering

In the next stage, files containing the unexplained, missing and forbidden files are filtered. Having such "filtering" stage has the following advantages:

- gives the system administrator the ability to ignore some cruft which they accept or over which they have no control (for example a result of a bug in package)
- gives cruft some flexibility that is useful in case of conflicting rules or buggy packages. Examples include packages removing, moving or renaming files belonging to other packages, broken symlinks (pointing to files from packages which are not installed) and other.
- makes it possible for cruft to use the dpkg "extrafiles" database, when it will be implemented

Each set of lists of a certain type (any of: `frbn miss unex`) is filtered separately. To filter a set of files of a given *TYPE* the following filter files will be used:

- `/etc/cruft/filters-`*TYPE*`/*`
- `/etc/cruft/filters/*`
- `/var/lib/dpkg/info/*.extrafiles`
- `/usr/lib/cruft/filters-`*TYPE*`/*`
- `/usr/lib/cruft/filters/*`

However, if a filter file called *foo* (or *foo.extrafiles*) exists in more than one of the above locations, only the first one in this list will be considered. This makes it possible to easily override a filter locally. Moreover, the same `UPPER-CASE` naming convention applies as with explain scripts (that is, a filter file which has any lower case letters will only be run if a package with the same name is present in the system).

3.5 General file checking

As noted in section 3.1, "find" outputs lists of files of a certain type. Each of these lists is inspected at this stage by a "checker" script, which determines that these files adhere to some policy, and prints on its output those which are somehow incorrect. After filtering the output (using a mechanism similar to the filtering of unexplained, missing and forbidden files, but with a completely separate set of filter files) the incorrect files are included in the report.

At the time of writing, only a list of symlinks is created, which is inspected to detect broken symlinks, which are then filtered to get rid of some "special cases" like `/dev/stdin`. However it would be very easy to add checks for directories, sockets, device files or other types of files, if need be.

3.6 Reporting

Each of the filtered unexplained, missing and forbidden file lists, as well as the filtered output from the "general file checking" stage is then slightly formatted, indented and concatenated into a plain-text report, which is printed to standard output, to a file, or sent by email. An example looks like this:

```
cruft report: Sat May  5 10:55:17 UTC 2007

---- unexplained: / ----
        /tmp/cmd
        /tmp/report
        /var/log/bootstrap.log

end.
```

There is also a Midnight Commander vfs script [5], which lets one browse the report like a virtual filesystem. This is especially useful in case of large reports, which take a long time to scroll up and down in plain text format.

3.7 How to make cruft not report some file

There are several ways to do this:

- use `--ignore` which makes cruft ignore whole directory trees by not entering them at all, which speeds it up considerably. This is useful for large directory trees which local administrator is not interested in, like `/home`. An example:

```
computer:~# cruft -m root --ignore /usr/local
```

- create a filter file, which contains patterns of files which are not to be reported. This is a little more flexible, but requires cruft to traverse the directory tree, which takes some time. An example could be:

```
computer:~# echo '/usr/local/**' > /etc/cruft/filters-unex/USR_LOCAL
```

- create an 'explanation' script which prints to FD3 the names of all files which are not to be reported. This is the most flexible way, since you can decide at run time which files should be there, and which not. On the other side, it is less efficient, since it usually requires cruft to traverse the directory tree twice. An example could be:

```
computer:~# cat >/etc/cruft/explain/USR_LOCAL
#!/bin/sh
find /usr/local >&3
^D
computer:~# chmod 755 /etc/cruft/explain/USR_LOCAL
```

4 Conclusions

Cruft's state has improved over the last couple of years. Many small but annoying bugs, which made cruft inconvenient to use, are now fixed. As a result of recent refactoring most of the codebase is now cleaned up and much easier to understand.

An interesting fact which revealed itself during development of cruft is that apart from being useful for finding user- and administrator-generated cruft, it can be used as a tool for finding packaging bugs. One such bug that the author encountered, was a problem with treatment of alternatives in the maintainer scripts, which caused wrong symlinks to be created. A cruft filter or explain script can be treated as an additional record of what files should exist after package installation. Cruft allows for easy and automated reconciliation of the intended and actual behaviour, and is a useful form of testing a package.

Even though "cruft" is becoming more and more useful, the work is far from being complete:

- some refactoring is still needed, for example splitting several parts of the functionality (like checker and formatter) off from the main script, as well as some more cleanups,

- improving the pattern-matching engine, which is neither easy to understand, nor efficient or safe to use,

- updating and improving current filters and explain scripts, as well as adding *lots* of new ones — current coverage is partial and outdated; some way to keep track of which files need updating would also be nice, until they are properly migrated to their respective packages,

- adding more scripts and filters will probably reveal some new problems or inefficiencies in cruft, which will mean more functionality changes,

- at some point, say, when cruft supports a `Priority: standard` or a basic desktop or server installation, and no more functionality changes are needed to support more packages, it may be a good idea to start moving the explain scripts and filters to their respective packages, so that they can be maintained as close to the code they describe as possible,

- when the filter/explain files are widely adopted, appropriate statements should be added to the Debian policy.

References

[1] Debian-Policy. *Conffiles and Configuration files (again).* `http://lists.debian.org/debian-policy/1998/04/msg00032.html`, 1998.

[2] Debian-Policy. *PROPOSAL: Extrafiles (was Re: Conffiles...).* `http://lists.debian.org/debian-policy/1998/04/msg00089.html`, 1998.

[3] Debian-Devel. *Cruft update.* `http://lists.debian.org/debian-devel/2001/09/msg00647.html`, 2001.

[4] Debian-Devel. *The package "cruft" needs help.* `http://lists.debian.org/debian-devel/2004/02/msg01915.html`, 2004.

[5] Marcin Owsiany. *Cruftfs wishlist bug.* `http://bugs.debian.org/407475`, 2007.

[6] *Cruft project on Alioth.* `http://alioth.debian.org/projects/cruft/`