Command-name pollution (and how to avoid it)

Andrew Findlay

Brunel University
Uxbridge, UK

Andrew.Findlay@brunel.ac.uk

ABSTRACT

Problems particular to large UNIX-based computer services are discussed, with particular emphasis on the management of applications packages. A way to avoid pollution of the command-name space is proposed and a minimal implementation is detailed.

1. The Problem(s)

Large computer services have to support a great number of application packages. These take up a lot of disk space and provide a lot of commands. The traditional UNIX */usr/local* approach to adding software soon becomes strained: the filesystem fills up, and sooner or later packages begin competing for command names. Graphics packages are probably the worst problem here, as they *all* seem to have a command called *plot!*

To make matters worse, on distributed systems it is necessary to replicate critical software so *all* the */usr/local* filesystems burst.

2. Divide and Conquer

The approach used at Brunel is based on keeping all parts of a package together in a common hierachy:

/...../applicationname/bin
/...../applicationname/lib
/...../applicationname/man
/...../applicationname/src

By including the version number in the application name, it is possible to do clever things like phased introduction of new versions.

This structure makes replication and management easy, but the poor user now has to have an immense and obscure search path. The dead-NFS-server problem rears its ugly head, and we have to think again...

3. Automounters to the Rescue!

The first step in getting all this under control is to build an idealised file structure to contain the packages. We call ours */vol/PACK* and all packages are considered to be subdirectories of this master directory:

/vol/PACK/pafec_6.2/...
/vol/PACK/matlab_3.5e/...
/vol/PACK/GCC_1.37/...

We even put unbundled languages like Sun's Fortran into this structure.

The */vol/PACK* structure is implemented using *amd* so that each package can be stored on the most appropriate fileserver(s). *Amd* also handles replication and versions for different machine types.

4. But what about $PATH ??

With all these packages (we currently have about 20 in the scheme and more are being adapted every week), the users' $PATH could get immense if everything was listed. Long paths lead to poor performance especially if many of the components are automounted, so something must be done.

We have two solutions to this problem, and we use both:

4.1. The devious shellscript

For a package like GCC which only has one command and clashes are not likely we put a script in /usr/local/bin:

```
#!/bin/sh
exec /vol/PACK/GCC/bin/gcc $*
```

The script is small, so /usr/local/bin does not grow out of control. It references an automounted filesystem with no danger of trying to mount the real files unless they are needed.

4.2. The 'use' command

This one is *really* devious! 'use' adds elements to $PATH on request. The important point is that new components go on the *front* of the path. This means that commands belonging to the selected package will override those belonging to other packages or to the system itself. Thus, the command 'use pafec' would add */vol/PACK/pafec/bin* to the front of $PATH. New versions can be introduced for testing with commands like 'use pafec-97'.

The 'use' command can do other things e.g. setup commands can be run and environment variables set.

'use' is implemented as a one-line *csh* alias plus a couple of shell scripts which access a YP (NIS) database. The alias is shown wrapped to fit on the page. Note that it was originally planned to call the command 'attach' so that 'detach' could reverse the effect, but it was decided that too many people would misspell 'attach'!

```
alias use 'set path=( `/usr/local/lib/attach/attachinfo \!:1` ${path} );
        eval `/usr/local/lib/attach/attachcommand \!:1``
```

The shell scripts are simple:

```
#!/bin/sh
#
# attachcommand
(echo `ypmatch ${1}.attachcommand attach`) 2>/dev/null

#!/bin/sh
#
# attachinfo - part of the attach command
(ypmatch ${1}.path attach) 2>/dev/null
```

There are problems with this simple implementation. In particular, no error messages are printed if the user tries to 'use' a non-existent package! So far, there is no 'unuse' command either.

5. Summary

The package concept is working at Brunel and no major problems have been hit so far. The implementation needs improving. Automounters are indispensable to the management of big distributed systems.

6. Acknowledgements

The package/use concept was developed at a meeting of the Brunel's system managers, with input from Tony Begg, Linda Birmingham, and Shila Mehta