

CONTRIBUTED DOCUMENTATION

This document is NOT supported by Computing Services.
DO NOT contact the Help Center with questions on this document.

Using Depot

This document provides a quick overview of the major functionality of Depot. More details can be obtained by reading the appropriate man pages. This document should be generally applicable but there is a bias towards the Andrew environment.

Moving Files to the Local Drive

There are two main benefits in moving applications to the local drive of a workstation:

- Faster access. It's faster to run it from the local disk than to get it from AFS.
- Improved availability. If the file servers are down, files may still be accessed.

Copying files

The commands used to copy the files are the "target.installmethod copy" and "collection.installmethod copy" depot.pref options. If every file in a collection is to be copied, use the collection.installmethod qualifier.

For example, to copy all of AFS, X11R5 and Motif to the local disk, use:
collection.installmethod copy afs,x11r5,motif

To copy a single file, such as /usr/local/bin/telnet, use:
target.installmethod copy bin/telnet

To copy all of lib, use:
target.installmethod copy lib

Customizing the copying

The target.installmethod link command can be used to customize the copying. The installmethod from target.installmethod will be used even if collection.installmethod specifies another.

For example, to link all the man pages, use:
target.installmethod link man

This would link everything in the man directory, even if the "collection.installmethod copy" option were used for a collection that has man pages.

There can be multiple target.installmethod copy and target.installmethod link occurrences in the depot.pref.proto. For example:

```
target.installmethod link man,help,lib
target.installmethod copy lib/X11,bin
target.installmethod copy lib/gnu-emacs
```

Notice that the `installmethod copy lib/X11` and `lib/gnu-emacs` will cause all files in `lib/X11` and `lib/gnu-emacs` to be copied while everything else will be linked.

Testing New Applications

To test new software, edit the `depot.pref.proto` so that there is a "path" pointer to the dest volume. For example, to test the new `wsadmin` version 040 on a `pmax`, add the following to the `package.proto`:

```
path wsadmin /afs/andrew.cmu.edu/system/dest/pmax_ul4/local/wsadmin/040
```

After making that change, run `/etc/dodepot` again.

Of course, you can also use the "path" option to define new software and point it to where it would exist. In the following example, the collection named `new_software` would be added to the environment.

```
path new_software /path/to/the/software
```

Installing Software on the Local Disk

1. Create the directory structures
Create a directory, under `/usr/local/depot`, using a reasonable name, based on the software being installed. This should be a unique name and should not be the same as something already on Andrew. If such a conflict occurs, the collection on the local disk will replace the old collection of the same name. The file `/usr/local/depot/collection_list` lists the names of all the Andrew collections.
Note: Further examples will refer to collections as `{software}`.
2. Create the subdirectories: `/usr/local/depot/{software}/(bin,lib,man,etc)`. If the software creates its own `lib`, `bin`, or `man` directories, just make sure that they are in `/usr/local/depot/{software}`.
3. Install the software.
Choose whichever method is most convenient and install the software into `/usr/local/depot/{software}/(bin,lib,man,etc)`. Remember if the file is to appear in the "foo" directory in `/usr/local`, install the file into `/usr/local/depot/{software}/foo`.
4. Run Depot. This can take up to 40 minutes. Use the following command to run Depot by hand:

```
# /etc/dodepot /usr/local
```

As it is now, Depot will build `/usr/local` containing pointers to Andrew software, and will also install local software onto the drive from the Depot directory.

Handling Conflicts

When running `/etc/dodepot`, Depot may report that there is a conflict between one of the collections that is being installed and another collection. If a file that has just been installed has the same name and exists in the same directory as another application, a conflict has occurred.

At this time, there are two choices:

- Override what is in local.
Do this by adding the following line to the depot.pref.proto but first be sure that it is okay to override the file.
override {software} {collection to be overridden}
- Move the offending file into another directory.
For example, instead of putting all the files in lib, they could be put into lib/{software}. After this is done, "unlock" the environment by typing:
depot -U -T /usr/local
and re-run
/etc/dodepot /usr/local

Ignoring Directories and Preference Options

There may be instances when the maintainer of the target environment wishes to discard any customizations made by remote environment maintainers without having to specify a customization rule of their own. While it is unlikely to occur, the possibility exists and should be specified.

To suppress collections, list the collections after the ignore preference option in the depot.pref.proto.

The depot.pref.proto and depot.pref preference options can be suppressed. In general, string options are the only options that can be suppressed. The string :IGNORE: will be reserved to specify that an option is to be ignored. For preference options that are not a STRINGSET or STRINGARRAY, then the text :IGNORE: will be sufficient to do the ignore. For example:

```
command foo :IGNORE:
```

will ignore all occurrences of the command labeled foo. In the case of STRINGSETS or STRINGARRAYs, give the value to be ignored after the :IGNORE: string. An asterisk (*) can be used as a wild card to specify all collections.

For example, the following command results in no overrides being included from remote environments.

```
override bar :IGNORE:*
```

Similarly,

```
override bar :IGNORE:foo
```

will result only in

```
override bar foo
```

being discarded.

Note: :IGNORE: processing is NOT done for entries that exist in the depot.pref.proto. The :IGNORE: specification is only used to prevent propagation of remote preference information into the target environment.

The global depot.conf in remote environments can only be ignored with the following entry in the depot.conf.proto:

COLLECTIONNAME :IGNORE:

Note: This is an all or nothing scenario. That is, this :IGNORE: discards all remote global depot.conf entries from COLLECTIONNAME. The only possible depot.conf entries for COLLECTIONNAME would be from the depot.conf.proto in the target environment.

Using Filters

As Depot migrates a file to the local disk, Depot can run the file through a filter. Filters are specified in the depot.pref.proto in a two part process - creating a mapping from a filterlabel to the filter command, and applying the filter to some part of the tree.

The first step creates a filterlabel, linking it to some executable command. The syntax for this is:

```
filter <filterlabel> <command> [<arguments>]
```

Arguments to the command are separated by spaces; a space may be escaped by preceding it with a '\' character. For programs that need to know the name of the file they're executing on, the destination and source file paths are substituted for the symbols "%from" and "%to". Standard input is directed from the input file, standard output is directed to the output file, and standard error is left unchanged.

To apply a filter to part of the tree, use a target.installmethod preference:

```
target.installmethod <filterlabel> <tree1>[,<tree2>...]
```

This tells Depot to use the filter labeled with filterlabel to process all files under tree1 (and tree2, and tree3...).

For example, to apply /bin/jive to all of the files under the man directory, add the following lines to depot.pref.proto:

```
filter jive /bin/jive
target.installmethod jive man
```

Another example is to add a caveat (/etc/caveat) to the beginning and end of all of the files under the help and info directories. To do this, add the following lines:

```
filter caveatize /bin/cat /etc/caveat - /etc/caveat
target.installmethod caveatize help,info
```

The filter in the last example could also be specified as:

```
filter caveatize /bin/cat /etc/caveat %from /etc/caveat
```

(Both work because the "-" tells cat to take input from its standard input, which is attached to the same file that %from expands to).

Running Commands

Another way Depot can run a program is via the command syntax. The command system is not frequently used and as such probably requires some additional development.

The first part of the command system is to specify a command label. The command label simply associates a command with a tag. For example:

```
command update_fonts mkfontdir /usr/local/lib/X11/fonts
```

After defining a label, you can now run it in two ways.

First, there is the `target.command` option. This preference option says to run the given command label whenever anything is updated in the given directory path. So, if one has

```
target.command update_fonts lib/fonts/X11
```

then the command associated with the 'update_fonts' label will be run whenever any application installs anything under 'lib/fonts/X11'.

The second way of specifying a command is via the `depot.conf` `~command` syntax. The `depot.conf` `~command` allows a collection to install files generated by a program. So, if one had a command label 'gen_what' that built the whatis database, one could have the following `depot.conf` entry:

```
~command gen_what lib/what
```

This lets Depot know that the file lib/what is being installed into the environment.

Two problems with the command system are:

1. When `target.command` calls a command label, one should be able to pass the path to the command label. Thus, in the above example, the command label for `update_fonts` should be something like:

```
command update_fonts mkfontdir %path
```

where `%path` gets replaced by the appropriate paths in the `target.command`. This, however, can not be done currently because it has not yet been implemented. It will be addressed in the future.

2. The only way to specify files that are installed by commands is via the `depot.conf`. It can be done in the global `depot.conf` but the specification must map to a valid collection. There should be a way of saying something like:

```
target.commandinstalledfiles <label> <filelist>
```

This has not yet been implemented.

Appendix: Files in the Depot Directory

All files ending in `.BAK` contain the previous version of the file.

depot.pref.proto

Edit this file when using Depot with `dpp`. This is the "prototype" file where changes and customizations should be put. See the man page for `depot.pref(5)` on what can be added here.

depot.pref

This file is automatically generated by `dpp`. Do not edit this file. This file is created from the `depot.pref.proto` and all the `depot.prefs` in the environments pointed at by the `searchpath` *. If `dpp` is used, this file is automatically generated by `dpp` from the `depot.pref.proto`.

collection_list

This file is automatically generated by dpp. This file lists all the collections that exist in the environment.

depot.db

This file is automatically generated by Depot. This contains the "database" information for Depot.

depot.lock

This file is generated by Depot when Depot is in use. It contains the userID of the person running Depot. In the case of an error, the depot.lock is not removed. One must manually remove it with "depot -U" before Depot can be run again.

depot.conf.proto

This is an optional file that allows the user to tailor the depot.conf. It is only used by local disk Depot.

depot.conf

This file is generated by dpp. The depot.conf can be located in the Depot directory or in the top level of a collection. The depot.conf located in the Depot directory is the global depot.conf. This file allows a gatekeeper to modify what a collection maintainer does. If there is a conflict between a collection depot.conf and the global depot.conf, the entry in the global depot.conf will be used. If dpp is used, this file is automatically generated by dpp from the depot.conf.proto.