Xhier Man Pages

Compiled by: Gary Ridley, MFCF
Date: Friday, March 18, 2005

hier - software maintenance/distribution package.

ESCRIPTION

The xhier package is a collection of tools and conventions intended to assist in the maintenance and distribution of software. The primary goals are to:

- simplify distribution of updates.
- simplify installation (on multiple architectures).
- facilitate sharing software via remote file systems.
- minimize changes and additions to the file structure distributed with operating systems.

The documentation consists of man pages with names of the form xhier-Topic for general discussion, and names usually of the form xh-Name for details about commands. A few commands and file descriptions don't begin with "xh-".

The tools and conventions are still under development, and there are many details that have not been resolved. Comments are welcome.

EE ALSO

xhier-users-guide(7) - what a user of xhier packaged software should know.
xhier-conventions(7) - summary of the software organization conventions.
xhier-tools(7) - summary of the xhier tools.
xhier-config(7) - configuration of the xhier package.
xhier-howto(7) - quick introduction to installing a new package.

---

xhier-users-guide - how to use Xhier packaged software

DESCRIPTION

Some software on UNIX systems at the University of Waterloo is distributed and maintained by a facility called Xhier (the origin of the name is of historical significance only).

This facility makes it possible to install and maintain current versions of software and automatically distribute them to other UNIX systems on campus that have Xhier installed. This guide will provide you with the information you will need in order to understand and make use of the unique features of systems using Xhier at the University.

Gaining Access to Software Distributed by Xhier

The most critical thing you need to know about Xhier is that software is not placed with the software that comes with the machine. That means that you may have to do something special (usually just once) in order to have access to the extra Xhier-supplied software.

Gaining Access to Commands

The UNIX PATH environment variable must be set to have access to Xhier-supplied commands. This variable is normally set by a startup script (e.g. .cshrc) so that it is done automatically when you log in. On many systems, new accounts are created with this already done.

The command showpath is provided to simplify the definition of the PATH variable. Issued as a separate command:

```
showpath standard
```

displays a list of search directories that contain both the additional software delivered by Xhier and the original software installed on your machine. By enclosing this command in grave quotes (`...`), the output can be used to define the PATH variable. For example in the C-shell:

```
setenv  PATH  `/bin/showpath standard`
```

In the output of showpath, the paths to Xhier-supplied commands come before the paths to the original commands on the machine so that Xhier-supplied commands take precedence and are found first.

Other showpath command parameters may be used to include specific paths (e.g. /usr/local/bin, $HOME/bin, ".") or Xhier package names (packages are described later) not in the standard Xhier search directory (e.g. gnu). For example:

```
setenv PATH `/bin/showpath $HOME/bin standard /usr/local/bin gnu .`
```

Virtually all of the software on your machine will be included by using the name standard in the showpath command line. The usual exceptions are packages that contain commands that conflict with other commands in a misleading or dangerous way.

Gaining Access to Documentation ("man pages")

The man command on Unix systems is usually the means for reading online documentation (called man pages). Since not all man commands have the means to examine man pages in arbitrary locations, a man command is supplied that can do so. Thus once one's PATH is set as given above, the resulting man command will be able to read the additional man pages, i.e. nothing additional need be done.

If you add packages to your PATH command search rules (usually not necessary), you must add the corresponding packages to your MANPATH man page search rules so that the man command can find the man pages. You do this in a manner similar to the setting of PATH, for example:

```
setenv PATH    `/bin/showpath $HOME/bin standard /usr/local/bin gnu .`
setenv MANPATH `/bin/showpath class=man standard gnu`
```

The -t option of the man command is useful if you're interested in reading man pages for packages that aren't in your search rules. E.g.

```
man  -t cnews  -k cnews
```

will show the names of all of the man pages in the cnews package, even if it's not in your search rules.

Gaining Access to Libraries and Include Files

In addition to executables, you may need to use libraries and include files (while doing program development). Most Unix systems don't implement user-settable search rules for libraries and include files (at least not in a way in which the rules can be set just once). As a compromise, Xhier makes libraries and include files appear in appropriate system locations. This is done in such a way as to minimize conflicts.

All systems provide a location for additional libraries (almost always in /usr/local/lib). To make library access easy, all of the libraries in the installed packages are made to appear in /usr/local/lib (or its equivalent). Thus the usual -l option can be used to reference the additional libraries, as in

```
cc MyXStuff.c -lX11
```

Unfortunately the system searches vendor-supplied libraries first, so you cannot always use the -l option to get the Xhier-delivered libraries. The -l option will always find the vendor libraries first. Some software (such as X11) may be available both in Xhier and in original, vendor-distributed form on a machine. In this case, you must substitute instead the full pathname of the Xhier-supplied library. (See below for how to find out what this full pathname might be.)

The combined include files from a large collection of software will likely have some conflicting names. To avoid this problem, include files are made to appear in /usr/include/PackageName/ rather than just in /usr/include. This means that (for example) this:

```
#include <gnu/gdbm.h>
```

is required rather than this:

```
#include <gdbm.h>
```

when referencing include files. Some software packages already use a similar convention (e.g. the X11 software), and in such cases the name adopted by the software is used. For example, even though the Xhier package name may be x11, the include files can still be referenced as

<X11/filename.h>.

Sometimes it is necessary to reference the full pathname of a library or include file (e.g. in Makefiles). The libraries associated with a package called PackageName are found in the directory /software/PackageName/lib/, and the include files are found in the directory /software/PackageName/include/. For example, the full pathname of the X11 library is /software/x11/lib/libX11.a.

Gaining Access to Termcaps

A termcap is a configuration file that describes characteristics of various terminals. Most operating systems have a single vendor-supplied termcap file. Additional terminal characteristics (useful for terminals found here at Waterloo) are provided in the termcap package. Versions of the common commands that use termcap are provided as well. These commands will automatically use termcap additions contained in the termcap package, so in most cases you don't need to do anything special to use them.

Some software will use the TERMPATH environment variable to search for separate termcap files in different locations. Although it's virtually never necessary (and most software doesn't use it), this variable can be set using the showpath command. For example, to define all three search paths for package `xxxx':

```
setenv PATH     `/bin/showpath class=command xxxx standard`
setenv MANPATH  `/bin/showpath class=man     xxxx standard`
setenv TERMPATH `/bin/showpath class=term    xxxx standard`
```

Finding Out What Software is Available

A large collection of software is available for distribution via Xhier. This includes: licensed software such as DISSPLA, NAG, MAPLE, MATLAB; freely distributable software such as TeX/LaTeX, X11, gnu; and software related to the operating system, such as the backup package called dumpster.

For a complete list of the software packages available (via Xhier) on campus, issue the command:

```
man software
```

For the packages that are available via the standard search rules on your machine, use the command

```
xh-packages -p
```

The Organization of Software into Packages

It's usually not necessary to know the filesystem organization of packaged software, but it is sometimes useful. The following summary should be adequate for most purposes. Details can be found in man xhier-package-structure and man xhier-conventions.

Software is organized into packages, which are collections of programs, data, documentation and examples related to a specific set of software. This makes it possible for administrators to pick and choose what software is to be installed on a particular machine. Software packages are organized under the directory /software, with one subdirectory for each package. For example:

```
lc /software
Directories:
 .admin       TeX          X11          X11R3
 accounting   accounts     agrep        archie
 awkcc        backup       bash         batch
```

The directory for each package contains subdirectories which contain all of the files required for the installation, maintenance and use of the package. Some of the more interesting directories are:

bin user commands provided by the package.

data data files required by the package.

doc documentation files (other than man pages).

man all man pages associated with the package.

include directories of include files.

lib libraries used by, or supplied by, the package.

A complete description of the sub-directories can be found in man xhier-package-structure.

Complications

The files that constitute the software are organized in a way that simplifies distribution and the use of remote file systems. An unfortunate side-effect of this is that the structure of a package may contain many symbolic links. This is generally of no concern for the users of a package, but sometimes elicits questions, e.g. what are those strange names involving /.software that the which command reports. Such questions (for the curious) are best answered by consulting reference documentation (e.g. starting with man - xhier-conventions).

A side-effect of the use of symbolic links in the structure is that exploring the structure of a package by changing the current working directory can result in confusion when using relative paths involving "..".

Problems

Updates to software are distributed from machine to machine, so there can be delays between the time new or updated software is announced and the time that it arrives on the machine you use.

The time required for software distribution also means that some machines may be running slightly different versions of software than others.

Not all systems have adequate disk space to store documentation (man pages), and thus make use of man pages on other systems (the version of the man command supplied with Xhier can do this). Therefore the possibility also exists that man pages will not be synchronized with the version of a software package running on a particular system.

Finding Out More About Xhier

An extensive collection of man pages have been written describing Xhier. While most of these pages are only of interest to systems administrators, the following man pages may be of interest to you.

Showpath

a complete description of the showpath command used to set PATH variables.

xh-packages

a command that produces various lists of packages.

xhier-package-structure

a description of how files related to a specific package are organized.

---

xhier-conventions - software organization conventions

DESCRIPTION

The following is a description of the organization to be used when installing software onto an existing operating system. The operating system organization is determined by its supplier.

1 - Organization of Installed Software

1.1 - The Simplest View

A package is a collection of related software. Software is grouped in packages because of licensing restrictions, disk space limitations, multiple software vendors, and multiple versions of a package.

For every package named package_name there is an installation directory named

/software/package_name

which (appears to) contain all of the files necessary for the operation of the package (e.g. including documentation and configuration files, but typically not program source files). All references to files in a package (with the exceptions detailed below) use pathnames of the above form.

It is expected that installation directories will typically contain bin and man directories. Details are given in man xhier-package-structure.

Software is installed in such a way as to make the local environment (consisting of the supplied operating system, additional software packages, and local additions to both) the default environment. Details are given in section 1.10.

1.2 - A Complication

Most if not all remote filesystems require that entire directories be made accessible. So in order to facilitate access to packages via remote filesystems (at the expense of a more complicated structure), files in installation directories that can't be shared across all hardware and software architectures are placed in separate (parallel) directory structures depending upon the extent to which they can be shared. To prevent this separation from complicating access to a package's files by requiring knowledge of which parallel structure to examine for a given file, symbolic links to the non-shared files are created in the installation direc-

tories so that all of the files in a package are still accessible via the installation directory. Since symbolic links are themselves files, the links in the installation directories must be shareable (i.e. the same) across all architectures. Thus paths of the form

    /software/package_name/path

may be symlinks to one of the following, depending upon the type of the file:

/.software/arch/package_name/path
    architecture specific files (e.g. binaries and libraries). Typically /software/package_name/bin will be a symbolic link to /.software/arch/package_name/bin.

/.software/arch_architecture_type/package_name/path
    as above, but for an architecture of type architecture_type. This is usually found only on servers providing for multiple client architectures.

/.software/admin/package_name/path
    files that are the same on all machines in an "administration". These are typically defaults for configuration files.

/.software/local/package_name/path
    (potentially) local files. In general, a local file is any file that isn't shareable everywhere and isn't architecture specific. Thus a local file may in fact be identical on multiple machines, due to common administration of the machines.

/.software/spool/package_name/path
    spool files (a special case of local files). Such files are usually transient, with unpredictable sizes.

/.software/regional/package_name/path
    files common between a client and server that have the same user community and same set of user files. These are a special case of local files.

For completeness,

/.software/share
    exists for shared files, and typically /software is just a symbolic link to it.

/.software is an implementation detail, and is not intended as a place for files not visible in /software. Thus for every /.software/type/path, there is a corresponding /software/path.

1.3 - Where Files Really Are

The appropriate locations for installation of files depends upon the operating system in use, and the particular partitioning chosen for a machine. Thus the names used above to reference the various classes of files:

/.software/type

are at least architecture specific and often local symbolic links to appropriate locations, or mount points for partitions. The general rule used in determining the value of the symlinks is to create a .software directory in the directory where the operating system would put the kind of file in question, keeping the name .software in the path to hint that it's part of the /.software structure. E.g. if /var/spool was an appropriate location for spool files then
/.software/spool -> /var/spool/.software/spool

In addition, should partitioning preclude all of /.software/type from residing in a single partition,
/.software/type/package_name
may be symlinks to wherever there is space. A typical scenario is:

/.software/type/package_name -> /fsysN/.software/type/package_name

for a subset of package_name that happen to be quite large. Often it's /.software/arch that provides the greatest opportunity to move the least number of package parts to balance partition usage.

Because packages will sometimes be made accessible using a remote file system, symbolic links to local, spool, and regional files must be absolute. Otherwise remote file system references to them could result in a reference to the remote versions of the files.

Symbolic links (in /software/) on a server should only reference paths that will be accessible on both the server and clients. Because there is (as yet) no guarantee that parts of a server's filesystem outside of /software and /.software/* are mounted (on clients in the same relative locations that they are on the server), this seems to imply that symbolic links that use /software or /.software must be absolute.

1.4 - Summary of Reserved Pathnames

A summary of installation locations (described in greater detail later) follows.

1.4.1 - New Names

/software/
default location of installation directories.

/software/package_name/
the installation directory for the package named package_name. Contains shareable files together with symbolic links to all other files associated with package_name. Details of the structure can be found in man xhier-package-structure.

/software/xhier/
a package of tools useful in maintaining this structure.

/software/.admin/
a directory containing files used in the implementation of this software organization.

bins/
directories of (absolute symbolic) links to binaries in packages. These exist solely as a means of avoiding long search rules (which shells unfortunately do not handle well).

bin/ (absolute symbolic) links to binaries in installed (current versions of) packages.

maintenance/
(absolute symbolic) links to maintenance programs in installed (current versions of) packages.

servers/
(absolute symbolic) links to servers in installed (current versions of) packages.

/.software/arch/
the name used (by symbolic links) to reference architecture dependent files. It is usually a symbolic link, with a value that is architecture dependent.

/.software/arch/package_name/
the architecture directory for package_name, containing architecture dependent files associated with /software/package_name, referenced only by absolute symbolic links from the installation directory.

/.software/arch_architecture_type/
As /.software/arch/, but with architecture specific files for a hardware/software architecture of type architecture_type. See man xhier-arch-types for the possible values of architecture_type. Such directories are typically only found on servers that must serve architectures that differ from their own.

/.software/local/
the name used (by symbolic links) to reference local files. It is usually a symbolic link, with a value that is architecture dependent.

/.software/local/package_name/
the local directory for package_name, containing host specific files (e.g. some configuration files) associated with /software/package_name, The files are referenced only by absolute symbolic links from the installation directory.

/.software/spool/
the name used (by symbolic links) to reference spool files. It is usually a symbolic link, with a value that is architecture dependent.

/.software/spool/package_name/
the spool directory for package_name, containing spool files associated with /software/package_name, The files are referenced only by absolute symbolic links from the installation directory.

/.software/regional/
the name used (by symbolic links) to reference files that are common between a client and its server. It is usually a symbolic link, with a value that is architecture dependent.

/.software/regional/package_name/
the regional directory for package_name, containing files /software/package_name that are common between a client and a server that have the same user community and same set of user files. These are a special case of local files. The files are referenced only by absolute symbolic links from the installation directory.

/usr/source/package_name/
source for the named package.

/usr/source/TEMP-CACHE/package_name/
source for the named package copied from its maintenance machine while being rebuilt here.

/vendor/path
where the originals of path are placed if it proves necessary to replace or delete them, so that the change or deletion can be undone if desired. This is usually a symbolic link (to avoid filling up the root partition).

/xhbin
is a real directory under the root partition in which stand-alone programs (e.g. diagnostic programs needed when single-user) can be found. It also contains programs that must have short absolute pathnames; this is needed for scripts that use the "#!" invocation for which there is a limit of usually 32 characters on the interpreter name (and arguments).

1.4.2 - Additions to Existing Names

The pathnames used in the following are those used on most systems. The pathnames on some architectures are slightly different.

Some packages have multiple versions, only one of which is in the standard search rules at a time, and which is always referred to with the same name (i.e. without a version number). In the following, package_basename refers to the name of the package without a version number.

/usr/lib/library.a
is a symbolic link to
/software/package_name/lib/library.a. Replaces a system supplied library.

/usr/local/lib/
contains symbolic links to the libraries in the lib
directory of installation directories.

/usr/include/package_basename/
is a symbolic link to
/software/package_name/include/package_basename (if
present).

/usr/include/someinclude.h/
is a symbolic link to
/software/package_name/replacements/include/someinclude.h if it
is a replacement for a system supplied include file.

1.5 - Existing Operating System Software

It is occasionally necessary to replace or delete files supplied with the operating system.

A file is replaced if that is required to augment it in some way that can't be done with search rules.

A file is removed (from its standard location) only if leaving it there would cause confusion, and there is no acceptable replacement for it. E.g. if the supplied version of a program is (dangerously) broken, and there is no replacement for it, it may be desirable to (re)move it to avoid calamity should incorrect (or no) search rules be used.

In either case, such files are first moved to

/vendor/path

where path is the original pathname of the file, if there is enough disk space available to do so.

In situations where a package and the OS could potentially share the same spool directory, it is generally best to use independent spool directories when possible. If spool directories must be shared, then it's best to have the package use the OS's spool directory, (rather than have the OS use the package's spool directory).

1.6 - Packaging Subsets of an OS

It is sometimes the case that software being installed comes from part of another operating system. Rather than have the package organized differently on the originating and target systems, it is packaged according to this standard on both the target and originating system (even if no local modifications have been made).

1.7 - Embellishing Existing Software

When a package contains additions to other packages, the additions remain in the installation directory of the (embellishing) package. If necessary, symbolic links (to the additions) may be placed in the packages being embellished if that is required for the use of the additions.

1.8 - Unsupported Software

A package is unsupported when requests for corrections, updates, or porting are given the lowest (or no) priority. The definition is rather vague, and is generally used to allow installation of potentially useful software of dubious quality, or software for which there are insufficient resources to provide any kind of support commitment. For example, crude prototypes may be unsupported but can serve to test the utility of an idea to be later incorporated into supported software. Popular (but essentially unmaintainable) programs may be installed as unsupported to avoid unnecessary copies.

Installation of unsupported software is discouraged. Unsupported software is usually not included in the standard search rules. It is preferable to have packages be either completely supported or completely unsupported. Documentation for unsupported software should make it clear that no support is available.

1.9 – Documentation

All installed packages must have associated man pages (the operating system itself is not considered an installed package in this case). See man xhier-man-pages for a description of man pages common to all packages. The templates in /software/xhier,dev/data/man-page-templates provide additional details.

1.10 - Search Rules

Search rules are used (when possible) to make packages accessible. The showpath command is used to set search rules.

1.10.1 – Programs

Search rules for user commands are obtainable using the showpath command, which is always present (as a symbolic link if possible) in /bin (an exception to the rule of not installing files in system supplied directories).

Unfortunately, CSH only hashes the first eight directories in search rules, and SH doesn't hash at all. So, the result of putting at least one directory per package in the standard search rules would be a potentially unacceptable performance penalty. To avoid this, a general bin directory (called /software/.admin/bins/bin) may exist to contain links to the binaries of the current versions of all installed software packages. Thus the standard search rules may contain /software/.admin/bins/bin rather than most of /software/*/bin. The same approach applies to other bin directories (e.g. servers or maintenance).

1.10.2 – Documentation

In order that there be search rules for documentation (man pages), the standard search rules ensure that the locally modified rman implementation of the man command is used rather than the standard implementation, which doesn't use settable search rules. The documentation search rules are set in rman's MANPATH environment variable, and are obtainable using the showpath command. The directories to be searched are defined in rman's rmand.cf configuration file. The rmand.cf configuration file defines a type for each version of every package. In addition, the documentation for all packages in the standard search rules are accessible without specifying a type.

1.10.3 – Libraries

Libraries are made accessible by symbolic links in a system library directory, e.g. /usr/local/lib/libname.a or whatever is appropriate for the architecture.

1.10.4 - Include Files

Include files are made accessible by symbolic links in the package name in the the operating system's include file directory, i.e. /usr/include/package_basename. References to such include files are therefore of the form "#include <package_name/include.h>". The inclusion of the package name is to reduce name conflicts that would almost assuredly occur otherwise.

1.11 – Versions

If there are multiple (installed) versions of a package, then each of the corresponding package names includes a version number. When there is otherwise no convention for including a version number in a package name, the form package_name-version (where version is a version number) is used. If there is a single (installed) version of a package, then the version number is optional. An installation directory without a version number may be a symbolic link to the appropriate directory that does have a version number.

1.11.1 – Testing

Final software testing is performed either by installing it in its current package and inhibiting distribution while testing, or by installing it as a new version of a package.

1.11.2 - Multiple Versions of a Package

When multiple versions of a package are installed there can be a conflict in the names of the files (e.g. programs) in the packages. This complicates access to versions which aren't the current version. In the following subsections, assume that package_name is the name of the package containing the files being referenced.

1.11.2.1 – Programs

Search rules are required to access the (binaries of a) non-current version of a package. At the discretion of the installer, there can be a command with the same name as the package which applies the corresponding search rules to its command line.

1.11.2.2 – Documentation

Search rules are required to access the (documentation of a) non-current version of a package. In addition, the -t option of the (locally modified) man command may be used to access documentation for a specific version.

Package names precede lines in the whatis database for man pages associated with packages for which there are multiple versions. Man pages that describe a package (or any object that has a version number) contain the version number.

1.11.2.3 – Libraries

Using a library associated with a non-current version of a package requires an explicit reference to the directory /software/package_name/lib/, e.g. with the appropriate option to compilation command lines, such as the -L option to the cc command.

1.11.2.4 - Include Files

Using an include file associated with a non-current version of a package requires either using/changing the package's name in include file references, or, if using an include file supplied as a replacement for a system supplied include file, requires an explicit reference to the directory /software/package_name/replacements/include, e.g. with the appropriate option to compilation command lines, such as the -I option to the cc command.

1.12 - Current Exceptions and Problems

There's too much to say about what could be as opposed to what is, so we've not attempted to document that here.

2 - Source Organization

The source (if present) for a package named package_name resides in /usr/source/package_name. The substructure of non-MFCF source directories is largely unspecified, with the exception of Makefiles, described in section 2.2 below.

2.1 - Embellishing Existing Source

When a package contains additions to other packages, the additions remain in the installation directory of the (embellishing) package. Corrections and changes may be applied directly (using RCS) to the embellished package. If necessary, symbolic links (to additions) may be placed in the packages being embellished. The preferred approach is to not make additions to the other package's source, but rather to add to the resulting (installed) package.

2.2 - Source Directory Contents

Each software package has a directory in /usr/source/. The package name misc is reserved for those things that don't belong in any other package.

A source directory contains a Makefile which has (at least) the following targets.

install
: builds and installs the software into an installation directory, completing the final installation steps by invoking the install target in the Makefile in the installation directory.

clean
: deletes intermediate files used in the building and installation of the package (typically object files).

and is recommended to have

all makes everything from source (usually in the source directory), but doesn't install it.

help prints a one line description of each Makefile target.

The default action of a Makefile does not cause a destructive rebuild (i.e. it does not do a make clean).

A record of where the source was obtained from, typically a URL, is put in the file named ORIGIN.

When practical, it's useful to have one directory per component, i.e. /usr/source/package_name/ComponentName, even if it's only a symlink into the source structure. Doing so allows the source-location package to find the source for the component, making it easy to find when searching for source.

The remainder of the directory contents are organized as supplied by the manufacturer, where possible. Additional structure may be imposed by creating additional directories and symbolic links.

Large files that don't have source (e.g. some data files) can represent a problem. If they are installed by copying, then a lot of disk space is "wasted". If they are installed by moving, then the source directory is incomplete. One possible solution is to install such files by replacing them in the source directory with symbolic links to their destinations. Another possibility is for the installation to move the files and to provide gather and clean_gather targets in the Makefile. The gather target would copy (not move) back files that were moved during installation, and the clean_gather target would delete them, after verifying that the files had been installed. There does not appear to be a good solution to the problem.

Files that don't have source (e.g. some data and include files) that are both to be installed and are required for building the package can also be a problem. The easiest solution is to simply decree that the files do have source. An alternative is to install such files by replacing them in the source directory with symbolic links to their destinations.

2.3 - Source Organization for MFCF Packages

Each package directory contains one directory per object (e.g. program or library) and possibly an include or Include directory containing "include files". If both directories are present, then it is assumed that the latter is the include file directory, and that the former refers to an object (under the assumption that object names are lower case). Always using the distinguished name (i.e. Include) is the preferred approach (e.g. for ls output).

Every object has an associated Makefile, with at least the usual targets (i.e. install, clean, and the default target which makes the object without installing it).

Some objects are private, i.e. are not intended to be installed for general use, or for use by other packages, but are used in the building of the package, either as tools or as shared objects (e.g. a shared private library, or shared include files). Whether something is private is determined by what the install target in its Makefile does.

The names of all objects (i.e. including the private objects) in a package must be unique. The names of public objects are unique across all packages for most but not all objects. Current exceptions are console commands. It is recommended that private objects be named so as to minimize the chance of name conflicts should they become public, e.g. libraries should be named libpackage_name.a rather than just lib.

SEE ALSO

xhier-package-structure(7) - reserved names in an installation directory.
xhier-arch-types(7) - names of architectures.
xhier-file-types(7) - names for how a file is shared.
xhier-man-pages(7) - man page conventions.

---

xhier-tools - summary of software installation/distribution tools

Description

The tools provided in the xhier package are intended to assist in the installation, distribution and maintenance of software in an architecture independent manner. Descriptions starting with "*" rather than "-" refer to commands that are accessible only by server search rules (i.e. are usually executed by other programs rather than directly by a person). The remaining commands are accessible using maintainer search rules. The tool set is not yet complete.

Programs of Interest to Package Maintainers

Package Creation and Deletion

The following are intended to assist in creating (the contents of) a package installation directory. xh-mkpkg is usually all that's needed.

xh-installation-prog
- create a template for a package installation program.

xh-mkdir
- mkdir -f `xh-mkpath ...`.

xh-mkpath
- actual path of a file of a given type in a package.

xh-mkpkg
- create a template for a typical package.

xh-mv
- change a shared file into a non-shared file.

xh-rm-package
- delete packages from a machine.

xh-say-unavailable
- a polite replacement for missing programs.

Package Information

xh-size
- reports the size of packages.

Installation Programs

Every package has an installation program responsible for doing whatever is required to make the package functional (after the installation directory has been created). The following are sometimes useful in such programs.

xh-arch
- display the xhier architecture of the current host.

xh-binify
- make a program appear in /xhbin.

xh-check-configs
- sanity check configuration file fragments of a package.

xh-check-package-dependencies
* complain if listed packages are not here.

xh-check_userid
- check passwd and group entries for a userid.

xh-is-regional-server
* determine if the local host is a regional server.

xh-is-standalone
- query/set the standalone status of the local host.

xh-ln
- replace a system file with a link to a local version.

xh-merge-lists
* merge files on a line basis, handling 'includes'.

xh-options
- query a package's options list.

xh-recommend
* "recommend" that a package be where another already is.

xh-unln
- undo an xh-ln (restores a replaced system file).

xh-install
- invoke a package's installation/deinstallation program.

Boottime/Shutdown Programs

A package may have programs to be run a system startup, and to undo the effects of such a program. The following is sometimes useful in such programs.

xh-daemon-pid
- tries to determine the PID(s) of a daemon

Error Reporting

Error logging usually occurs from crontab entries.

xh-logger
* log and optionally mail text.

Source Maintenance

in_progress
- flag a source directory as not being installable.

xh-make
- invoke the appropriate form of make.

xh-sdist
- send to and compile on other architectures.

The Xhier Package

Package Distribution

xh-distribute
- distribute a package to other hosts

xh-dist-recurse
- do an xh-distribute down the entire tree

xh-dist2
- configure and distribute packages to a specific host

xh-sdist
- distribute source and invoke xh-make on distribution machines

xh-tar
- create a tar tape of a package

Querying the Configuration

xh-ancestors
- distribution path of a package to the current host.

xh-descendants
- targets of a package from this host .

xh-dist-hosts
- asks questions of the access-rights file,

xh-packages
- list package names according to various criterion.

xh-unmaintained
- list packages that don't have "Maintainers".

Changing the Configuration

xh-alter-requests
- alters the local requests file.

xh-lock
- locks the distribution configuration before changes.

xh-request
- request the presence of a package, by sending mail, and if possible, modifying local configuration.

xh-transfer-requests
- copies requests information to a server. Usually not necessary (unless in a hurry and using xh-distribute).

First Installation

xh-first-time
- put the basics in an empty /software.

xh-check
- verify the existence of the basic xhier pathnames.

Miscellaneous

xh-maintenance
* executed regularly by cron to maintain the xhier structure.

xh-local-maintenance
- the local part of xh-maintenance.

xh-dist-maintenance
- the distribution part of xh-maintenance.

xh-make-links
- create links to package directories from common system directories; needed when moving packages around.

Makefiles

xh-install-file
    * installs a file in a specified location

xh-install-man
    - install a man page from its source.

xh-make-r
    * invoke xh-make in some subdirectories.

xh-strip
    - strip a file, if it can be.

setogm
    - set owner group and mode on a file.

wipcheck
    - check for a Work_In_Progress file.

System Startup

xh-boottime
    * execute package system startup programs.

xh-add-rc
    * add to the system boot-time startup script.

xh-del-rc
    * delete a package entry from the system startup script.

Housekeeping

xh-cleanup
    * deletes old temporary files produced by the xhier package.

xh-sniff
    - checks for packages that aren't being updated regularly.

xh-source-monster
    * deletes old source sent from a remote system.

xh-whine
    - checks the structure of a package.

Updating System Configuration Files

The following are used by the mechanism that invokes a package installation program, and in theory never need be invoked otherwise.

xh-crontab
- regenerate crontab file from package additions.

xh-inetd-conf
- regenerate inetd configuration file from package additions.

xh-login-shells
- generate /etc/shells file from package additions.

xh-mail-aliases
- regenerate system mail aliases file from package additions.

xh-patch
- apply package patches to system text files.

xh-services
- regenerate TCP/UDP services files from package additions.

Package Registry

xh-register
- registers characteristics of a package with a central registry.

xh-registrar
* the registry daemon.

xh-registry-stats
- create a usage summary for the xhier registry.

xh-reg-roller
* roll xhier_registry spool/log files.

software(i)

xh-get-synopses
* gather (local) package Synopsis files

xh-make-software
* create software(i) from gathered Synopsis files.

Miscellaneous

rdist_merge
- merge rdist output for multiple machines.

rdist_summary
- summarize rdist output.

rdist_trash
- remove unwanted lines from rdist output.

xh-available-packages
- create lists of packages available from a server.

xh-check-registration
- check if a package name is registered.

xh-dist-files
* creates a Distfile for a package.

xh-dist-paths
* asks questions of package file-types files, and builds Distfile fragments.

xh-gather-lists
* used to combine sets of related config files into a single file.

xh-inet-su
* simplifies some inetd.conf files.

xh-last-dist-step
* the last step in distributing a package.

xh-mail
* send mail to maintainers.

xh-make-xhier-aliases
* creates the local mail aliases for the xhier package.

xh-options-create
- create a template for a package options file.

xh-quote
- apply quotes to text for use by shells.

xh-remote-make
- runs xh-make on remote hosts.

xh-set-local-maintainer

* create a default local_maintainer mail address file.

xh-tweak
- modify a package to agree with today's structure definitions.

---

xhier-config - configuring the "xhier" package

DESCRIPTION

The xhier package configuration determines how the local host will receive, maintain, and distribute software. The initial configuration of the package, performed by the package's installation program, results in no distribution from the local host. In order to distribute a package to a specific host, the host must be configured to request the package, and access rights between the desired destination and some host that already has the package must be defined

Configuration File Conventions

All of the configuration files described in the following may contain comments (that start on a new line with a "#") and blank lines that will be ignored.

Most configuration files are local, which means that they are maintained on each host. Some of the configuration files are administration specific, which means that they are maintained on one host in an administration and distributed to the rest of the hosts in the administration. Such files should start with a comment giving the name of the administration master host where the files are maintained. All administration specific files are automatically initialized with such a comment, and are identified as such below.

All configuration files are (intended to be) in /software/xhier/config/ with all of the administration specific files in /software/xhier/config/admin/, and the regional files in /software/xhier/config/regional/. That's currently not the case though, as there are still some configuration files elsewhere, e.g. in /software/xhier/data/, /software/xhier/local/ and /software/xhier/admin/, that haven't been moved yet.

Requesting a Package

The names of packages intended to be delivered to a host are determined by the following requests files:

/software/xhier/data/default-requests
: is a shared requests file. This is just a list of packages required for Xhier itself to function.

/software/xhier/config/admin/requests
    is an administration specific requests file. Put the names of packages intended for (almost) all of your machines here.

/software/xhier/config/local/requests
    is a local requests file. In practice, this contains almost all of the package names intended for a host. Administration specific requests can be overridden here.

The files can contain a list of package names, one per line. See man xhier-requests for a detailed description of the syntax. See man software for a list of possibly available packages. There is currently no way for a client to determine what packages are available on a server.

Distribution hosts gather the requests files regularly (currently once an hour) and merge them to produce a list of packages to be delivered. Automated deliveries typically only happen once a week, but the first distribution of a package usually happens within a day or so (see "man xh-request").

The list of requests is limited by the access rights defined for this host, and by the packages available on the servers defined in the access rights. Thus it's possible to request a package and never receive it, or only receive parts of it should the access-rights be inadequate. There is currently no mechanism that will inform you of a missing package, or missing parts of a package.

Over time, the local requests file will likely grow, as programs that modify the file (e.g. as the result of an xh-dist2 or xh-request) do so by appending text to the file. This results in a crude history of changes. All but the most recent changes can of course be deleted from the file should it get annoying long. One way to do this on machines that request just a list of packages (i.e. most machines) is to use:

    xh-packages -M >/software/xhier/config/local/requests

which will produce requests for every package that has been distributed to the local host (but will remove any comments that were present).

Monitoring Xhier Housekeeping

Various housekeeping activities are performed periodically that often result in mail messages describing what needs to be attended to (see "man xh-maintenance"). These mail messages are mailed to the maintainer of the xhier package. They are also sent to the mail addresses in the file

/software/xhier/data/local_maintainer

The contents of local_maintainer file are determined from The

/software/xhier/config/admin/xhier_maintainer

and

/software/xhier/config/local/xhier_maintainer

files, representing the administration-wide and local maintainers. These files are combined, striped of comments, and have all duplicate entries removed before the result is place in the local_maintainer file. Addresses are valid if they are acceptable to the xh-mail command (on all supported architectures), so it's best to restrict the contents to simple addresses (e.g. of the form user@site).

Nature of the Current Host

The configuration of a package in the current host varies depending on the nature of this host (eg. standalone, regional server).

Whether the current host is a standalone or not is recorded in the local file:

/software/xhier/data/standalone

This is a plain-text file containing the string `on' or or the string `off'. By default, the xhier installation programs puts `off' in this file, implying that the current host is not standalone. A standalone host is not (in general) connected to a network (and thus receives no regular updates via a network, and doesn't send package status information via a network). xh-is-standalone(8) can be used to change the default or test the status.

The file

/software/xhier/data/hosts/regional_server

contains the name of the server host for the software region to which the current host belongs. A software region is a group of hosts that shares the same passwd file and home directories, and thus the same user community. These hosts usually share files via NFS mounting of filesystems on one central host. That host is known as the regional server.

The file

/software/xhier/data/architecture

contains the name of the current host's architecture. It's a local file (rather than architecture dependent) to avoid it being inadvertently set incorrectly. See man xhier-arch-types for a description of architecture names.

The file

/software/xhier/data/admin/admin_server

contains the name of the current host's administration. It's an admin file to avoid it being set incorrectly. It must be set by hand on the administration server for the hierarchy. It can be either a host or domain name.

Defining Default/Allowable types

WARNING: the allowed-types file is under construction, so it serves mostly as a comment for the moment, in spite of what the following says. When complete, it will make /software/xhier/data/architecture /software/xhier/data/admin/admin_server and

/software/xhier/data/hosts/regional_server obsolete.

The file

/software/xhier/config/local/allowed-types

names the instances of each distributable non-shared file type, i.e. admin, arch and regional file types, that are intended to be present on the local host.

Instances of arch specific files are named using the names documented in man xhier-arch-types.

Instances of admin specific files are named by a fully qualified domain or host name. The name should be a name that is suggestive of the administration and that is unlikely to change as machines come and go (so domain names are often a good choice).

Instances of regional files are named by the fully qualified host name of the regional server. A software region is a group of hosts that shares the same passwd file and home directories, and thus the same user community. These hosts usually share files via NFS mounting of filesystems on one central host. That host is known as the regional server.

Non-comment entries in the allowed-types file follow the syntax:

<filetype>=<primary_instance>(<instance>)*

Files of type <filetype> on this host are expected to be one of the named instances. <primary_instance> files are stored under /.software/<filetype>. Files of other instances are be stored under /.software/<filetype>_<instance>. See man xhier-nfs-conventions for details of how one might use such files.

For example

arch=OSF1_Dec.1.3-Alpha Ultrix4.2-Mips

would mean that this machine wants/accepts arch distributions from machines which distribute either OSF1_Dec.1.3 for Alpha machines or Ultrix4.2 for Mips machines. The OSF distribution is stored under /.software/arch and The Ultrix distribution is stored under /.software/arch_Ultrix4.2-Mips.

The xhier installation program determines default values for the type instances from both the soon-to-be obsolete files /software/xhier/data/architecture, /software/xhier/data/hosts/regional_server, and /software/xhier/data/admin/admin_server, together with allowed-types values propagated from servers via the /software/xhier/data/default-allowed-types/ directory. For example, setting the administration name on an administration master will result in unconfigured machines that

receive admin files from that machine to be automatically configured as being in the same administration.

Should the default allowed types for a machine change from the current configuration, a warning is produced, as that's usually a sign of misconfiguration either on the local host or on the machine distributing files to it. The result of such misconfiguration can be files of various types being distributed to /.software/<filetype>_<instance> rather than to /.software/<filetype>. Depending upon partitioning, that could result in a (root) partition filling up. But that's considered to be better than clobbering valid files.

Altering Options of Other Packages

Some of the options found in /software/PackageName/.admin/Options are recommendations by package maintainers, rather than rules. As such, they can be altered. The options that can be altered are the installation, distribution, and defaultPath options. The value for any of the above options for a package named PackageName is determined by consulting the following files in the following order:

/software/PackageName/.admin/Options
/software/xhier/config/admin/packages/PackageName
/software/xhier/config/regional/packages/PackageName
/software/xhier/config/local/packages/PackageName

The last option value encountered is the value used.

Changes to the defaultPath option in /software/xhier/config don't take effect until the xh-update-standard-packages command is run (at least for the affected package), and then the xh-make-links command is run. This isn't done automatically by (re)installing xhier because of the overhead involved, so if you're not willing to wait for the automatic update (usually done weekly), you'll have to run them yourself.

Examples

Sometimes a package is present on a machine only to facilitate further distribution, and its installation causes undesired effects on the machine. In such cases, it's appropriate to set installation=never in /software/xhier/config/local/packages/PackageName.

In the case that a package is being ported to an architecture master, it may be useful to inhibit distribution until porting is complete, just in case there is an outstanding request for the package. This would be done by putting distribution=none in /software/xhier/config/local/packages/PackageName.

The choice of which packages ought to be in the standard search rules is a matter of policy, taste, and knowledge of the local environment. Thus a machine administrator may wish to set the defaultPath option in any of the three packages/PackageName files, depending upon which machines are to be affected.

Miscellaneous Options

The local file

/software/xhier/config/local/options

contains options (in the style accepted by the xh-options command) that affect the operation of the package. They are:

inetdWrapper=/xhbin/tcpd

which is an optional entry that provides the inetd.conf building tool with the path name of a wrapper program to be applied to all added network daemons.

wrapperExclude=identd

which is an optional list of network daemons that should not be wrapped by the preceding wrapper. It is a comma separated list of program names. If the last component of the program path in the inetd.conf entry matches one of the program names in the wrapperExclude list, the entry will not be wrapped.

The administration specific file

/software/xhier/data/.cshrc_additions

defaults to a .cshrc fragment that prepends the showpath Usedby=maintainer standard search rules to the current search rules.

The administration specific file /software/xhier/data/config.d can contain these options:

XhierRootPath on
means append /software/xhier/data/.cshrc_additions to /.cshrc.

XhierRootPath off
means don't modify /.cshrc.

Configuring Access Rights

The access-rights file

/software/xhier/data/access-rights

is an administration specific file that describes what packages and types of files are allowed to move to or from the machines in an administration. An administration is the set of machines that receive the same versions of administration specific files, so access-rights files are used to define administrations. An administration is usually defined to be a set of machines configured by the same group of people (i.e. the systems administrators of the machines).

The existence of Access Rights for a host (A) to a specific package on a specific host (B) simply allows the package to be distributed from A to B. It does not cause the distribution to take place. For that to happen host A must already have the package installed, and host B must be have requested the package (as described in the previous section), and the distribution mechanism must be invoked.

Because not all files in a package can be shared with (i.e. distributed to) all hosts, access rights depend upon the kinds of files that a host may receive (from the local host). They correspond roughly to the file types described in xhier-file-types(7), together with some additional types.

See man access-rights for more details.

Defining Aliases for NFS Servers

the file

/software/xhier/config/local/servers

can be used to assign an alias to an NFS server so that servers can change without having to rename mount points. Most clients rarely change servers if ever, so this file is usually empty. See man xhier-config-servers for details.

Distributing Packages

In order to distribute software (rather that just receive it), it is necessary to have the xhier,dev part of Xhier installed. I.e. request that package, configure access rights for the hosts to receive the software (on both this host and the destination hosts), and then distribution will happen automatically once a week.

FILES

/software/xhier/.admin/Maintainer
- mail address(es) of the xhier package maintainer.

/software/xhier/config/*/requests
- names of packages requested for delivery.

/software/xhier/data/access-rights
- determines what files can be accessed by specific hosts.

/software/xhier/data/standalone
- indicates whether the current host is standalone.

/software/xhier/data/regional_server
- name of the regional server host.

/software/xhier/data/architecture
- name of the current host's architecture.

/software/xhier/config/local/allowed-types
- specifies the default/allowable values of the architecture, administration and region for the host.

SEE ALSO

xhier-config-servers(5) - format of the config/*/servers files.
xhier-file-types(7) - names for how files are shared.
xhier-arch-types(7) - names for machine architectures.

xh-merge-lists(8) - used to merge host lists.
xh-dist2(8) - handy way of sending a package somewhere.
xh-update-standard-packages(8) - use after updating defaultPath.
xh-make-links(8) - use after updating defaultPath.
access-rights(5) - control of access rights to package files.
xhier-requests(5) - syntax of a requests file.
xhier-nfs-conventions(7) - how to use /.software/Type_Instance.
software(i) - names of packages that might be available.
xhier(7) - description of the xhier software organization.

BUGS

Not all configuration files are in /software/xhier/config. There is currently no way for a client to determine what packages are available on a server. It's possible to request a package and never receive it, or only receive parts of it. There is no mechanism to notice a missing package or missing parts of a package. Almost no provision is made for remote mounting rather than distribution. The local_maintainer mechanism doesn't monitor package installation. The contents of the local_maintainer file should be characterized in an architecture-independent manner.

It shouldn't be necessary to run xh-update-standard-packages after changing the defaultPath option in /software/xhier/config/*/packages, instead an "xh-install xhier" should be adequate. And it could be argued that running xh-make-links (manually) shouldn't be necessary either.

There is some reduncancy in the configuration data, particularly the config/local/allowed-types file and data/architecture, data/hosts/regional_server and data/admin/admin_server.

---

xhier-howto - quick introduction to installing a new package

SYNOPSIS

You are given some source and you want to install it on an MFCF machine. Here's how.

Introduction

Software is packaged to facilitate its maintenance and distribution across a hetergeneous collection of machines. This is a cookbook style method for quickly getting your programs packaged, installed and running.

See "man xhier" for an explanation of the overall structure of xhier and the motivation behind the instructions given here. Since there are many details which don't affect most installations, you may not want to attempt reading about its details until you actually get stuck at some step in this installation process.

Cookbook for Installation

You are given some source and you want to install it according to the xhier conventions. Here's how.

You will follow these steps:

- divide the source into packages
- name the packages
- set up the source and xh-imakefile
- classify the files in the package
- install the files into the Package Installation Directory

1) Dividing source into Packages

If required, divide up a source collection into convenient packages. Each package should be reasonably self-contained and should group software that belongs together and that might reasonably be installed together. Small source collections, or collections in which pieces will never be separately installed or distributed, need not be split up. A package is indivisible; you can not distribute or install only part of a package on a machine. The remaining steps will deal with one of your packages; repeat the steps for each package.

2) Naming Your Package

Pick a name for your package. Avoid special characters or upper-case (see "man xhier-package-names" for the precise rules for choosing names). The name you are choosing should be unique among all package names in the known software universe. It must not conflict with any name currently chosen or likely to be chosen in future. This is a flat name space.

Package names are described, recorded (and reserved) in "man software". To reserve your name, send mail to the xhier package maintainer (using the mail addresses in /software/xhier/.admin/Maintainer).

Give some thought as to whether your package needs to incorporate version numbers in its name. For example, the ditroff package is available as ditroff-1 and ditroff-2, allowing new versions to be installed and co-exist with old versions. If you don't use version numbers, you must update the package "in place", and this may annoy the users of your package.

When you have finished the xhier process, all the files in your package will be accessible under the directory:

    /software/PackageName/

The directory /software/PackageName/ is known as the Package Installation Directory, since it is where the package is "installed" and ready for use.

3) Install the Package Source

Put the source for your package under the directory:

    /usr/source/PackageName/

on the machine that will be distributing your package. Examples of package directories: /usr/source/make/, /usr/source/ditroff-2/, /usr/source/x11/, etc.

4) Classify Package Files by Purpose

Your package will likely have several types of things that need to be built and installed. Most packages have an executable program or two, man pages, some have libraries, some have data files. Xhier makes distinctions between various types of things; each type is installed in its own specially-named directory located just under the Package Installation Directory. You need to choose which specially-named directory to use for each file. Recognized xhier directories include:

bin - user executable commands.

maintenance - system maintenance commands.

servers - daemons.

lib - object libraries.

include - #include files.

data - miscellaneous data files.

man - man pages.

doc - manuals too big to be man pages.

See "man xhier-package-structure" for explanations of each specially-named directory if you need help in classifying all the files in your package. The above classifications are recognized by the xhier system; don't capriciously name your directories other things.

The above classifications tell you exactly where you will install files under the Package Installation Directory /software/PackageName/. Some examples of these kinds pathnames are:

/software/mypkg/bin/somebinary
/software/mypkg/server/somedaemon
/software/mypkg/lib/somelibrary.a
/software/mypkg/man/man1/somemanpage.1

Remember these pathnames; your xh-imakefile (or Makefile) will need to know how to copy each file from the source to there.

5) Determine How to Install Files

There are two general approaches to placing files in a package. The first approach, often used when dealing with inflexible software that insists upon a specific organization for all of the files, is to place all of the supplied files in

/software/PackageName/distribution

then make symbolic links from the rest of the standard package locations into the distribution. The distribution directory is typically typed as arch, i.e. it may be necessary to direct any supplied installation tool to put the package's file into

/.software/arch/PackageName/distribution

It's often the case that only a few symlinks need be made into distribution. For example:

man -> distribution/man
doc -> distribution/info
bin/prog1 -> distribution/tools/prog1
bin/prog2 -> distribution/tools/prog2

For those software suites that mix the purposes of files in a single directory, more symbolic links will be needed, and this becomes the only practical approach for packaging.

The second approach is to simply put the files of the software directly into the standard xhier locations. For software with source, that usually means making or modifying a Makefile.

6) Construct a standard Makefile

You have to build a Makefile for your package that is consistent with the standard Makefiles for all the other packages. MFCF provides a Makefile building tool that you can use to build a standard Makefile for your package by starting with an xh-imakefile. See "man xh-imakefile". The standard stuff knows how to construct and classify pathnames automatically; use it if you can.

If the package you're installing already has a complex Makefile, you might do best by moving all the source down one directory level and creating a standard Makefile that simply calls the Makefile of the package in the sub-directory, possible supplying variable definitions on the make command line to override the defaults in the supplied Makefile. If the package has a simple Makefile, modifying it to be "standard" might be easier.

Arrange that the Makefile knows about at least the following targets:

help, all, install, clean

The install target of your standard Makefile must copy each installable file from the source directory to the installed place you have chosen for it under the Package Installation Directory. That is, it will copy the file from the source directory to the constructed pathname you chose for it above. (The xh-install-file command may be useful here.)

Examples of files installed under their Package Installation Directory:

/sofware/gcc/bin/gcc
/software/ditroff-2/bin/troff
/sofware/find/maintenance/updatedb
/software/x11/lib/libX11.a
/software/sendmail/data/local-rscs
/software/x11/man/man1/xterm.1

The install target for a package should install at least all of the non-shared installed files in a package. Since shared files are identical everywhere, and thus often lack "source", having source for shared files (e.g. man pages) is optional. The Package Installation Directory on the central distribution machine will serve as the master copy of the shared files for a package. Thus, the shared part of the Package Installation Directory is considered to be part of the package's source, and you do not have to have your Make-file know how to create it. (You are allowed to have RCS directories in a Package Installation Directory. The RCS directories are not distributed when a package is dis-tributed to other machines.)

Once your xh-imakefile (or Makefile) knows how to install all the non-shared files in a package, you must create some file shareability structure before trying a "xh-make install". Read on.

7) Classify by Shareability

You have already decided what purpose each file in your package serves, and have chosen appropriate xhier sub-directories under which to put each (bin, lib, man, etc.). Thus, you know the exact installed pathname of every file that will be in your package.

Now, for each file you are installing, classify the file according to how shareable the file is between different

hardware (e.g. DEC, Sun, Sequent, SGI, MIPS, ...) and different software (e.g. 4BSD, Ultrix, SunOS, Dynix, Irix, UMIPS, ...). A summary of the relevant classifications are as follows; see "man xhier-file-types" for details.

shared
- files that are identical on all machines, e.g. most documentation and man pages.

arch - files that are identical only on machines that have the same hardware and software architecture, e.g. compiled binaries and object libraries.

regional
- files that are the same on machines that have the same user community and that share the user's files. It's unlikely you'll need to use this type.

admin
- files that are the same within a system administration, e.g. defaults for some configuration files.

spool
- local files that are transient in nature, e.g. files queued waiting for printing.

local
- files that are (or might be) different on different machines, e.g. host specific configuration files.

Having determined the shareability Type (above) of a given file, create the structure of the package by using the xh-mkpath and xh-mkdir command on each file. That is, if a file that you want installed as

```
/software/PackageName/somepath
```

is of shareability type Type, then use

```
xh-mkpath -t Type /software/PackageName/somepath
```

which will record the file's Type and create any directories needed for the creation of somepath. For example, if you decide that

```
/software/gcc/bin/gcc
```

is of shareabilty type "arch", you would type

```
xh-mkpath -t arch /software/gcc/bin/gcc
```

If your determination of file types is typical, then you'll likely discover that the xh-mkpkg command will create what you need, and then some. After using it, be sure to delete any unneeded paths (including those in /.software).

If there is a directory of files of the same type, as is often the case with the bin directory, then it can be handy to simply type the directory first (which has the effect of typing all of its files correctly) by using xh-mkdir, e.g.

    xh-mkdir -t arch /software/PackageName/bin

is quite common. While it's not technically correct, it's become common practice. In fact, it's what xh-mkpkg command does.

Files in the Package Installation Directory will always be accessible using:

    /software/PackageName/somepath

however, after xh-mkpath, such a path will often be a symbolic link to a place reserved for files of that particular Type (that's how the type is recorded). Separating the actual locations like this simplifies distribution a bit, and greatly simplifies accessing packages using remote file systems such as NFS.

Once you have used xh-mkpath and xh-mkdir to set the shareability of every file in your package and create the necessary symlink structure, you can use "xh-make install" in your source directory. If all has been done correctly, your xh-imakefile (or Makefile) will make and copy each file to the Package Installation Directory.

8) Determine What Other Packages Yours Depends On

If your package requires programs or files from other packages for its operation or installation, you should list the names of these other packages in the file

    /software/PackageName/.admin/Dependencies

and this will ensure that those packages are available for your package to use.

9) Write a Short Description/Synopsis of Your Package

Write a short description of your package, suitable for inclusion in the software(i) man page, and put it in the file

    /software/PackageName/.admin/Synopsis

10) Write an Installation Program

If your package needs anything other than just the make install in the source directory to make it fully operational (e.g. it needs some host specific configuration after all the files are installed in the Package Installation Directory), provide an Installation Program for the package that does what you need.

The Installation Program should be an executable file named:

    /software/PackageName/.admin/Install

Packages that require no initialization or special installation must still have an Installation Program. All it has to do is exit normally (zero).

The Installation Program can initialize local files to reasonable default values (when possible), and tell the invoker what should be done to make the package fully operational. If the Installation Program detects that files for which there are no reasonable defaults haven't been initialized, it should exit with a non-zero exit success to indicate that the installation failed.

An Installation Program must be able to undo whatever it did to install the package when given the -u option.

In addition, the Installation Program of a package is responsible for making any symlinks from the OS supplied file system into the package (use xh-ln for this). Most packages won't need to create such links, but sometimes software is provided in such a way that it's not practical to change certain assumptions.

For example: a package might require that /usr/lib/MyStuff exist. Rather than create the directory and its contents there, in the vendor file system, we keep the content under /software/ with the rest of the package and create a symbolic link to it from /usr/lib/MyStuff, e.g.

    xh-ln ${PackageDir}/data/MyStuff /usr/lib/MyStuff

would be used for installation and

    xh-unln ${PackageDir}/data/MyStuff /usr/lib/MyStuff

would be used for deinstallation.

The variables ${PackageDir} and ${Package} are two of many predefined in the Install template created by the xh-installation-prog command.

Another example of symlinks that need to be created are the symlinks in /usr/include (or wherever your operating system puts them) that allow include files supplied with the package to be accessible. This linking will be done automatically by xh-make-links if you observe the following naming rule: A package's include files must be put under one or more include/IncludeName directories under the Package Installation Directory:

```
/software/PackageName/include/IncludeName/
```

For example:

```
/software/gnu/include/gnu/
/software/x11/include/X11/
/software/c++2.1/include/c++/
/software/kerberos_dev/include/kerberos/
```

After xh-make-links runs, the package include directory will be merged with all similarly-named directories and symlinked correctly. Include files will thus be accessible in programs using

```
#include <IncludeName/file.h>
```

The easiest way to make an installation program is to use xh-installation-prog to create a template of a CSH script that will be the installation program for the package. xh-installation-prog uses the structure of the installed package to determine the names of local files. It's invoked as part of xh-mkpkg. E.g.

```
xh-mkpkg PackageName
vi /software/PackageName/.admin/Install
<...edit and fill in the template...>
```

You might want Install program itself to have source, in which case you should make your xh-imakefile (or Makefile) install the Installation Program from it source.

Look at other xhier Install scripts to learn more.

11) Configure Changes to System Files

Some packages require that certain system configuration files be changed. For example a daemon might be required to be started at system startup (boot) time or some process may have to be run every day from the cron.

Changes that require modifying the operating system's configuration are the responsibility of the Package Installation program. However, for changes to some common system configuration files such as the system crontab and inetd.conf, xhier has an automatic mechanism that saves work. The xhier-package-structure man page explains how to create files under

```
/software/PackageName/export/
```

that will automatically be included in the appropriate system configuration file. The additions take effect once the package's installation program has been successfully run by xh-install.

12) Identify Yourself as the Package Maintainer

Whoever is to be the maintainer of the package should record their mail address(es) in

```
/software/PackageName/.admin/Maintainer
```

The file is processed by decomment, so you can include comments as well, e.g.

```
# mail address of the package maintainer
MyUserid@MyMachine
```

13) Set Any Appropriate Options

If appropriate for this package, use xh-options-create to create a options file

```
/software/PackageName/.admin/Options
```

and set any appropriate options.

14) Test the Installation

Try a xh-make install in your package source directory and see if the non-shared parts of the package are correctly created and installed (with the correct shareability) under the Package Installation Directory.

Find a machine on which it's safe for the software to be operational, and run

```
xh-install PackageName
```

which will invoke the Installation Program you've written and record the fact that the installation completed success-

fully. You can then test the functioning of the package. Ideally software should be tested on every architecture, although in practice that almost never happens. But at least try a simple test on at least one architecture.

This completes requirements for the basic installation.

You should now be able to "xh-make install" and have the different pieces of your package fly off to their proper resting places, possibly with symlinks left behind so that everything looks like it is under

/software/PackageName/somepath

Making Software Available

After "xh-make install", your installed software is accessible using pathnames starting with /software/PackageName/. You can use the commands in the package by adding the package name to the showpath invocation that sets your search rules in your .cshrc, e.g.:

```
setenv PATH `/bin/showpath PackageName MyUsualOptions`
```

If your package is something you think everyone on your machine should have in their default search rules, see a system guru to have the list of default packages updated ("man xhier-config" describes how it's done). Then, assuming none of your binaries have name conflicts with any other binaries on the system, commands in your software's /bin directory will be linked in with the standard list (known to the showpath command). The man pages for your package will also eventually be added to the rmand man page database, the next time the database is rebuilt.

Distributing Your Package to Other Machines

If your package is to be distributed to other machines, see a system guru to have a distribution configuration changed; "man xhier-config" describes how it's done.

If you are making the software available on machines of a different architecture (i.e. where it will have to be rebuilt from source), you might want to use xh-sdist, as in:

xh-sdist /usr/source/PackageName -m "clean all"

which will distribute (updates from) /usr/source/PackageName to the machines of a different architecture currently configured to receive source updates and then invoke

xh-make clean all

for you.

FILES

/usr/source/PackageName/
- where to put the source for your package.

/software/PackageName/
- where the installed files in the package are.

SEE ALSO

xhier-package-structure(7) - what a package looks like.
xhier(7) - description of the xhier software organization.
xh-imakefile(7) - what xh-imakefile's look like.
xh-mkpkg(8) - create a typical package structure

AUTHOR

IDAllen@watcgl.waterloo.edu, with additions and changes by MFCF staff.

BUGS

This is hard to read. It was harder to write.

---

xhier-package-structure - the structure of a package

SYNOPSIS

Every package has an Installation Directory in
/software/package_name/
that (appears to) contain all of the files in a package, organized according to the following conventions.

Some packages have multiple versions, only one of which is in the standard search rules at a time, and which is always referred to with the same name (i.e. without a version number). In the following, package_basename refers to the name of the package without a version number.

1 Package Structure

It is permissible to use names not documented below, however it is highly recommended that conventions be determined for any new names so that they may be applied to all packages.

/software/package_name/
the installation directory for the package named package_name. Contains shareable files together with symbolic links to all other files associated with package_name.

1.1 The Package Proper

bin/ user commands.

maintenance/
commands only in the search rules of maintenance personnel.

servers/
programs invoked only by other programs.

include/
directories of include files.

include/package_basename/
include files supplied by this package. It is permissible to create additional package_basename links to the original in varying case, if that assists in the use of the package. A justifiable case for doing this would be a collection of include files (include/package_basename/*) that reference the upper case form of the package name (as in "#include <PACKAGE_BASENAME/...>").

lib/ libraries.

data/
preferred location for miscellaneous data files.

man/ documentation for use with the man command.

doc/ miscellaneous documentation not suitable for access via the man command.

private/
parallels the package structure. Contains things not to appear in any (showpath produced) search rules.

servers/
as servers above, but these don't appear in any search rules.

config/
the preferred location of all package configuration files that can be changed by administrators, together with shared defaults. See man xhier-package-config for details.

export/
contains files examined by other packages when providing services for this package. It's currently the case that all such files are in fact related to package installation, so it's not obvious that this description doesn't belong in the next section. All of the following are optional; they need not appear in a package's export directory. All but one of the following (paths) are additions to (effectively) be made to some system control file. Such additions are made only when the package's installation program completes successfully. If a package's installation program fails to complete successfully, then no changes are made.

aliases
is a fragment of a sendmail system mail alias file. See man xh-mail-aliases for details.

boottime
is a program that will be executed at system startup, provided that the package's installation program completed successfully when last run. See man xh-boottime for details.

crontab
    is a fragment of a generic crontab that will be converted to the crontab form in use on the system and added to the system's crontab. See man xh-crontab for details.

group
    describes requirements for file system groups (i.e. for the /etc/group file). The syntax is an extension of the syntax of the system group file. See man xh-group for details. Xh-group is not yet implemented, so this file is essentially documentation.

inetd.conf
    is a fragment of an inetd.conf file, i.e. a fragment of the inetd daemon configuration file. See man xh-inetd-conf for details.

passwd
    describes requirements for userids (i.e. for the /etc/passwd file). The syntax is an extension of the syntax of the system passwd file. See man xh-passwd for details. Xh-passwd is not yet implemented, so this file is essentially documentation.

patch/
    contains patch files, in diff(1) format, that are to be used to tailor system text files. See man xh-patch for details.

paths/
    defines package specific search rules accessible via the ":" syntax in the showpath command. See man showpath for details.

services
    is a fragment of a TCP/UDP services file. See man xh-services for details.

shells
    describes requirements for allowed login shells (i.e. for the /etc/shells file). The syntax is an extension of the syntax of the system /etc/shells file. See man xh-loginshells for details.

shutdown
is a program intended to undo what the export/boottime program did. It should be written so that it would work if invoked anytime after the export/boottime had run. It is invoked automatically by `xh-install -u`, and (when possible) during system shutdown.

1.2 Installation, Maintenance, and Distribution Files

`.admin/`
contains files used by the installation and distribution mechanism. All of them are shared, but some of them need not be present in every package.

Install
the (de)installation program for the package. It installs by default, and will uninstall the package if invoked with the -u option. This program is intended to be invoked by xh-install, so that the success or failure of the installation may be recorded. It is expected to issue messages describing any changes made to the system, and any failure to complete the installation. It is expected to issue no other messages if the installation of the package is successful.

Dependencies
an optional file containing a list of packages that this package requires before it can be installed. Can contain # comment lines, which should document the reason for each dependency. Each package name must appear on a separate line. The xhier package is assumed to be a dependency of all packages, and thus never needs to be included.

HostMaintained
a required file that is created by xh-tweak(8) and interrogated by xh-whine(8), and should not normally be manipulated by humans. It was created to help identify software distribution problems and should contain the second field of the output of the hostnames command from the package's maintenance machine. Can contain # comment lines. If the maintenance host of a package changes, the maintainer should remove this file and allow it to be re-created.

Synopsis
a required file containing a description/synopsis of the package suitable for inclusion in the software(i) man page. It should start with one of \-, + or * to indicate whether the software in the package is un-modified, UW modified, or UW originated (see software(i) for details. Can contain # comment lines.

Maintainer
a required file containing the (electronic) mail address(es) of the maintainer(s) of this software package.

Options
an optional file containing the options that the maintainer(s) of this software package feel are appropriate for this package. Created by xh-options-create(8) and queried by xh-options(8). See options(5) for details.

Targets
an optional shared file containing the names of hosts to which this package is allowed to be distributed. This allows the package maintainer to list the hosts licensed for the package (the primary application of the file). The file is processed by xh-merge-lists, so hostnames should be one per line. The hostnames must be in the form used in the access-rights file (i.e. as output by the hostname command on the target machines). For historical reasons, it is only honoured if the targets=restricted option is set in the Options file.

file-types
an optional file that can be used to define new file types and redefine the default file types. See man xh-dist-paths for details.

1.3 Names Reserved For non-Xhier Use

Using any name in an installation directory (/software/PackageName/) that is not documented may eventually result in a conflict (i.e. when/if it is used by the Xhier mechanism), so it's best to avoid doing so. In addition it would be useful to agree upon interpretations of names, which can only be done if the names are documented.

What follows is a list of names that are safe to use in an installation directory, as they will never be used by the Xhier mechanism. Names (that have reasonably obvious interpretations) can be added to this list by contacting the maintainer of the Xhier package. Note that it would be nice to keep this list small.

The following names may safely be used in a package installation directory. Suggested interpretations are given where the use of a name is not entirely obvious.

1.3.1 Current Names

distribution/
the preferred location for files in a package that simply doesn't split up easily at all. In such cases most of the rest of the package structure will be links into this directory. The structure in this directory should match the structure suggested by the supplier of the directory contents.

logs/
the preferred location of log files produced by this package. Unfortunately it's awkward to roll logs in a shared directory, so this is usually a directory of type spool. The package is responsible for rolling its logs.

1.3.2 Deprecated Names

Most of the names in this list are present for historical reasons only. Their existence in the list is not intended to encourage their use in new packages. Of course, if a name has a simple characterization and is used in enough packages it will be moved into the current section above.

admin/
    administration specific versions of other files in the
    package.

conf/
    use config instead.

demo/

dict/

font/

home/
    a home directory.

includes/
    include files, one sub-directory per type.

info/
    Emacs info files (i.e. documentation).

lisp/

local/
    local versions of other files in the package.

locks/

obsolete/

pids/

share/
    files contributed to a package by various users ?
    We're not really sure.

spool/
    this name is probably too general, so avoid it if pos-
    sible.

1.4 Files that Unavoidably Exist Outside of /software

The pathnames used in the following are those used on most systems. The pathnames on some architectures are slightly different.

/usr/lib/library.a
: is a symbolic link to /software/package_name/lib/library.a if it is a replacement for a system supplied library.

/usr/local/lib/
: contains symbolic links to the libraries in the lib directory of installation directories of those packages that are in the standard search rules.

/usr/include/package_basename/
: is a symbolic link to /software/package_name/include/package_basename (if present).

1.5 Other Reserved Names

/usr/source/package_name/
: source for the named package.

/usr/source/TEMP-CACHE/package_name/
: source for the named package copied from its maintenance machine while being rebuilt here.

/vendor/path
: where the originals of path are placed if it proves necessary to replace or delete them. This is a usually a symbolic link (to avoid filling up the root partition).

/xhbin
: is a real directory under the root partition in which stand-alone programs (e.g. diagnostic programs needed when single-user) can be found. It also contains programs that must have short absolute pathnames; this is needed for scripts that use the "#!" invocation for which there is a limit of usually 32 characters on the interpreter name (and arguments).

1.6 Where Many of the Files Really Are

Files (in an installation directory) that can't be the same on every host (in the world) must be symbolic links to the version appropriate to the current machine. In general, if a file

/software/package_name/Path

is not shareable everywhere, then it will be of a particular Type and will be a symbolic link to

/.software/Type/package_name/Path

See man xhier-file-types for a description of the possible file types, and man xh-mkpath and man xh-mkdir for a description of tools to assist in the creation of the links.

SEE ALSO

xhier(7) - description of the xhier software organization.
xhier-package-names(7) - rules for naming packages
xhier-config(7) - how to configure the xhier package.
xhier-file-types(7) - names for how files are shared.
xhier-package-config(7) - conventions for configuration files.
xh-merge-lists(8) - pre-processes various configuration files.

xhier-arch-types - names for the architecture of a machine.

DESCRIPTION

It is sometimes necessary to be able to use consistent names for an architecture across multiple machines or programs. To that end, an architecture_type is defined to be of the form

<software_type>-<hardware_type>

or just

<hardware_type>

when the <software_type> is the same as that of the local machine, or just

<software_type>

when that necessarily implies the <hardware_type> and there is no <hardware_type> defined for the architecture. Depending upon context, the hardware and software types may be dual case words (rather than all lower case).

Currently defined <software_type>s are:

AIX4.2 AIX4.3
Irix5.3 Irix6.5 Irix6.5_64
OSF1_Dec.4.0
Linux2.2
SunOS4.1 SunOS4.1.3
SunOS5.4 SunOS5.5 SunOS5.6 SunOS5.7 SunOS5.8

and these obsolete types are defined as well:

AIX3.1 AIX3.2 AIX4.1
BSD4.3
BSDi2.0
Dynix3.0 Dynix3.1
Irix3.3 Irix4.0 Irix5.1 Irix5.2 Irix6.2
OSF1_Dec.1.2 OSF1_Dec.1.3 OSF1_Dec.2.0 OSF1_Dec.3.2
Slackware2.2
SunOS3.5 SunOS4.0 SunOS4.1.2 SunOS4.1.3_U1 SunOS4.1.4
SunOS5.2 SunOS5.3
Ultrix2.0 Ultrix3.0 Ultrix4.1 Ultrix4.2 Ultrix4.4
UMIPS4.0 UMIPS4.5

The version numbers of an operating system that denote minor bug fix releases are usually not included in the <software_type> as such releases rarely cause significant architectural changes.

Currently defined <hardware_type>s are:

Alpha IbmRS Intel Mips Sun3 Sun4 Vax

No attempt is made to include such things as floating point architecture, or other variable aspects of a hardware platform.

FILES

/software/xhier/data/architecture
- contains the (dual case form of) the architecture type of the local host (with optional comments).

/.software/arch_architecture_type/
- contains the type arch files for the particular architecture_type (which in this context is all lower case), if it differs from the local host's own architecture.

SEE ALSO

xhier(7) - description of the xhier software organization.
xh-arch(8) - queries the local host architecture type.

BUGS

The dual case form of the names is used in some contexts, but not in others. The defaulting rules are perhaps extreme. The simple naming scheme given above comes nowhere near to being a complete specification of significant hardware and software characteristics (in many cases). Thus we wonder if some depressingly complicated attribute based scheme would eventually prove preferable.

xhier-file-types - names for how a file is shared between hosts.

DESCRIPTION

The software organization distinguishes between files based upon the extent to which they can be shared (made the same) on multiple hosts. In addition, files are classified by how they might differ in required location (e.g. due to disk partitioning conventions). Names for the resulting classes of files are used in several contexts, and are sometimes referred to as file share types or simply as file types. The names (types) are as follows.

shared
- files that can be the same on all hosts. For example documentation, symbolic links, shell scripts, and configuration files are often identical everywhere.

arch - files that can be the same on hosts of the same hardware and software architecture. For example program binaries and object libraries usually depend (only) upon the hardware and operating system in use.

admin
- files that are the same on machines administered by a specific group of system administrators. Administration specific files are often useful in providing a default for a local file across the machines "administered" by a single group.

regional
- files that must be the same on hosts that have the same user community and share the user's files. Regional files are shared read-write between hosts in a region using a remote file system, i.e. changes to regional files on any host in a region are seen by all other hosts in the region. This form of sharing generally only happens between (diskless) clients and their (single) server. For example files that contain per-user information (i.e. that have a one-to-one mapping with the passwd file) are considered regional.

spool
- local files that are transient in nature and thus are to be stored in a place reserved for such. E.g. temporary files in any queuing mechanism would be considered to be spool files.

local
- any other (potentially) local file. Note that even though a file may be potentially different on every host, administrative policies may result in a local file being the same on many hosts.

FILES

/.software/Type/PackageName
- files in the package PackageName of type Type (other than share).

/software/PackageName
- shared files in the package PackageName.

SEE ALSO

xh-mkpath(8) - uses share types.
xh-mkdir(8) - uses share types.
xhier(7) - description of the xhier software organization.

BUGS

The definition of admin is somewhat nebulous.

xhier-man-pages - man page name conventions

DESCRIPTION

There are several man pages that may be included with a package, containing information common in form among all packages. For a package named PackageName, the following man pages may exist.

man PackageName-support
: describes how the software is "supported", and a variety of other somewhat related information. Typical information includes where to submit bug reports and how to obtain (a license for) the software. See /software/xhier,dev/data/man-page-templates/PackageName-support.7 for a suggested template that describes itself. This man page is currently optional. Support information is sometimes provided with software in the man pages for the software itself.

man PackageName-changes
: is a summary of recent changes (in reverse chronological order) that may be of interest to the users of the package.

man PackageName-config
: describes what to do to (re) configure the package. Many packages require no configuration, and some packages configure themselves and have no "user serviceable parts", so this man page is optional. This man page will generally only be of interest to system administrators. See /software/xhier,dev/data/man-page-templates/PackageName-config.7 for a suggested template.

FILES

/software/xhier,dev/data/man-page-templates/
: a directory of templates for the man pages described above (and more).

SEE ALSO

xhier(7) - description of the xhier software organization.

BUGS

The PackageName-support man page is poorly named (nobody has thought of a single name that adequately describes the variety of things that can appear in the man page).

access-rights - xhier software distribution control file

SYNOPSIS

/software/xhier/data/access-rights

DESCRIPTION

This distribution configuration/control file provides xhdist-hosts(8) with the complete software distribution structure (tree/graph) for a given administration (the file is global to an administration). It describes all possible software distribution to and from every machine in an administration (thus there is some duplication between administrations).

The file consists largely of target hostnames and the corresponding hostname(s) from which they are to obtain/receive software. The various parts that make up the xhier software structure/organization are categorized into types. For example, shared, arch and admin.

Blank lines and lines beginning with `#' are ignored. A line starting with

```
%%filename
```

indicates that filename (an absolute pathname is most useful here) should be included at this point, as if it was part of the original file. Include files may be arbitrarily nested.

More specifically, a configuration line is of the form:

```
<TargetHost> \
    (<SrcHost>|<FileType>=<SrcHost>|<FileType>="["<FileType>"]")+ \
    (package=<PackageName>)*
```

A <TargetHost> of `*' is used to set the default relationships between types as well as a default rule when no explicit <FileType> is associated with a <SrcHost>. An expression of the form:

```
*  shared=[*] admin=[shared] man_pages=[] different_admin=[!admin]
```

sets the default action so that <FileType> "shared" matches any <SrcHost> that does not have an explicit <FileType> mentioned. "admin" is defined to be a <FileType> that inherits any host that matches "shared" (unless otherwise explicitly defined). The empty '[]' explicitly sets no inheritance from any other type. The "!" preceeding the "admin" type allows the type "different_admin" to match any host that does not match the <FileType> "admin."

If a <TargetHost> starts with `%', it is assumed to be a file in the xhier package's data/hosts-access directory and a list of target-hosts will be obtained from the file (with a record created for each host). If <TargetHost> contains a `/', it is assumed to be an absolute path to a host-list file.

Here is a small example to help demystify the format:

```
#  distribution config file
#

*       shared=[*] admin=[shared] structure=[shared] \
                arch=[shared] fpa=[arch] getwd=[arch] \
            man_pages=[] different_arch=[!arch]

# BSD4.3-Vax

%MFCF_Vaxen     watmath
grand           watmath   man_pages=watmath     admin=
watdragon                 man_pages=watmath

# Dynix3.0

maytag          watmath   man_pages=watmath     arch=

# SunOS4.0-Sun3

%lily_clients    lily   p=xhier p=mfcf-basics p=termcap
```

All the hosts listed in the file "MFCF_Vaxen" are to obtain/receive software from the <SrcHost> "watmath." Since no <FileType> is associated with the <SrcHost>, the default rule is applied. "shared" matches any host, so all files of type "shared" can be distributed to the target hosts. Now "admin," "structure" and "arch" all inherit any "shared" match, therefor each of these types can be distributed to the target hosts. Similarly, both "getwd" and "fpa" inherit what matches "arch" so they also can be distributed. As there is no explicit mention of the type "man_pages," files of this type will not be distributed by this rule. "arch" does match so "watmath" will not be of the type "different_arch" for each of the targets.

The <TargetHost> "grand" explicitly mentions the <FileType> "man_pages" which means files of this type can be obtained from the host "watmath."

The "admin=" item explicitly says do NOT send me files of this <FileType> (or more accurately, send me files of type "admin" from nowhere). This is done as "grand" is in a different administration from the host with this access-rights file.

Although the <TargetHost> "watdragon" is mentioned in the file "MFCF_Vaxen", this third record says that files of <FileType> "man_pages" will be sent to "watdragon" (since "man_pages" has a default action that associates no <SrcHost> by default).

The record for "maytag" asks that "watmath" send everything BUT things of <FileType> "arch" (architecturally specific files). Notice that <FileType>'s "getwd" & "fpa" have default actions that inherit the <SrcHost> from "arch". Thus, only things inheriting <FileType> "shared" ("admin" & "structure") may be sent/obtained.

The final record provides explicit packages that are to be sent from <SrcHost> "lily" to the hosts listed in the file "lily_clients". By default (i.e. no "package=" item(s)) all packages can be distributed. This record restricts the <package>'s that should be distributed to the <TargetHosts>'s. In this case, other packages will be mounted via NFS.

FILES

/software/xhier/data/hosts-access/
- location of include files used in an access-rights file.

SEE ALSO

xhier(7)
xhier-config(7) - how to configure the xhier package.
xh-dist-hosts(8) - the program that uses access-rights.

xhier-config-servers - file to map NFS server ids to host-names.

SYNOPSIS

/software/xhier/config/local/servers

This file contains a mapping from ids to hostnames for NFS servers for mounting software.

EXAMPLE

For a client (lily04) of a server (lily), the following done together could result in a valid configuration:

- lily04:/software/xhier/config/local/servers contains:
  lily MrBin

- on lily04, lily:/.software/arch has been mounted on /.software/arch@MrBin

- the access-rights for lily04 are:
  lily04 local_struct=lily p=p1 p=p2 ...

DESCRIPTION

The syntax of each data line of the file is:

ServerName ServerId

where ServerName is the name of an NFS server as it appears in the access-rights file, and ServerId is as it appears in mount point names as described in man xhier-nfs-conventions. A ServerId can be any word that follows the rules for host-names.

The file may contain comments (that start on a new line with a "#") and blank lines that will be ignored. It usually starts with some comments describing what it is and when it was produced.

The purpose of providing a mapping between hostnames and ids is to facilitate changing servers without having to change the names of mount points.

SEE ALSO

xhier(7) - description of the xhier software organization.
xhier-config(7) - how to configure the package.
xhier-nfs-conventions(7) - conventions for NFS mounting packaged software.
xh-nfs-links(8) - needs to map server ids to hostnames.

BUGS

Perhaps server alias would be a better way of describing a server id.

xhier-requests - format of package requests files.

DESCRIPTION

A package requests file describes the names of packages (intended to be present on a machine). A requests file is maintained on the machine that it affects (but see the "Transition to Client Originated Requests" section for historical exceptions). In general several requests files are concatenated into a single requests file in order to determine the set of packages being requested by a machine (see the "FILES" section below, and "man xhier-config" for details).

As with all xhier configuration files, comments (blanks lines or lines beginning with a "#") are ignored. Each non-comment line in a requests file is one of:

* - means include the names of everything available on accessible servers. The distribution=none and targets=restricted package options can affect availability.

PackageName
- means include the package name.

-PackageName
- means exclude the package name. Excluding a name not previously mentioned has no effect.

filename
- replaced by the contents of the named file (the filename must start with a "/"), and then processed as request file lines.

The order of entries in a requests file matters, as each entry is treated as a modifier to the list of package names collected so far (starting with an empty list).

The list resulting from the application of the modifiers that constitute a requests file should contain nothing but package names. The meaning of aliases (or names that aren't package names) in the resulting list is currently undefined.

WARNING: unlike most configuration files, requests files are almost always processed on some other machine, so any errors that are detected (and not all are) won't be reported locally.

Examples

This (together with some comments):

```
xh-packages -M >/software/xhier/config/local/requests
```

will produce requests for every package that has been distributed to the local host, and is appropriate in most cases (but beware of "stranded" packages, that weren't properly distributed in the first place and thus are recorded as having originated on the local host). Replacing the "-M" with a "-c" includes requests for packages that are maintained locally, which is safe to do and avoids the above problem at the expense of a possibly larger requests file.

This:

```
*
```

i.e. a request for everything available, is rarely appropriate by itself, and is usually followed by a small list of excluded packages. It is generally used only for "architecture masters" and clients that mount all of the packages on a server. Any other use of it is dangerous (as it can easily fill up whole disks).

Transition to Client Originated Requests

The following conventions exist only as aids in the transitions from server targets to server originated requests to client originated requests.

The presence of a file named

```
/software/xhier,dev/data/client-requests/.ClientName
```

indicates that the requests file for host ClientName originated on the server, rather than on ClientName. I.e. if the file is present, then the corresponding file

```
/software/xhier,dev/data/client-requests/ClientName
```

is maintained on the same machine (i.e. the server).

A package name of .NotUsedYet is used on a client to signify that the corresponding requests files on servers are considered authoritative. As soon as .NotUsedYet does not appear in any of a client's requests files, they will replace the server-originated requests file.

FILES

/software/xhier,dev/data/client-requests/ClientName
- server's copy of merged client requests.

/software/xhier,dev/data/client-requests/.ClientName
- indicates server originated requests.

/software/xhier/config/local/requests
- requests by the local host.

/software/xhier/config/admin/requests
- overridable requests by the administration.

/software/xhier/data/default-requests
- overridable default requests.

SEE ALSO

xhier(7) - description of the xhier software organization.
xhier-config(7) - how to configure the xhier package.
access-rights(5) - allowed distribution paths between machines.
xh-transfer-requests(8) - gathers requests from clients.
xh-targets-to-requests(8) - creates requests files.
xh-gather-lists(8) - used to process filenames in requests files.
xh-merge-lists(8) - used to process requests files.
xh-requests-conversion(8) - copies server requests to the client.

BUGS

It has been argued that the syntax for excluding a file
should only be allowed to apply to names that were obtained
from the expansion of "*" or some other file.

xhier-nfs-conventions - conventions for NFS mounting packaged software.

DESCRIPTION

Software that has been packaged according to the xhier conventions can usually be made to appear on remote machines via NFS filesystem mounts. Except in one very special case, the mount points can't be the usual package locations.

What Has to Be Mounted

For any given package, the files of each of the file types must be made available. Files local to a machine (i.e those of type local and spool) obviously aren't obtained remotely. Regional files for single-host regions are treated as local files. Regional files for multiple-host regions are always mounted (read/write, with root mapped to root) from the file server for the region. Other file types (e.g. arch and share) are usually mounted.

A mixture of mounting and distributing is possible, e.g. share and arch files could be mounted, and admin files could be distributed. It's usually not worth doing.

What Can't Be Mounted

If the types of files in a package are redefined using the .admin/file-types file, it's possible that the affected files, or possibly the entire package can't be mounted. If the retyped files before retyping have a type that is shared by only a subset of the machines that share files of the new type, then mounting is possible provided that the retyped files are distributed.

On most architectures, a running program that's NFS mounted, and changed on the NFS server, will usually cause a problem. So never NFS mount software that will run as a system daemon. Some examples are:

- the DNS server
- the LPD server
- the XNTPD server

On some architectures, there's no guarantee that NFS mounts will be done before specific system services are started. So even if no daemons are involved, configuration/data files for such services must also reside locally.

What has to be Distributed

When a client mounts all or parts of a package, it is still necessary for one (pseudo) file type of the package to be distributed to the client from the machine from which the software is being mounted. This type is called local_struct, and is a type used only by the distribution mechanism. It represents the directory structure associated with files of type local. It is ordinarily included as part of the package structure when the (pseudo) type structure is distributed. Thus, with the usual defaults for an access-rights file, converting a distribution rule:

```
ClientName   ServerName
```

for a client that doesn't mount packages to a rule for a client that does mount some packages (named p1, p2 etc. in the following) would result in:

```
ClientName   ServerName

ClientName   local_struct=ServerName p=p1 p=p2 ...
```

That is, only the local_struct type is distributed for packages to be mounted, and the usual file types are distributed for packages to be distributed. If most packages are to be mounted, with just some packages distributed, the rules would be of the form:

```
ClientName   local_struct=ServerName
ClientName   ServerName p=p1 p=p2 ...
```

If all the packages on the client are mounted, then the access rights would change to just:

```
ClientName   local_struct=ServerName
```

Where to Mount

In all cases, all regional files must be visible as /.software/regional. So in general, fstab's contain

```
FileServerName:/.software/regional  /.software/regional nfs rw 0 0
```

The "@ServerName" convention described later is never used for regional files.

Client Mounts All Server's Software

In the rare case that the set of packages on a client is intended to be identical to the set of packages on a server, you can mount file types directly onto the usual package locations. For example, for a server named ServerName, a client fstab might contain:

```
ServerName:/.software/share     /.software/share    nfs ro 0 0
ServerName:/.software/arch      /.software/arch     nfs ro 0 0
ServerName:/.software/admin     /.software/admin    nfs ro 0 0
ServerName:/.software/regional  /.software/regional nfs rw 0 0
```

Client Mounts Some of Server's Software

In the more common case of mounting some but not all of a server's packages, the general approach is to mount the server software off to the side, and then create symlinks in the usual package locations for the desired packages. Here's how.

The currently recommended mount point for files of type FileType from a server named ServerName is

```
/.software/FileType@ServerId
```

where ServerId is either ServerName or a name that can be mapped by


```
/software/xhier/config/local/servers
```

into a server name. See man xhier-config-servers for details.

For example, a client fstab might contain:

```
ServerName:/.software/share    /.software/share@ServerName    ...
ServerName:/.software/arch     /.software/arch@ServerName     ...
ServerName:/.software/admin    /.software/admin@ServerName    ...
ServerName:/.software/regional /.software/regional                ...
```

After you have made these mounts (manually, as there is as yet no automation for mounting), distribute the desired packages to the client in the usual manner (such as xh-dist2). The distribution mechanism recognizes this recommended mounting convention and uses the xh-nfs-links command to create for you the appropriate symbolic links from

```
/.software/FileType/PackageName
```

to wherever the corresponding server directory has been mounted. Note that the above mount points don't replace the usual /.software/FileType directories.

Servers for Multiple Architectures

In the unlikely case that a server is of a different architecture than its client, the architecture name typically found in the server's filename, e.g.

```
/.software/arch_SunOS4.1-Sun4
```

is dropped from the mount point name as it's redundant, given that there's no point in mounting files of an incom-

patible architecture type. Thus an fstab entry like this
(for the single architecture case):

```
ServerName:/.software/arch     /.software/arch@server     nfs ro 0 0
```

would be replaced by this:

```
ServerName:/.software/arch_SunOS4.1-Sun4
                        /.software/arch@server     nfs ro 0 0
```

You wouldn't use:

```
ServerName:/.software/arch_SunOS4.1-Sun4
                    /.software/arch_SunOS4.1-Sun4     nfs ro 0 0
```

Obsolete (but pervasive) Conventions

For historical reasons there are several conventions for mount points. It is intended that they will be abandoned in favour of the convention documented above. The xh-nfs-links command handles all of the following conventions mentioned in the following FILES section:

```
/.software/.TypeServerId/
/nfs/ServerName/software/
/nfs/ServerName/.software/Type/
/nfs/ServerName/.software/arch_ArchType/
```

FILES

/.software/Type/
- files of type Type.

/.software/Type@ServerId/
- files of type Type, mounted from the server identified by ServerId.

SEE ALSO

xhier(7) - description of the xhier software organization.
xh-nfs-links(8) - makes links to mounted software.
xhier-file-types(7) - types of files in a package
xh-dist-hosts(8) - queries access rights to local_struct files.
access-rights(5) - configuration for what's allowed to be distributed.
xhier-config-servers(5) - maps a server id into a hostname.
xh-arch(8) - provides the name of the current architecture.
xhier-arch-types(7) - names of architectures.