Building XFree86 from a Source Distribution

David Dawes, Matthieu Herrb

23 February 2004

Abstract

This document describes how to build XFree86 from the **source** distribution and is designed to be used in conjunction with the operating system (OS) specific README files.

NOTE: Refer to the appropriate OS-specific README file before attempting to build XFree86. These files often contain additional information that you need to successfully build for your OS.

We highly recommend using gcc to build XFree86, but XFree86 also generally builds with the native compiler for each OS platform;

1. How to get the XFree86 4.4.0 source

The recommended way of getting the XFree86 4.4.0 source is to obtain it directly from the XFree86 CVS repository. There are several ways of doing that, and they are described at our CVS web page `<URL:http://www.xfree86.org/cvs/>` The CVS tag for this release is "`xf-4_4_0`". The tag for the maintenance branch for this release is "`xf-4_4-branch`".

Another method of getting the XFree86 4.4.0 source is to either download the 4.4.0 source tarballs from the XFree86 ftp site. The procedure for this is as follows:

- The XFree86 4.4.0 source is contained in the files:

```
XFree86-4.4.0-src-1.tgz

XFree86-4.4.0-src-2.tgz

XFree86-4.4.0-src-3.tgz

XFree86-4.4.0-src-4.tgz

XFree86-4.4.0-src-5.tgz

XFree86-4.4.0-src-6.tgz

XFree86-4.4.0-src-7.tgz
```

These can be found at ftp://ftp.xfree86.org/pub/XFree86/4.4.0/source/ and similar locations on XFree86 mirror sites. `XFree86-4.4.0-src-4.tgz` and `XFree86-4.4.0-src-5.tgz` contains the fonts. `XFree86-4.4.0-src-6.tgz` contains the documentation source. `XFree86-4.4.0-src-7.tgz` contains the hardcopy documentation. `XFree86-4.4.0-src-1.tgz`, `XFree86-4.4.0-src-2.tgz` and `XFree86-4.4.0-src-3.tgz` contains everything else. If you don't need the docs or fonts you can get by with only `XFree86-4.4.0-src-1.tgz`, `XFree86-4.4.0-src-2.tgz` and `XFree86-4.4.0-src-3.tgz`.

- Extract each of these files by running the following from a directory on a filesystem containing enough space (the full source requires around 305MB, and a similar amount is required in addition to this for the compiled binaries):

```
gzip -d < XFree86-4.4.0-src-1.tgz | tar vxf -

gzip -d < XFree86-4.4.0-src-2.tgz | tar vxf -

gzip -d < XFree86-4.4.0-src-3.tgz | tar vxf -

gzip -d < XFree86-4.4.0-src-4.tgz | tar vxf -

gzip -d < XFree86-4.4.0-src-5.tgz | tar vxf -

gzip -d < XFree86-4.4.0-src-6.tgz | tar vxf -

gzip -d < XFree86-4.4.0-src-7.tgz | tar vxf -
```

Alternatively, if you already have a pristine copy of the XFree86 4.3.0 source, you can download patches from ftp://ftp.xfree86.org/pub/XFree86/4.4.0/patches/ that will allow you to convert it to 4.4.0. Information about which patch files to download and how to apply them can be found in the "How to get XFree86" section of the README for this release.

All methods will produce one main source directory called `xc`.

2. Configuring the source before building

In most cases it shouldn't be necessary to configure anything before building.

If you do want to make configuration changes, it is recommended that you start by going to the `xc/config/cf` directory, and copying the file `xf86site.def` to `host.def`. Then read through the `host.def` file (which is heavily commented), and set your configuration parameters. Usually you can find the default settings by checking the `.cf` file(s) relevant to your OS.

A good rule to follow is only to change things that you understand as it's easy to create build problems by changing the default configuration. Check the configuration parameters specified in the `xc/config/cf/README`.

If you are using just the `XFree86-4.4.0-src-1.tgz`, `XFree86-4.4.0-src-2.tgz` and `XFree86-4.4.0-src-3.tgz` parts of the source dist, you will need to define **BuildFonts** to **NO**.

3. Using a shadow directory of symbolic links for the build

A recommended practice is to use a shadow directory of symbolic links to do the build of

XFree86 as this allows you to keep the source directory unmodified during the build. It has the following benefits:

- When you are using CVS to maintain your source tree, the update process is not disturbed by foreign files not under CVS's control.
- It is possible to build XFree86 for several different Operating System or architectures from the same sources, shared by read-only NFS mounts.
- It is possible to build XFree86 with different configuration options, by putting a real copy of the `host.def` file in each build tree and by customizing it separately in each build tree.

To make a shadow directory of symbolic links, use the following steps:

- create the directory at the top of the build tree. It is often created at the same level that the `xc` directory, but this is not mandatory.

  `cd` *the directory containing the* xc*directory*

  mkdir build

- use the "`lndir`" command to make the shadow tree:

  ```
  lndir ../xc
  ```

  Note that you can refer to the `xc` directory with an absolute path if needed.

  See the lndir(1) manual page for details.

If `lndir` is not already installed on your system, you can build it manually from the XFree86 sources by running the following commands:

```
cd xc/config/util

make -f Makefile.ini lndir
```

`cp lndir` *some directory in your PATH*

Occasionally there may be stale links in the build tree, like when files in the source tree are removed or renamed. These can be cleaned up by running the "`cleanlinks`" script from the build directory (see the cleanlinks(1) manual page). Rarely there will be changes that will require the build tree to be re-created from scratch. A symptom of this can be mysterious build problems. The best solution for this is to remove the build tree, and then re-create it using the steps outlined above.

4. Building and installing the distribution

Before building the distribution, read through the OS-specific `README` file in `xc/programs/Xserver/hw/xfree86/doc` that is relevant to you. Once you have addressed the OS-specific details, go your build directory (either the `xc` directory or the shadow tree created before) and run "`make World`" with the **BOOTSTRAPCFLAGS** set as described in the OS-specific README (if necessary, but most systems supported by XFree86 don't need **BOOTSTRAPCFLAGS**). It is advisable to redirect stdout and stderr to `World.Log` so that you can track down problems that might occur during the build.

With Bourne-like shells (Bash, the Korn shell, `zsh`, etc.) use a command like:

```
make World > World.log 2>&1
```

Witch C-shell variants (`csh`, `tcsh`, etc), use:

```
make World >& World.log
```

You can follow the progress of the build by running:

```
tail -f World.log
```

in a terminal.

When the build is finished, you should check the `World.Log` file to see if there were any problems. If there weren't any then you can install the binaries. By default the "make World" process will ignore errors and build as much as possible. If there were problems and they are not corrected at this stage, the installation process will fail. To restart the build process after correcting the problems, just run 'make'. If Imakefiles or part of the build configuration was changed as part of correcting the problem, either re-run "make World", or run "make Everything".

If you would prefer "make World" to exit at the first error, run it in the following way instead of the way described above:

for Bourne-like shells:

```
make WORLDOPTS= World > World.log 2>&1
```

for C-shell variants:

```
make WORLDOPTS= World >& World.log
```

To do the install, run "`make install`" and "`make install.man`". Make sure you have enough space in `/usr/X11R6` for the install to succeed. If you want to install on a filesystem other than `/usr`, make a symbolic link to `/usr/X11R6` before installing.

5. Reconfiguring the server (source distribution)

To build a different set of servers or servers with a different set of drivers installed:

1. Make sure the source for any new drivers is in the correct place (e.g., driver source should be in a subdirectory of `xc/programs/Xserver/hw/xfree86/drivers`).
2. Change the settings of the server defines in `host.def` to specify which servers you wish to build. Also, change the driver lists to suit your needs.
3. From `xc/programs/Xserver`, run:

```
make Makefile
make Makefiles
make includes
make depend
make
```

6. Other useful make targets

There are some other useful targets defined in the top level `Makefile` of XFree86:

- **Everything** after a `make World`, `make Everything` does everything a `make World` does, except the cleaning of the tree. It is a way to quickly rebuild the tree after a source patch, but it is not 100% bullet proof. There are cases were it is better to force a full build by using `make World`.

- **clean** does a partial cleaning of the source tree. Removes object files and generated manual pages, but leaves the Makefiles and the generated dependencies files in place. After a `make clean` you need to re-run

  ```
  make includes
  make depend
  make
  ```

  to rebuild the XFree86.
- **distclean** does a full cleaning of the source tree, removing all generated files. After a `make distclean`, `make World` is the only option to rebuild XFree86.
- **includes** generates all generated header files and in-tree symbolic links needed by the build. These files are removed by a `make clean`.
- **depend** recomputes the dependencies for the various targets in all Makefiles. Depending on the operating system, the dependencies are stored in the Makefile, or as a separate file, called `.depend`. This target needs the generated include files produced by `make includes`.
- **VerifyOS** displays the detected operating system version. If the numbers shown do not match your system, you probably need to set them manually in `host.def` and report the problem to <XFree86@XFree86.org>.

CONTENTS

$XFree86: xc/programs/Xserver/hw/xfree86/doc/sgml/BUILD.sgml,v 3.14 2004/02/24 03:41:39 dawes Exp $