Remote X Apps mini−HOWTO

Table of Contents

Remote X Apps mini−HOWTO

Vincent Zweije, zweije@xs4all.nl

v0.7.5, 8 December 2001

This mini−HOWTO describes how to run remote X applications. That is, how to have an X program display on a different computer than the one it's running on. Or conversely: how to make an X program run on a different computer than the one you're sitting at. The focus of this mini−HOWTO is on security. This mini−HOWTO also contains information on running X applications locally, but with a different user−id, and information on setting up a computer as an X terminal.

---

10. Troubleshooting

---

1. Introduction

This mini−HOWTO is a guide how to do remote X applications. It was written for several reasons.

1. Many questions have appeared on usenet on how to run a remote X application.
2. I see many, many hints of ``use `xhost +hostname`'' or even ``xhost +'' to allow X connections. **This is ridiculously insecure**, and there are better methods.
3. I do not know of a simple document that describes the options you *do* have. Please inform me zweije@xs4all.nl if you know more.

This document has been written with unix−like systems in mind. If either your local or remote operating system are of another flavour, you may find here how things work. However, you will have to translate examples yourself to apply to your own system(s).

The most recent version of this document is always available on WWW at http://www.xs4all.nl/~zweije/xauth.html. It is also available as the Linux Remote X Apps mini−HOWTO at http://sunsite.unc.edu/LDP/HOWTO/mini/Remote−X−Apps. Linux (mini−)HOWTOs are available by http or ftp from sunsite.unc.edu.

This is version 0.7.5. No guarantees, only good intentions. I'm open to suggestions, ideas, additions, useful pointers, (typo) corrections, etc... I want this to remain a simple readable document, though, in the best−meant HOWTO style. Flames to `/dev/null`. This document is released under version 1.1 of the GNU Free Documentation Licence.

Contents last updated on 8 December 2001 by Vincent Zweije

---

2. Related Reading

A related document on WWW is ``What to do when Tk says that your display is insecure'', http://ce−toolkit.crd.ge.com/tkxauth/. It was written by Kevin Kenny. It suggests a similar solution to X authentication to that in this document (xauth). However, Kevin aims more at using xdm to steer xauth for you.

The X System Window System Vol. 8 ``X Window System Administrator's Guide'' from O'Reilly and Associates has also been brought to my attention and confirmed as a good source of information. However, it has not been revised since its original publication in 1992. As such it only covers X11R4 and X11R5, anything specific to X11R6 will not be covered.

Yet another document much like the one you're reading now, titled ``Securing X Windows'', is available at http://ciac.llnl.gov/ciac/documents/ciac2316.html.

Also check out usenet newsgroups, such as `comp.windows.x`, `comp.os.linux.x`, and `comp.os.linux.networking`.

---

3. The Scene

You're using two computers. You're using the X window system of the first to type to and look at. You're using the second to do some important graphical work. You want the second to show its output on the display of the first. The X window system makes this possible.

Of course, you need a network connection for this. Preferably a fast one; the X protocol is a network hog. But with a little patience and suitable protocol compression, you can even run applications over a modem. For X protocol compression, you might want to check out dxpc http://www.vigor.nu/dxpc/ or LBX http://www.paulandlesley.org/faqs/LBX−HOWTO.html (also known as the LBX mini−HOWTO).

You must do two things to achieve all this:

1. Tell the local display (the server) to accept connections from the remote computer.
2. Tell the remote application (the client) to direct its output to your local display.

---

4. A Little Theory

The magic word is `DISPLAY`. In the X window system, a display consists (simplified) of a keyboard, a mouse and a screen. A display is managed by a server program, known as an X server. The server serves displaying capabilities to other programs that connect to it.

A display is indicated with a name, for instance:

- `DISPLAY=light.uni.verse:0`
- `DISPLAY=localhost:4`
- `DISPLAY=:0`

The display consists of a hostname (such as `light.uni.verse` and `localhost`), a colon (`:`), and a sequence number (such as `0` and `4`). The hostname of the display is the name of the computer where the X server runs. An omitted hostname means the local host. The sequence number is usually 0 −− it can be varied if there are multiple displays connected to one computer.

If you ever come across a display indication with an extra `.n` attached to it, that's the screen number. A display can actually have multiple screens. Usually there's only one screen though, with number `n=0`, so that's the default.

Other forms of `DISPLAY` exist, but the above will do for our purposes.

For the technically curious:

- `hostname:D.S` means screen `S` on display `D` of host `hostname`; the X server for this display is listening at TCP port `6000+D`.
- `host/unix:D.S` means screen `S` on display `D` of host `host`; the X server for this display is listening at UNIX domain socket `/tmp/.X11−unix/XD` (so it's only reachable from `host`).
- `:D.S` is equivalent to `host/unix:D.S`, where `host` is the local hostname.

---

5. Telling the Client

The client program (for instance, your graphics application) knows which display to connect to by inspecting the `DISPLAY` environment variable. This setting can be overridden, though, by giving the client the command line argument `−display hostname:0` when it's started. Some examples may clarify things.

Our computer is known to the outside as light, and we're in domain uni.verse. If we're running a normal X server, the display is known as `light.uni.verse:0`. We want to run the drawing program xfig on a remote computer, called `dark.matt.er`, and display its output here on light.

Suppose you have already telnetted into the remote computer, `dark.matt.er`.

If you have csh running on the remote computer:

```
dark% setenv DISPLAY light.uni.verse:0
dark% xfig &
```

or alternatively:

```
dark% xfig −display light.uni.verse:0 &
```

If you have sh running on the remote computer:

```
dark$ DISPLAY=light.uni.verse:0
dark$ export DISPLAY
dark$ xfig &
```

or, alternatively:

```
dark$ DISPLAY=light.uni.verse:0 xfig &
```

or, of course, also:

```
dark$ xfig −display light.uni.verse:0 &
```

It seems that some versions of telnet automatically transport the `DISPLAY` variable to the remote host. If you have one of those, you're lucky, and you don't have to set it by hand. If not, most versions of telnet do transport the `TERM` environment variable; with some judicious hacking it is possible to piggyback the `DISPLAY` variable on to the `TERM` variable.

The idea with piggybacking is that you do some scripting to achieve the following: before telnetting, attach the value of `DISPLAY` to `TERM`. Then telnet out. At the remote end, in the applicable `.*shrc` file, read the value of `DISPLAY` from `TERM`.

6. Telling the Server

The server will not accept connections from just anywhere. You don't want everyone to be able to display windows on your screen. Or read what you type −− remember that your keyboard is part of your display!

Too few people seem to realise that allowing access to your display poses a security risk. Someone with access to your display can read and write your screens, read your keystrokes, and read your mouse actions.

Most servers know two ways of authenticating connections to it: the host list mechanism (xhost) and the magic cookie mechanism (xauth). Then there is ssh, the secure shell, that can forward X connections.

Notice that some X servers (from XFree86) can be configured not to listen on the usual TCP port with the `−nolisten tcp` argument. Notably the default configuration of Debian GNU/Linux is to disable the X server listening on the TCP port. If you wish to use remote X on a Debian system, you should re−enable this by altering the way the X server is started. Look at `/etc/X11/xinit/xserverrc` for a start.

6.1 Xhost

Xhost allows access based on hostnames. The server maintains a list of hosts which are allowed to connect to it. It can also disable host checking entirely. Beware: this means no checks are done, so *every* host may connect!

You can control the server's host list with the xhost program. To use this mechanism in the previous example, do:

```
light$ xhost +dark.matt.er
```

This allows all connections from host `dark.matt.er`. As soon as your X client has made its connection and displays a window, for safety, revoke permissions for more connections with:

```
light$ xhost −dark.matt.er
```

You can disable host checking with:

```
light$ xhost +
```

This disables host access checking and thus allows *everyone* to connect. You should *never* do this on a network on which you don't trust *all* users (such as Internet). You can re−enable host checking with:

```
light$ xhost −
```

xhost − by itself does *not* remove all hosts from the access list (that would be quite useless − you wouldn't be able to connect from anywhere, not even your local host).

Xhost is a very insecure mechanism. It does not distinguish between different users on the remote host. Also, hostnames (addresses actually) can be spoofed. This is bad if you're on an untrusted network (for instance already with dialup PPP access to Internet).

6.2 Xauth

Xauth allows access to anyone who knows the right secret. Such a secret is called an authorization record, or a magic cookie. This authorization scheme is formally called MIT−MAGIC−COOKIE−1.

The cookies for different displays are stored together in `~/.Xauthority`. Your `~/.Xauthority` must be inaccessible for group/other users. The xauth program manages these cookies, hence the nickname xauth for the scheme.

You can specify a different cookie file with the `XAUTHORITY` environment variable, but you will rarely need this. If you're not sure which cookie file your xauth is using, do an `xauth −v`, and it will tell you.

On starting a session, the server reads a cookie from the file that is indicated by the `−auth` argument. After that, the server only allows connections from clients that know the same cookie. When the cookie in `~/.Xauthority` changes, *the server will not pick up the change*.

Newer servers can generate cookies on the fly for clients that ask for it. Cookies are still kept inside the server though; they don't end up in `~/.Xauthority` unless a client puts them there. According to David Wiggins:

> A further wrinkle was added in X11R6.3 that you may be interested in. Via the new SECURITY extension, the X server itself can generate and return new cookies on the fly. Furthermore, the cookies can be designated ``untrusted'' so that applications making connections with such cookies will be restricted in their operation. For example, they won't be able to steal keyboard/mouse input, or window contents, from other trusted clients. There is a new ``generate'' subcommand to xauth to make this facility at least possible to use, if not easy.

Xauth has a clear security advantage over xhost. You can limit access to specific users on specific computers. It does not suffer from spoofed addresses as xhost does. And if you want to, you can still use xhost next to it to allow connections.

Making the Cookie

If you want to use xauth, you must start the X server with the `−auth authfile` argument. If you use the startx script, that's the right place to do it. Create the authorization record as below in your startx script.

Excerpt from `/usr/X11R6/bin/startx`:

```
mcookie|sed −e 's/^/add :0 . /'|xauth −q
xinit −− −auth "$HOME/.Xauthority"
```

Mcookie is a tiny program in the util−linux package, primary site ftp://ftp.math.uio.no/pub/linux/. Alternatively, you can use md5sum to massage some random data (from, for instance, `/dev/urandom` or `ps −axl`) into cookie format:

```
dd if=/dev/urandom count=1|md5sum|sed −e 's/^/add :0 . /'|xauth −q
xinit −− −auth "$HOME/.Xauthority"
```

If you can't edit the startx script (because you aren't root), get your system administrator to set up startx properly, or let him set up xdm instead. If he can't or won't, you can make a `~/.xserverrc` script. If you have this script, it is run by xinit instead of the real X server. Then you can start the real X server from this script with the proper arguments. To do so, have your `~/.xserverrc` use the magic cookie line above to create a cookie and then exec the real X server:

```
#!/bin/sh
mcookie|sed −e 's/^/add :0 . /'|xauth −q
exec /usr/X11R6/bin/X "$@" −auth "$HOME/.Xauthority"
```

If you use xdm to manage your X sessions, you can use xauth easily. Define the DisplayManager.authDir resource in `/etc/X11/xdm/xdm−config`. Xdm will pass the `−auth` argument to the X server when it starts. When you then log in under xdm, xdm puts the cookie in your `~/.Xauthority` for you. See xdm(1) for more information. For instance, my `/etc/X11/xdm/xdm−config` has the following line in it:

```
DisplayManager.authDir: /var/lib/xdm
```

Transporting the Cookie

Now that you have started your X session on the server host `light.uni.verse` and have your cookie in `~/.Xauthority`, you will have to transfer the cookie to the client host, `dark.matt.er`. There are several ways to do this.

Shared Home Directories

The easiest is when your home directories on light and dark are shared. The `~/.Xauthority` files are the same, so the cookie is transported instantaneously. However, there's a catch: when you put a cookie for `:0` in `~/.Xauthority`, dark will think it's a cookie for itself instead of for light. You must use an explicit host name when you create the cookie; you can't leave it out. You can install the same cookie for both `:0` and `light:0` with this little piece of sed wizardry:

```
#!/bin/sh
mcookie|sed −e 's/^/add :0 . /' −e p −e "s/:/$HOST&/"|xauth −q
exec /usr/X11R6/bin/X "$@" −auth "$HOME/.Xauthority"
```

By the Remote Shell, `rsh`

If the home directories aren't shared, you can transport the cookie by means of rsh, the remote shell:

```
light$ xauth nlist "${HOST}:0" | rsh dark.matt.er xauth nmerge −
```

1. Extract the cookie from your local `~/.Xauthority` (`xauth nlist :0`).
2. Transfer it to dark.matt.er (`| rsh dark.matt.er`).
3. Put it in the `~/.Xauthority` there (`xauth nmerge −`).

Notice the use of `${HOST}`. You need to transport the cookie that is explicitly associated with the local host. A remote X application would interpret a display value of `:0` as referring to the remote machine, which is not what you want!

Manually, by Telnet

It's possible that rsh doesn't work for you. Besides that, rsh also has a security drawback (spoofed host names again, if I remember correctly). If you can't or don't want to use rsh, you can also transfer the cookie manually, like:

```
light$ echo $DISPLAY
:0
light$ xauth list $DISPLAY
light/unix:0 MIT−MAGIC−COOKIE−1 076aaecfd370fd2af6bb9f5550b26926
light$ rlogin dark.matt.er
Password:
dark% setenv DISPLAY light.uni.verse:0
```

```
dark% xauth
Using authority file /home/zweije/.Xauthority
xauth> add light.uni.verse:0 . 076aaecfd370fd2af6bb9f5550b26926
xauth> exit
Writing authority file /home/zweije/.Xauthority
dark% xfig &
[15332]
dark% logout
light$
```

See also rsh(1) and xauth(1x) for more information.

Automating the Telnet Way

It may be possible to piggyback the cookie on the `TERM` or `DISPLAY` variable when you do a telnet to the remote host. This would go the same way as piggybacking the `DISPLAY` variable on the `TERM` variable. See section 5: Telling the Client. You're on own here from my point of view, but I'm interested if anyone can confirm or deny this.

Notice, however, that environment variables can be observed by others on some unices, and you won't be able to prevent the cookie in `$TERM` from showing up if people are looking for it.

Using the Cookie

An X application on dark.matt.er, such as xfig above, will automatically look in `~/.Xauthority` there for the cookie to authenticate itself with.

There's a little wrinkle when using `localhost:D`. X client applications translate `localhost:D` into `host/unix:D` for the purpose of cookie retrieval. Effectively, this means that a cookie for `localhost:D` in your `~/.Xauthority` has *no* effect.

If you think about it, it's only logical. The interpretation of `localhost` depends entirely on the machine on which it's interpreted. It would give a horrible mess when you have a shared home directory, such as through NFS, with several hosts all interfering with each other's cookies.

6.3 Ssh

Authority records are transmitted over the network with no encryption. If you're even worried someone might snoop on your connections, use ssh, the secure shell. It can do X forwarding over encrypted connections.

To turn on X forwarding over ssh, use the command line switch `−X` or write the following in your local ssh configuration file:

```
Host remote.host.name
    ForwardX11 yes
```

The ssh server (`sshd`) at the remote end automatically sets `DISPLAY` to point to its end of the X forwarding tunnel. The remote tunnel end gets its own cookie; the remote ssh server generates it for you and puts it in `~/.Xauthority` there. So, X authorisation with ssh is fully automatic.

By the way, ssh is great in other ways too. It's a good structural improvement to your system. For more information, visit http://www.ssh.org/, the ssh home page.

Who knows anything else on authentication schemes or encrypting X connections? Maybe kerberos?

---

7. X Applications from Another User−id

Suppose you want to run a graphical configuration tool that requires root privileges. However, your X session is running under your usual account. It may seem strange at first, but the X server will *not* allow the tool to access your display. How is this possible when root can normally do anything? And how do you work around this problem?

Let's generalise to the situation where you want to an X appliation under a user−id `clientuser`, but the X session was started by `serveruser`. If you have read the section on cookies, it is clear why `clientuser` cannot access your display: `~clientuser/.Xauthority` does not contain the right magic cookie for accessing the display. The right cookie is found in `~serveruser/.Xauthority`.

7.1 Different Users on the Same Host

Of course, anything that works for remote X also works for X from a different user−id as well (particularly `slogin localhost −l clientuser`). It's just that the client host and the server host happen to be the same. However, when both hosts are the same, there are some shortcuts for transferring the magic cookie.

We'll assume that you use `su` to switch user−ids. Basically, what you have to do is write a script that will call `su`, but wraps the command that `su` executes with some code that does the necessary things for remote X. These necessary things are setting the `DISPLAY` variable and transferring the magic cookie.

Setting `DISPLAY` is relatively easy; it just means defining `DISPLAY="$DISPLAY"` before running the su command argument. So you could just do:

```
su − clientuser −c "env DISPLAY=$DISPLAY clientprogram &"
```

This doesn't work yet, because we still have to transfer the cookie. We can retrieve the cookie using `xauth list "$DISPLAY"`. This command happens to list the cookie in a format that's suitable for feeding back to the `xauth add` command; just what we need!

We shall want to pass the cookie through a pipe. Unfortunately, it isn't easy to pass something through a pipe to the `su` command, because `su` wants to read the password from its standard input. Fortunately again, in a shell script we can joggle some file descriptors around, and get it done.

So we write a script around this, parameterizing by `clientuser` and `clientprogram`. Let's improve the script a little while we're at it, making it less readable but more robust. It looks like this:

```
#!/bin/sh

if [ $# −lt 2 ]
then echo "usage: `basename $0` clientuser command" >&2
     exit 2
fi

CLIENTUSER="$1"
shift

# FD 4 becomes stdin too
```

```
exec 4>&0

xauth list "$DISPLAY" | sed -e 's/^/add /' | {

    # FD 3 becomes xauth output
    # FD 0 becomes stdin again
    # FD 4 is closed
    exec 3>&0 0>&4 4>&-

    exec su - "$CLIENTUSER" -c \
         "xauth -q <&3
          exec env DISPLAY='$DISPLAY' "'"$SHELL"'" -c '$*' 3>&-"

}
```

I think this is portable and works well enough in most circumstances. The only shortcoming I can think of right now is that, due to using `'$*'`, single quotes in `command` will mess up quoting in the `su` command argument (`'$*'`). If there's anything else seriously wrong with it, please drop me an email.

Call the script `/usr/local/bin/xsu`, and you can do:

```
xsu clientuser 'command &'
```

Can't be much easier, unless you get rid of the password. Yes, there are ways for that too (`sudo`), but this is not the place for that.

The tiny `xsu` script just mentioned has served as the basis for a more extended script called `sux` which apparently has found its way as a package into the Debian distribution.

7.2 Client User Is Root

Obviously, anything that works for non-root client users is going to work for root as well. However, with root you can make it even easier, because root can read anyone's `~/.Xauthority` file. There's no need to transfer the cookie. All you have to do is set `DISPLAY`, and point `XAUTHORITY` to `~serveruser/.Xauthority`. So you can do:

```
su - -c "exec env DISPLAY='$DISPLAY' \
                  XAUTHORITY='${XAUTHORITY-$HOME/.Xauthority}' \
                  command"
```

Putting it into a script would give something like:

```
#!/bin/sh
if [ $# -lt 1 ]
then echo "usage: `basename $0` command" >&2
     exit 2
fi
su - -c "exec env DISPLAY='$DISPLAY' \
                  XAUTHORITY='${XAUTHORITY-$HOME/.Xauthority}' \
                  "'"$SHELL"'" -c '$*'"
```

Call the script `/usr/local/bin/xroot`, and you can do:

```
xroot 'control-panel &'
```

Although, if you've set up `xsu` already, there's no real reason to do this.

---

8. Running a Remote Window Manager

A window manager (like `twm`, `wmaker`, or `fvwm95`) is an application like any other. The normal procedure should work.

Well, almost. At most one window manager can be running on a display at any time. If you are already running a local window manager, you cannot start the remote one (it will complain and exit). You have to kill (or simply quit) the local one first.

Unfortunately, many X session scripts end with an

```
exec window−manager−of−choice
```

and this means that when the (local) window manager exits, your session exits, and the X system (xdm or xinit) considers your session over and effectively logs you out.

You have to jump through a few extra hoops, but it can be done and it's not too difficult. Just play with your session script (normally `~/.xsession` or `~/.xinitrc`) to get it as you want it.

Beware that a window manager often provides ways to run new programs, and that these will run on the local machine. That is, local to where the window manager runs. If you run a remote window manager, it will spawn remote applications, and this may not be what you want. Of course, they still display on the display that is local to you.

---

9. Setting Up an X Terminal

Make use of your old PC! Turn it into an extra work place! No need for buying expensive new hardware! You've already got all it takes!

Seriously, you can set up an old PC as an X terminal. An X terminal is a computer that basically runs nothing but an X server. You can log in on it, and get an X session, with xterms, xbiff, xclock, every other conceivable X client. However, all clients are running on a remote host, and are using remote X to display their output on your local X terminal. Even the window manager is running remotely.

An X terminal takes very few resources, compared to a full blown unix machine. Over here I have an X terminal with a 486 CPU, 16M of RAM, and 250M of disk space. Oh, and a network connection, of course. It doesn't even have user home directories.

For some related reading, have a look at:

- The *XDM and X Terminal mini−HOWTO* (http://www.ibiblio.org/pub/Linux/docs/HOWTO/mini/other−formats/html_single/XDM−Xterm.html). This document is an extensive description of what is possible with XDMCP and xdm, applied for building X terminals. Definitely have a look at this.
- The *XDMCP HOWTO* (

http://www.ibiblio.org/pub/Linux/docs/HOWTO/other−formats/html_single/XDMCP.html). This document describes the steps necessary to set up xdm for use with remote X servers, such as from X terminals. The setup of the X server in such a situation is described less extensively.
- The *Xterminal mini−HOWTO* (http://metalab.unc.edu/pub/Linux/docs/HOWTO/unmaintained/mini/Xterminal). It is currently unmaintained, but it might contain some useful information for you.

Contrasted to the above documents, this document (the Remote X Apps mini−HOWTO) limits itself to a short description of XDMCP, but puts more emphasis on the security issues involved.

9.1 Once More, a Little Theory First

As far as X is concerned, the X terminal will be running nothing but an X server. This X server will be configured to talk to a remote host using XDMCP (the X Display Manager Control Protocol). It will ask the remote host for an X session. The remote host will put up a login window on the X terminal, and after login it will run an X session with all bells and whistles, including the window manager, all using remote X to display on the X terminal.

You will probably notice that the remote host is acting like a server, though not an X server. The remote host is providing X sessions to X servers that ask for one. So, with respect to XDMCP, the remote host is actually a server, providing X sessions, also known as an XDMCP server. The X server is playing the role of an XDMCP client! Are you still with me?

The program that provides the XDMCP service on the XDMCP server is `xdm`. So, in order to get an X terminal up and running, you must configure two programs: `X` (the XDMCP client) on the X terminal, and xdm (the XDMCP server) on the remote host.

You must always remember that the X protocol (and the XDMCP protocol) are not encrypted. If you use remote X, everything that goes over the network can be sniffed by other hosts on the network. This is especially bad with remote X sessions, since the first thing that happens is logging in by giving a username and password. So, you must run remote X over a trusted network only!

9.2 Configuring X as an XDMCP Client

If you want to set up a Linux machine as an X terminal, you need very few resources. Basically, you need what it takes to get a bare bones Linux machine running, plus an X server. Specifically, you do *not* need the X clients and libraries. It can be useful to install some X fonts, but you can also use a font server somewhere on the network.

There are a few ways for an X server to get an X session from an XDMCP server. The simplest one is to go straight to a known XDMCP server and ask for one. Alternatively, the X server can broadcast a request for an XDMCP service and use the first XDMCP server that responds. Lastly, the X server can go to an XDMCP server, ask it for a list of hosts willing to provide a session, and let the user choose a session host.

1. When you know the host that is going to provide you with sessions, go straight to it. Run
   ```
   X −query sessionhost
   ```
   and, assuming `xdm` is running on `sessionhost`, you'll get a login window, and after login, an X session.
2. When you don't really care on which host you're getting your session, use the broadcast method. Run

```
X −broadcast
```

   and, assuming `xdm` is running somewhere on the network, you'll get a login window from the first (and hopefully quickest) `xdm` that responds, and after login, an X session.

3. When you want to choose the host where you want to have your session, ask an XDMCP server for a list. Run

   ```
   X −indirect xdmcpserver
   ```

   and, assuming `xdm` is configured right there, you'll be presented a list of hosts to choose from. Choose one; you'll get the login window for that host, and after login, the session you were looking for.

You may have noticed the absence of the `−auth` option. The X server will use XDMCP to negotiate a magic cookie with the XDMCP server. The XDMCP server will put the cookie in your remote `~/.Xauthority` after login.

After a session is over, the X server will loop and go back to the original XDMCP server and ask for another session (or chooser list). If you don't want that, you can use the `−once` option. Note: this doesn't seem to work with the `−indirect` option due to the implementation of the chooser.

When you have determined the way in which you want to run the X server, you can also put it in a startup script, or even run it straight from `/etc/inittab`. Please consult your own distribution's documentation for how to modify your startup scripts or `/etc/inittab`.

Do *not* run an X server like this from the `Xservers` configuration file. `xdm` expects to be able to connect to such servers, and may kill them if it can't connect.

9.3 Configuring `xdm` as an XDMCP Server

The program that provides the XDMCP service (the session service) is usually `xdm`. There are variants of this such as `wdm` or `gdm` on Linux, but these basically work the same way. So, make sure `xdm` or variant is installed on the host where you want to run your X sessions. If you've got a local graphical login on the X session host, `xdm` is already installed; most Linux distributions come that way these days.

In addition to `xdm`, you will need the programs that you wish to be able to run in an X session. That is, all X clients like `xterm`, `xfig`, `xclock`, window managers and all that. However, for an XDMCP server, you do *not* have to install an X server; the X server will be running on the X terminal instead.

From the X server story above, you can conclude that there are basically two kinds of XDMCP service. There is the *direct* service, consisting of letting an XDMCP client log in, and then providing it with an X session. Alternatively, there is the *indirect* service, in which an XDMCP client is provided with a list of hosts, providing a direct service, to choose from.

All `xdm` services are configured in the access file, generally located at `/etc/X11/xdm/Xaccess` or a similar location. This location is actually defined in the general `xdm` configuration file `/etc/X11/xdm/xdm−config`, through the `accessFile` resource. See your `xdm` manual for the default location.

1. If you want to allow `xdm` to provide connecting XDMCP clients with an X session, whether by broadcast or not, you put the host name of the XDMCP client (the X server, remember?) by itself on

a line in `Xaccess`. Actually, you can put a pattern on the line matching multiple hosts. Here are some valid patterns:

```
xterm023.my.domain      # xterm023.my.domain can get an X session
*.my.domain             # any host in my.domain can get an X session
*                       # any host on Internet can get an X session (unsafe)
```

Whether you should want to provide any host in Internet with an X session is arguable. Obviously, any service you provide is one more possible hole in your server's security. On the other hand, the server should be secure itself, and an XDMCP client asking for an X session has to provide a valid authentication before the X session is granted.

Furthermore, the X session uses a remote X connection, which is not encrypted. The username/password pair for the login will be transported on this connection. People out there could be sniffing valid username/password combinations, just as with plain telnet connections. This is even worse then having xauth magic cookies sniffed.

Make your own decisions here, but I recommend not enabling this service to the world unless you have a good reason.

2. If you want to provide XDMCP clients (`X −indirect xdmcpserver`) with a chooser list (a list of hosts to choose from to get an X session), follow the client pattern with the keyword `CHOOSER` and the list of hosts that that client may choose from. Instead of the list of hosts to choose from, you can also specify `BROADCAST`; with this, `xdm` broadcasts on the network to query for servers willing to provide the session. Some valid examples:

```
xterm023.my.domain      CHOOSER seshost1 seshost2
*.my.domain             CHOOSER BROADCAST
*                       CHOOSER extseshost1 extseshost2
```

The first lets `xterm023` choose between sessions on either `seshost1` and `seshost2`. The second lets any host in `my.domain` choose from any host that is willing to provide an X session. The third lets any host out there choose between a session on `extseshost1` or `etsseshost2`.

It is probably not a good idea to do `* CHOOSER BROADCAST`. This will allow hosts outside your network to get information about the hosts inside your network. You probably don't want to pass out such information. In fact, allowing a chooser to any outside host is probably not useful anyway, since you should not be enabling arbitrary direct connections either.

When you have reconfigured `xdm`, send it the `HUP` signal to make it re−read its configuration files.

```
# kill −HUP pid−of−xdm
#
```

9.4 XDMCP Technically

Technically, as far as I can see, XDMCP is not entirely what you would expect from the above description. `xdm` can redirect connecting X servers to another place, and uses this trick to implement the chooser. So, the choosing happens inside xdm, not in the X server, although the chooser list is represented on the X server's display. This is also why the X server's `−once` option does not combine with `−indirect`.

---

10. Troubleshooting

The first time you try to run a remote X application, it usually does not work. Here are a few common error messages, their probable causes, and solutions to help you on your way.

```
xterm Xt error: Can't open display:
```

There is no `DISPLAY` variable in the environment, and you didn't tell the application with the `−display` flag either. The application assumes the empty string, but that is syntactically invalid. To solve this, be sure that you set the `DISPLAY` variable correctly in the environment (with `setenv` or `export` depending on your shell).

```
_X11TransSocketINETConnect: Can't connect: errno = 101
xterm Xt error: Can't open display: love.dial.xs4all.nl:0
```

Errno 101 is ``Network is unreachable''. The application could not make a network connection to the server. Check that you have the correct `DISPLAY` set, and that the server machine is reachable from your client (it should be, after all you're probably logged in to the server and telnetting to the client).

```
_X11TransSocketINETConnect: Can't connect: errno = 111
xterm Xt error: Can't open display: love.dial.xs4all.nl:0
```

Errno 111 is ``Connection refused''. The server machine you're trying to connect to is reachable, but the indicated server does not exist there. Check that you are using the right host name and the right display number.

Alternatively, it is possible that the X server was configured *not* to listen to the usual TCP port. To find out if this is the case, see if the X server is started with the `−nolisten tcp` argument, and if so, remove it.

```
Xlib: connection to ":0.0" refused by server
Xlib: Client is not authorized to connect to Server
xterm Xt error: Can't open display: love.dial.xs4all.nl:0.0
```

The client could make a connection to the server, but the server does not allow the client to use it (not authorized). Make sure that you have transported the correct magic cookie to the client, and that it has not expired (the server uses a new cookie when a new session starts).
