# 4 X displays

## 4.1 The display parameter

- The instructions to X clients on where to display are contained in a parameter called, naturally enough, the "display".

  A typical value for a display would be `griffin.csi.cam.ac.uk:0.0`.

- The display has three components: *hostname*:*server*.*screen*.

  **hostname:** The hostname identifies the actual machine to be sent the X instructions.

  **server:** Machines have one X server running for each attached monitor being used for X. These are numbered from 0 upwards.

  **screen:** Each monitor may have multiple, virtual screens which can be flipped between at the push of a key. These are numbered from 0 upwards.

- Because most machines have a single monitor (and hence a single server) with no facility for mutiple virtual screens the display parameter typically ends `:0.0`.

- If a machine is displaying to its own console monitor then it will often use the display value `:0.0`. The absence of a hostname indicates that the client should display locally, by whatever means it wants and often not using the network at all.

- If your system is set up so that the display is set to `:0.0` and you want to run X clients on a remote machine displaying back to your machine, then you must override the display and replace it with the full *hostname*`:0.0` for that instruction.

- Some displays are set as `unix:0.0`. This should be treated as the same as `:0.0` except that while the latter gives the client some flexibility in how it talks to its own server, the former *requires* it not to use the network.

## 4.2 Telling the client

The client is told the display to use in one of two ways:

**Environment variable:** If the environment variable `DISPLAY` is set, then the client will use its value.

**Command line:** Every X client should support the `-display` flag. Adding the option "`-display` *display-name*" to an X command will set the display value and override the `DISPLAY` evironment variable.

*Note:* If the command `xterm -display machine:0.0` is run, then the shell run in the `xterm` will have `DISPLAY=machine:0.0` in its environment.

## 4.3 X permission mechanisms

Most current implementations of X have two permission mechanisms available to the user: host-based and token-based. The token-based system is more secure.

*Warning:* The default setting for many systems is to have *no* security at all. The obvious problem with this is that anyone can place windows on your display. The more subtle, but potentially more serious, problem is that anyone can take a copy of what is on your screen. *You have no privacy at all if you have no X security.*

### 4.3.1 Host-based permissions

This permission mechanism works by telling the server which machines it should accept connections from. It is controlled by the `xhost` command.

- "`xhost`" on its own will give details of the current host-based security scheme.

```
$ xhost
access control enabled, \
only authorized clients can connect
```

- "`xhost +`" will turn off `X` security checking.

```
$ xhost +
access control disabled, \
clients can connect from any host
```

- "`xhost -`" will turn on the checking.

```
$ xhost -
access control enabled, \
only authorized clients can connect
```

- "`xhost +` *hostname*" will allow connections from the named host.

```
$ xhost + banshee
banshee being added to access control list
$ xhost
access control enabled, \
only authorized clients can connect
banshee.csi.cam.ac.uk
```

- "`xhost -` *hostname*" will disallow connections from the named host.

```
$ xhost
access control enabled, \
only authorized clients can connect
banshee.csi.cam.ac.uk
$ xhost - banshee
banshee being removed from access control list
$ xhost
access control enabled, \
only authorized clients can connect
```

Host-based permissions are of limited use with multi-user machines. The command `xhost + bootes.cus` will permit *any* user of `bootes` (one of the C.U.S. machines) to access the display.

#### 4.3.2 Token-based permissions

A better scheme for multi-user systems (and the scheme recommended by the Computing Service) is based around tokens called "magic cookies".

The server is given one of these cookies that acts as a password for all `X` requests.

By some means (to be described in section 7) the user has a copy of this cookie in a file called `.Xauthority` that only the user can read and all `X` clients the user starts look in this file.

To allow access from another machine the user must pass the cookie to his or her account on that other machine.

The command for manipulating these cookies and the `.Xauthority` file is called `xauth`.

- "`xauth nlist`" will list all the cookies currently held in the user's `.Xauthority` file.
- "`xauth nlist $DISPLAY`" will list just the cookie for the particular display.
- "`xauth nmerge` *filename*" will add cookies to the `.Xauthority` file from file *filename*.
- "`xauth nmerge -`" will add cookies to the `.Xauthority` file from standard input.

The `nlist` and `nmerge` options between them provide the functionality needed to transfer a cookie between a user's accounts on two different machines.

**Example:**
Suppose the user has an account called `bob` on a machine `klingon` which is running an `X` server on its console. The account `bob` has a `.Xauthority` file with the correct cookie in it. The user wants to be able to run `X` clients on the machine `vulcan` displaying to `klingon`'s console. His account on `vulcan`

is called `rjd`. The `rjd` account on `vulcan` trusts the `bob` account on `klingon` (in the `.rhosts` sense).

`bob`, on `klingon`, needs to issue a command like

```
$ xauth list klingon:0.0 | rsh -l rjd vulcan xauth merge -
```

to transfer the cookie from `bob`'s account on `klingon` to `rjd`'s account on `vulcan`.

Note that the display name had to be given as "`klingon:0.0`" rather than "`:0.0`" so that the cookie would be useful on `vulcan`. If the name `:0.0` had been used, `vulcan` would have believed that it was receiving a cookie for its *own* display.

## 4.4 Troubleshooting

So what happens if the display isn't set, or the server refuses permission?

### 4.4.1 No display

- Example:

  ```
  $ xterm &
  Error: Can't open display:
  $
  ```

- No display name after the string "`Can't open display:`"

### 4.4.2 Permission denied

- Example:

```
$ echo ${DISPLAY}
griffin.csi.cam.ac.uk:0.0
$ xterm &
Xlib:  connection to "griffin.csi.cam.ac.uk:0.0" \
refused by server
Xlib:  Client is not authorized to connect to Server
Error: Can't open display: griffin.csi.cam.ac.uk:0.0
$
```

- Display name after the string "`Can't open display:`"

## 4.5 Starting X clients on remote machines

- As the X server and client can run on different machines, a mechanism is needed for starting up X clients on remote machines.
- First attempt:

```
griffin$ rsh banshee xterm &
```

- `rsh` does not pass the `DISPLAY` environment variable so `xterm` does not know where to display
- Second attempt:

```
griffin$ rsh banshee xterm -display ${DISPLAY} &
```

- This works, but the launching `rsh` process never goes away.
- **stdin**, **stdout** and **stderr** are still open on the local machine `griffin`.
- Third attempt:

```
griffin$ rsh banshee xterm -display ${DISPLAY} \
1> /dev/null 2>&1 < /dev/null &
```

- This works and leaves no connections open for `rsh`.
- It is rather clumsy just to run `xterm`.

- **Special command:** `xon`
- Wrapper around `rsh`

```
griffin:~$ xon banshee xterm
```

- Sets up the display automatically
- Tries to rewrite the display `:0.0` as `griffin:0.0`.
- Automatically backgrounds the process
- Automatically closes all connections (**stdin** etc.)