UNIX AND LINIX IN INFOCOMMUNICATION
Week 6

O. Sadov

Networking history

UUCP

From the very beginning of the development of computer communication technologies, UNIX has been closely associated with them. Historically, the first worldwide network to operate over conventional dial-up telephone lines was created in the late 1970s at At&T and called `UUCP` — "UNIX to UNIX copy". And in 1979, two students at Duke University, Tom Truscott and Jim Ellis, originated the idea of using Bourne shell scripts to transfer news and mail messages on a serial line UUCP connection with nearby University of North Carolina at Chapel Hill. Following public release of the software in 1980, the mesh of UUCP hosts forwarding on the Usenet news rapidly expanded and named UUCPnet.

Technically, in the beginning, these could be dial-up modems, simply attached to the telephone tubes with suction cups which makes connects on hundreds of bits per second speed with very unstable connection. Even so, on this stage UNIX offered a fully functional network with the ability to remotely execute commands and transfer data over a complex mesh network topology.

UUCP provided just two main utilities:

- `uucp` — system-to-system copy
- `uux` — remote command execution

It was a very simple addressing scheme with no dynamic routing or anything similar, and the command to do something on a remote machine with files hosted on other machines might look like this:

```
uux 'diff sys1!~user1/file1 sys2!~user2/file2 >!file.diff'
```

Fetch the two named files from system `sys1` and system `sys2` and execute `diff` putting the result in file.diff in the current directory. It's funny, this addressing is still supported, for example, by the "`sendmail`" mail system, which adds some complexity to MTA.

TCP/IP

The development of modern networks began with research sponsored by the military Advanced Research Projects Agency (now DARPA) — ARPANET started in the late 1960s. The main goal of the development was systems that could withstand operation if partially destroyed, for example, as a result of a nuclear war. The main idea of the developed theoretical model was distributed adaptive switching of message blocks.

Research was carried out in some universities and companies, and as a result, the `TCP/IP` protocol was developed in the mid-70s. Its open source implementation on BSD UNIX in June 1989 spread it around the world and made it the backbone of the Internet. And we now have a mature 30-year-old worldwide network with dynamic routing, rich protocol stacks, and ready-to-use applications. The spread of this technology is now so great that the 32-bit address space of Internet Protocol version 4 (`IPv4`) is not enough now, and we are moving to 128-bit `IPv6`.

Traditional Network Utilities

In the world of `TCP/IP` Network, other programs have been developed that are still relevant in some cases, classical Internet programs:

- `telnet` — user interface to the TELNET protocol
- `ftp` — ARPANET file transfer program
- `mail` — send and receive mail

Again we have a tool for remote execution and a tool for data transfer.

Generally, telnet just gives us a connection to the TELNET protocol server:

```
man telnet
```

It's just a CLI for another host and this protocol still used for access to some hardware devices. Moreover, you can use it for debugging by connecting to other servers by choosing of TCP server's port. On modern systems, you can use the lighter '`nc`' utility, `netcat`. For example let's try to play with `HTTP` protocol:

```
$ telnet google.com 80
Trying 173.194.73.101...
Connected to google.com.
Escape character is '^]'.
GET /index.html HTTP/1.1
```

To switch to telnet command mode, press the "Ctrl-]" key. Here we can ask for help and exit, for example, if the program on the other side is frozen:

```
telnet> h
Commands may be abbreviated. Commands are:
close   close current connection
logout  forcibly logout remote user and close the connection
display display operating parameters
mode    try to enter line or character mode ('mode ?' for more)
open    connect to a site
quit    exit telnet
send    transmit special characters ('send ?' for more)
set     set operating parameters ('set ?' for more)
unset   unset operating parameters ('unset ?' for more)
status  print status information
toggle  toggle operating parameters ('toggle ?' for more)
slc     set treatment of special characters

z       suspend telnet
environ change environment variables ('environ ?' for more)
telnet> q
Connection closed.
```

FTP or File Transfer Protocol is another well-known part of the networked world of the Internet. It is still supported by some internet servers and is also built into some devices. We can access the FTP server through a regular web browser as well as through the `ftp` utility:

```
man ftp
```

In some cases, the latter variant is preferable, because, for example, we may want to restore a file or upload/download many files. First, we have to log into the FTP server. Let's try to do this as an anonymous user:

```
$ ftp ftp.funet.fi
Name (ftp.funet.fi:user): ftp
331 Any password will work
Password:
```

In this case any password will work, but often FTP-server wait email address as a password.

FTP has its own command line interface where we can ask for help:

```
ftp> ?
Commands may be abbreviated. Commands are:

!       dir mdelete qc site
$       disconnect mdir sendport size
account exit mget put status
append  form mkdir pwd struct
ascii get mls quit system
bell glob mode quote sunique
binary hash modtime recv tenex
bye help mput reget tick
case idle newer rstatus trace
cd image nmap rhelp type
cdup ipany nlist rename user
chmod ipv4 ntrans reset umask
close ipv6 open restart verbose
cr lcd prompt rmdir ?
delete ls passive runique
debug macdef proxy send
ftp>
```

We can first determine our current directory, and as we understand it, we have two current directories: remote and local. We can get the remote directory with the standard '`pwd`' command. To get the current local directory we can use the same command preceded by an exclamation mark. This means — call this command on the local computer. You may change directory remotely by '`cd`' and local directory by '`lcd`'.

We can get a list of remote directoriy using the well-known '`ls`' command.

And what about local '`ls`'? Yes — just preced it by an exclamation mark. This means — run the program locally. If you have sufficient permissions, you can download file by '`get`' command and upload by '`put`', but only a single file. If you want to work with multiple files, you will need to use the '`mget`'/'`mput`' commands.

In this case, it makes sense to disable the questions about confirming operations using the `prompt` command. Also switching to binary mode using the `bin` command can be important for the Microsoft client system. Otherwise, you may receive a corrupted file.

Finally, you can use the '`reget`' command to try to continue downloading the file after an interrupted transfer. And the '`hash`' command toggle the 'hash' printing for each transmitted data block, which can be informative if the connection to the server is poor.

Another useful scripting program is '`mail`', which is a simple command line client for sending email:

```
$ mail user@localhost
Subject: test
This is a test!
.
```

The mail message must end with one '`dot`' per line.

Internet Tools

OK. But what about modern Internet world?

The main problem with these classic '`telnet`' and '`ftp`' tools is insecurity: the user and password are transmitted over the network in plain text and can be hijacked by an evil hacker. Today they have practically been replaced by the Secure Shell utilities. Secure Shell provides secure, encrypted communication between untrusted hosts on an unsecured network. And again we have a remote Shell and transfer tool:

- `ssh` — SSH client (remote login program)
- `scp` — secure copy (remote file copy program)

```
man ssh
```

In ssh, you must specify the host and can set user and port. If you don't set a user, by default you will try to log in with the same name as the local user. Alternatively, you can add the command you want to run remotely directly to the command line, without that ssh will just start an interactive shell session on the remote host.

'scp' copies files between hosts on a network. It uses the same authentication and provides the same security as ssh, also data transfer encrypted by ssh. You also may choose a port, you can use compression while transferring.

Secure Shell utilities can be configured for passwordless authentication using certificates.

Internet Data-transfer Utilities

Finally, there are many tools that can be used to non-interactively access network resources in scripts.

The first is the 'lynx' text web browser. With the "-dump" parameter, it dumps the text formatted output of the URL of the WEB resource to standard output. This output can then be used when processing the web page in a script.

Also we have non-interactive network downloaders — 'wget' and 'curl'. These tools can be used to download and mirror online resources offline.

'lftp' — sophisticated file transfer program with different access methods — FTP, FTPS, HTTP, HTTPS, HFTP, FISH, SFTP and file.

And finally 'rsync' — remote (and local) file-copying tool which reduces the amount of data sent over the network by sending only the differences between the source files and the existing files in the destination. Rsync is widely used for backups and mirroring and as an improved copy command for everyday use. There are two different ways for rsync to contact a remote system: using a remote-shell program as the transport (such as ssh or rsh) or contacting an rsync daemon directly via TCP.

X-Window concepts

The history of the UNIX graphics system goes back to the 1984 MIT Athena education project. Athena was not a research project, and the development of new models of computing was not a primary objective of the project. Indeed, quite the opposite was true. MIT wanted a high-quality computing environment for education.

In collaboration with DEC and IBM, the project developed a platform-independent graphics system to link together different systems from multiple vendors through a protocol that can both run local and remote applications. This system was the basis of the X-Window System, which began its growth in 1985 and was ported to various UNIX and not just UNIX platforms.

We now have several successors to the classic X-Window system, the most famous being the Android or Wayland graphics system on Linux desktops, but X-Windows is still relevant today. And we will discuss some non-trivial concepts related to this. To see them just look into:

```
man X
```

First of all, the X-Window System is a network oriented client-server architecture. But the relationship between the client and the server doesn't seem very clear at first glance. Generally, the X server simply accepts requests from various client programs to display graphics (application windows) and sends back user input (from keyboard, mouse, or touchscreen). And the server will run on your tablet, and the apps that run on the supercomputer are just clients.

The main principle of X-Window: "*Provide mechanism rather than policy. In particular, place user interface policy in the clients' hands.*"

And the main locator for the X server is the DISPLAY environment variable:
`DISPLAY=hostname:displaynumber.screennumber`
Hostname is optional. Its absence means localhost. The display number is a unique identifier for this X server and should always appear in the display specification. And the screen number is an optional parameter for multi-screen X server configuration.

Let's try to play with this. First, we will switch to another virtual console, for example to second one by '`Ctrl+Alt+F2`'. Actually, in Linux we have

twelve virtual consoles in terms of the number of function keys on keyboard. And we can switch between them using '`Ctrl+Alt+function key`'. On some of them, we see a login prompt and can log in here. Now let's run X server:

```
$ X
(EE)
Fatal server error:
(EE) Server is already active for display 0
       If this server is no longer running, remove /tmp/.X0-lock
       and start again.
(EE)
(EE)
Please consult the The X.Org Foundation support
        at http://wiki.x.org
 for help.
(EE)
```

This is expected — we already have a running X-server in the system, which occupies a zero display. This is not a problem — let's try to run on the next display:

```
$ X :1
X.Org X Server 1.20.4
X Protocol Version 11, Revision 0
Build Operating System: 3.10.0-957.12.2.el7.x86_64
Current Operating System: Linux localhost 3.10.0-1127.18.2.el7.x86_64 #1 SMP Thu J
Kernel command line: initrd=initrd0.img root=live:CDLABEL=NauLinux_Qnet77-LiveCD r
rhgb rd
.luks=0 rd.md=0 rd.dm=0 BOOT_IMAGE=vmlinuz0
Build Date: 07 August 2019 08:52:04AM
Build ID: xorg-x11-server 1.20.4-7.el7
Current version of pixman: 0.34.0
       Before reporting problems, check http://wiki.x.org
       to make sure that you have the latest version.
Markers: (--) probed, (**) from config file, (==) default setting,
       (++) from command line, (!!) notice, (II) informational,
       (WW) warning, (EE) error, (NI) not implemented, (??) unknown.
(==) Log file: "/var/log/Xorg.1.log", Time: Wed Aug 26 18:45:45 2020
(==) Using system config directory "/usr/share/X11/xorg.conf.d"
```

```
(II) [KMS] Kernel modesetting enabled.
resizing primary to 1024x768
primary is 0x5645745b54b0
^C(II) Server terminated successfully (0). Closing log file.
```

It looks like a black screen without any graphic elements. Something is wrong? Oh no... On the fourth virtual screen, login again and set the DISPLAY variable:

```
$ export DISPLAY=:1
```

Now let's start the good old terminal interface — the 'xterm' application:

```
$ xterm
```

OK. Let's go to the third virtual console, where we left our X server. Great — we see the terminal window! But this is strange — we can only print something while staying in the terminal window, we cannot move or resize it, moreover, we do not have a button to destroy it!

It's just because we have a developed system based on the KISS paradigm — xterm simply emulates a terminal. If we want to move or resize windows (for example, we don't need this for an information kiosk), we need a special program for this — a window manager. Let's run it on 'xterm' by starting one of the graphical user environments — GNOME:

```
$ gnome-session &
```

OK. We now have a fully functional graphical user system in which we can work with graphical applications in the usual way.

GEOMETRY=WIDTHxHEIGHT+XOFF+YOFF
Now let's turn to another important point — geometry. With this parameter, we can set the position and size of the application window:

```
$ xterm -geometry 100x30+10+10
$ xterm -geometry 150x50+100+100
```

Finally, we can choose colors for applications that support such settings. X supports the use of abstract color names as described in the configuration file /usr/share/X11/rgb.txt. In this file, we can see the red, green and blue

values for the named color definitions.
COLOR=<color_space_name>:<value>/.../<value>
And we can use color names like this for example as application parameters:

```
xterm -bg blue -fg red
```

X-Window fonts

Another non-trivial point is fonts. XWindow supports both bitmaps and scalable fonts. In the latter case, it's possible to use so-called font servers to remotely render scalable fonts to bitmaps, which was useful for low-level X terminals.

Classic XWindow fonts are handled by utilities: '`xfontsel`', '`xfd`' and '`xlsfonts`'.

```
xfontsel
```

In the font specification, we see the manufacturer's name, type, variety, size, resolution, encoding, and so on:
`-adobe-courier-medium-?-normal-10-100-75-75-m-60-iso8859-*`

Font names tend to be fairly long as they contain all of the information needed to uniquely identify individual fonts. However, the X server supports wildcarding of font names, so the full specification.

This is good, but not good enough for the modern WYSIWYG world. The standard XWindow paradigm knows nothing about presentation on paper, only on screen. All documents are executed by applications creating PostScript language output for high quality printing.

And with the widespread distribution of office suites, this paradigm turns out to be insufficient. For the modern WYSIWYG graphical interfaces, a new font engine has been developed — FontConfig, which works with PostScript and TrueType fonts and is processed by utilities: `fc-cache`, `fc-list`, `fc-cat` and `fc-match`.

*

X-Window options

Classic XWindow applications are built using the XToolkit library and generally support a standard set of options:

- `-display` — name of the X server to use
- `-geometry` — initial size and location of the window
- `-title` — window title
- `-bg`, `-background`, `-fg color`, `-foreground` — window background/-foreground color
- `-fn`, `-font` — font to use for displaying text
- `-name` — name under which resources for the application should be found
- `-xrm` — resource name and value to override any defaults

X-Window resources

Finally, the so-called resources described in the manual pages for such applications can be used to customize the default settings for XWindow applications. Resources must be located in the '`.Xdefaults`' or '`.Xresources`' file in the $HOME directory and can be processed by the '`xrdb`' utility on the fly.

The description looks like this:
`obj.subobj[.subobj].attr:  value`
And in XWindow, the object-oriented paradigm was implemented even before it was implemented in well-known programming languages. Program components are named in a hierarchical fashion, with each node in the hierarchy identified by a class and an instance name. At the top level is the class and instance name of the application itself. By convention, the class name of the application is the same as the program name, but with the first letter capitalized:

- `Obj` — class name
- `obj` — instance name

Some examples:

```
XTerm*Font: 6x10
emacs*Background: rgb:5b/76/86
```

GNOME user interface uses its own resource management — Gconf (`gconf-editor`, `gconftool-2`).

Xserver

OK. Let's look to some classical XWindow applications.

The first one as we know is a X server:

```
man X
```

Most important options:

- `:displaynumber` — default is 0
- `-fp fontPath` — search path for fonts
- `-s minutes` — screen-saver timeout time in minutes

And some options that can help organize a local XWindow network with low-cost X terminals and application servers:

- `-query hostname` — enables XDMCP and sends Query packets to the specified hostname on which this or that display manager is running;
- `-broadcast` — enables XDMCP and broadcasts BroadcastQuery packets to the network. In this way, simple load balancing between application servers can be organized.
- `-indirect hostname` — enables XDMCP and send IndirectQuery packets to the specified hostname. In this case, you will see a list of available application servers that you can select.

Xserver settings

After starting the X-server, it is possible to change some parameters on the fly by '`xset`' command:

```
man xset - user preference utility for X
```

Options:
`-display display` — set display
`q` — current settings
`[+|-]fp[+|-|=] path,...` — set the font path for X-server, including font-server
`fp default` — font path to be reset to the server's default.
`fp rehash` — reset the font path to its current value (server reread the font databases in the current font path)
`p` — pixel color values
`s` — screen saver parameters

X-Window utiltities

As we discussed earlier, the main principle of the X Window System is "Provide a mechanism, not a policy." And the look and feel in X Window can be anything — it is simply determined by the set of widgets on which a particular application is built. It is not a paradox, but the appearance of the original XWindow applications may seem a little odd to modern users, as they are based on an ancient set of widgets from the Athena project. It looks "ugly" at now days, but they were often used in the period of X history that he describes as the "GUI wars", as a safe alternative to the competing Motif and Open Look toolkits.

Let's look at the well-known for us '`xterm`' application:

```
xterm
```

mverb

As we can see, these are very simple 2D graphics with very unusual scrollbar behavior, which often discourages new users. The general abstraction of a mouse pointer in an XWindow is a three-button device. If you have a mouse with fewer buttons, the middle button is emulated, for example, by

simultaneously pressing the left and right buttons. So here: pressing the left button on the scroll bar scrolls forward, the right button backward, and the middle button scrolls to the selected position.

Yet another classic XWindow utilities:

`xkill` — kill a client by its X resource

`xdpyinfo` — display information utility for X

`xwininfo` — window information utility for X

`xlsclients` — list client applications running on a display

`showrgb` — display an rgb color-name database

`appres` — list X application resource database

`xrdb` — X server resource database utility

`editres` — a dynamic resource editor for X Toolkit applications

`xsetroot` — root window parameter setting utility for X

`xev` — print contents of X events

`xmodmap` — utility for modifying keymaps and pointer button mappings in X

`setxkbmap` — set the keyboard using the X Keyboard Extension

`xrefresh` — refresh all or part of an X screen

and others.