UNIX AND LINIX IN INFOCOMMUNICATION Week 4

O. Sadov

Aug 23, 2020 10:33 PM

Files

Finally, let's discuss the third pillar that holds the whole UNIX world — files.

To facilitate the work of users who are not familiar to working with the command line, there are a number of free file management interfaces, for example, Midnight Commander (mc), reminiscent of Norton Commander, or graphical file managers, reminiscent of MS Windows Explorer. But we'll see how we can work with files and directories from the CLI or scripts.

First, let's take a look at some of the symbols that have special meaning in the file path:

```
/ - root directory and directory separator
. - current directory
.. - parent directory
~/ - home directory
```

As we can see, UNIX uses a slash as the directory separator, and Windows uses a backslash. This is interesting because early versions of Microsoft's MSDOS operating systems did not support subdirectories just because it was just a clone of CP/M OS from Digital Research. It was a small OS for 8-bit microcomputers. It was a small OS for 8-bit microcomputers without disk storage or with a small floppy disk. Usually there were only a few dozen files on a floppy disk, and only a flat file system with one directory per file system was supported.

And at first, Microsoft MSDOS operating systems didn't support subdirectories. Only when developing its own "multiuser" OS — OS Xenix based on UNIX, Microsoft implemented a hierarchical file system and ported it to the "single user" MSDOS. But at that point the forward slash was already taken — it was used as a standard CP/M command option marker, like a 'dash' in UNIX commands. And Microsoft choose a 'backslash' as a directory marker.

OK. As we remember, we have a hierarchical file system with a single root directory and for newbies, this file system hierarchy can seem too complex. They say, "When we install some software on Windows, we have separate directories for each product, and it's too easy for us to find something, but on your system we don't know where we can find something".

But in fact, in UNIX-like systems, we have a very clear and stable standard for file system hierarchy, which is reflected, for example, in the corresponding Linux specification: https://refspecs.linuxfoundation.org/fhs.shtml

In fact, we have three main levels with a repeating directory structure. At the first level, we have directories like this: (http://sdn.ifmo.ru/education/courses/free-libre-and-open-source-software/lectures/lecture-5/). In the '/usr' and '/local' directories we see again: (http://sdn.ifmo.ru/education/courses/free-libre-and-open-source-software/lectures/lecture-5/).

And, as I said, devices in UNIX-like systems look like files, but as special files placed in a special directory '/dev': (http://sdn.ifmo.ru/education/courses/free-libre-and-open-source-software/lectures/lecture-5/). Usually each such file is just a rabbit hole in the OS kernel. When working with a pseudo-file in this directory, we see this device as a stream of bytes and work with it as with a regular file.

We may also have many other secret paths to the kernel, such as `/proc` and `/sys`. For example, we can see:

- `/proc/cpuinfo`
- `/proc/meminfo`
- `/proc/interrupts`

File Commands

UNIX tools support a standard set of commands for working with files and directories:

- `ls` — list directory contents. Let's look in '`man ls`'. We can simply specify files and directories as arguments and view the listing in different ways according to the options.

  Ok, let's take a look at our current directory — it's just '`ls`' without arguments. As we remember, after logging in, this is the home directory.

```
ls
```

We see some directories, but we don't see, for example, shell startup files. No problem, let's run:

```
ls -a
```

We can see the shell startup files and more — the directories "point" (current) and "double point" (top level) are also visible. Because that means "all" files and directories, including hidden ones. Hidden files in UNIX are just a naming convention— names must begin with a period. It is not an attribute as it is on Microsoft systems. Initially it was just a trick in the '`ls`' utility to hide the current and top directories, and then it came to be used as a naming convention to hide any file or directory.

Also we can see directory listing recursively:

```
ls -R
```

Another very important option is the "long list":

```
ls -l
```

We see a table with information about the file/directory in the corresponding lines.

- The first column is the file attribute. The first letter is the file type: "dash" is a regular file, "d" is a directory, and so on. Then we can see read, write, and execute permissions for three user groups: owner, owner group, and everyone else. Once again, we see the difference between UNIX and Microsoft. In the first case it is an attribute, in the second case executability is just a naming convention: '.com', '.exe', '.bat'.
- Some mystery column that we will discuss later.
- Then we can see owner and owner group, size of file, time of modification and the name of file.

- `pwd` — print name of current/working directory

- cd — change directory. Without arguments – to home firectory.

- cp — copy files and directories. Most interesting option is '-a|--archive' with create recursive archive copy with preserving of permissions, timestamps, etc. . .

- mv — move (rename) files and directories.

- rm — remove files or directories.

  ```
  rm -rf ...
  ```

  means recursive delete without asking for confirmation.

- mkdir — make directories. If any parent directory does not exist, you will receive an error message:

  ```
  mkdir a/b/c
  mkdir: cannot create directory 'a/b/c': No such file or directory
  ```

  To avoid this, use the -p option:

  ```
  mkdir -p a/b/c
  ```

- rmdir — remove empty directories. If directory is not empty, you will receive an error message. Nowadays, running 'rm -rf something...' is sufficient in this case. But in the old days, when 'rm' did not have a recursive option, to clean up non-empty directories, you had to create a shell script with 'rm's in each subdirectory and the corresponding 'rmdir's.

- ln — make links between files. Links are a very specific file type in UNIX and we will discuss them in more detail. If we look at the man page for the 'ln' command, we see a command very similar to 'cp'. But let's take a closer look:

  ```
  cat > a
  ln -s a b
  ln a c
  cat b; cat c
  ```

At the moment everything looks like a regular copy of the file, but let's try to change something in the one of them:

```
cat >> c
cat a; cat b
```

Wow, all the other linked files have changed too! We are just looking at the same file from different points, and changing one of them will change all the others. And in this they all seem to be alike. But let's try to delete the original file:

```
rm a
cat b; cat c
```

In the first case, we can still see the contents of the original file, but in the second case, we see an error message. Simply because the first is a so-called hard link and the second is symbolic. We can see the difference between the two in the long ls list:

```
ls ?
```

And we can restore access to the content for the symbolic link by simply recreating the original file:

```
ln b a
cat c
```

Another difference between them is the impossibility of creating a hard link between different file systems and the possibility of such a linking for soft links. For more details on internal linking details, see the corresponding lecture.

Permissions

And finally, let's discuss file permissions. As we remember, we have the owner user, the owner group and all the others, as well as read, write and execute permissions for such user classes. And we have the appropriate command to change these permissions:

```
chmod [-R] [ugoa][-+=](rwx)
```

And as we understand it, permissions are just a bit field. As far as we understand, permissions are just a bitfield and in some cases it might be more useful to set them in octal mode — see for information on this.

You can also change the owner and group for a file or directory by command 'chown'.

```
man chown - change file owner and group
```

But keep in mind — for security reasons, only a privileged user (superuser root) can change the owner of a file. The common owner of a file can change the group of a file to any group that owner is a member of:

```
chown :group file...
chgrp group file...
```

Text Viewers

As we remember, UNIX was originally created to automate the work of the patent office, has a rich set of tools for working with text data. But what is text? Generally it is just a collection of bytes encoded according to some encoding table, originally ASCII. In a text file, you will not see any special formatting like bold text, italics, images, etc. — just text data. And this is the main communication format for UNIX utilities since the 1970s.

As you know, Microsoft operating systems have different modes of working with files — text and binary. In UNIX, all files are the same, and we have no difference between text and binary data. See details in (`under_the_hood/pipelining_Windows.txt`).

Concatenating and spliting

The first creature that helps us work with text files is the "`cat`". Not a real "cat", but an abbreviation for concatenation. With no arguments, cat simply copies standard input to standard output. And as we understand it, we can just redirect the output to a file, and this will be the easiest way to create a new file:

```
cat > file
```

When we add filenames as arguments to our command, this will be a real concatenation — they will all be sent to the output. And if we redirect them to a file, we get all these files concatenated into an output file.

```
cat f1 f2... > all
```

If we can combine something, we must be able to split it. And we have two utilities for different types of breakdowns:

- `tee` — read from standard input and write to standard output and files
- `split` — split a file into fixed-size pieces

Text viewers and editors

What is it viewer? In the TTY interface, the man command seems like a good one — when you run it, you get paper manuals that you can combine into a book, put on a shelf, and reread as needed. On a full-screen terminal — before, Ctrl-S (stop)/Ctrl-Q(repeat) was enough for viewing, because at first the terminals were connected at low speed (9600 bits per second for ex.), and now special programs were used — viewers. Unlike text editors, viewers does not need to read the entire file before starting, resulting in faster load times with large files.

Historically, the first viewer was the “`more`” pager developed for the BSD project in 1978 by Daniel Halbert, a graduate student at the University of California, Berkeley. The command-syntax is:

```
more [options] [file_name]
```

If no file name is provided, ‘more’ looks for input from standard input.

Once ‘more’ has obtained input (file or stdin), it displays as much as can fit on the current screen and waits for user input. The most common methods of navigating through a file are Enter, which advances the output by one line, and Space, which advances the output by one screen. When ‘more’ reaches the end of a file (100%) it exits. You can exit from “more” by pressing the “q” key and the “h” key will display help. In the ‘more’ utility you can search with regular expressions using the ‘slash’ or the ‘+/’ option. And you can search again by typing just a slash without regexp. Regexp is a very important part of UNIX culture and is used in many other programs and programming environments: (`http://sdn.ifmo.ru/education/courses/free-libre-and-open-source-software/lectures/lecture-6`)

The ‘main’ limitation of the more utility is only forward movement in the text. To solve this problem, an improved ‘more’ called ‘`less`’ was developed. The “less” utility buffers standard input, and we can navigate forward and backward through the buffer, for example. using the cursor keys or the PgUp/PgDown keys. A reverse search with a question mark is possible.

Text Editors

OK. We can create a file using the ‘`cat`’ utility and view the file using a viewer. But what if we need to change something, especially if we only have a TTY interface? And it is possible — we have a so-called line editor named ‘`ed`’. The only interface for such an editor is the command line: (`http://sdn.ifmo.ru/education/courses/free-libre-and-open-source-software/lectures/lecture-6`).

So let’s try to edit new file.

```
$ ed tst
tst: No such file or directory
```

At first — we will add some lines:

```
i
1 2
3 4
```

and we must end our input with one ‘dot’ per line.

```
.
```

Let’s take a look at our file, moving to the first line:

```
1
```

```
1 2

3 4

?
```

Seems good. Now we can add something to the end:

```
a
5 6
.
1
1 2

3 4

5 6

?
```

OK — we have 3 lines in the file now. And finally — let's try to make a magic pass:

```
1,$s/\(.\) \(.\)/\2 \1/
1
2 1

4 3

6 5

?
```

This means: from the first to the last line, replace the lines where we have two separate letters separated by a space, exchanging those letters with places. And now 'write' and 'quit':

```
w
12
q
```

Let’s check the result:

```
$ cat tst
2 1
4 3
6 5
```

But for what purposes can you use a line editor now that we have full-screen editors with a user-friendly interface? Of course, you can imagine a situation where your full-screen environment is broken and only the line editor will be the salvation. And in general I had such situations. But the main use case for `ed` is for automatic editing in scripts. Anything you need to change in the text data can be done with this editor, including sophisticated regex search and replace.

Moreover, we have a ‘`sed`’ — stream editor, for editing text data in pipelines:

```
$ sed ’s/\(.\) \(.\)/\2 \1/’ < tst
1 2
3 4
5 6
```

As you can see, the original file does not change, all changes are simply sent to standard output:

```
$ cat tst
2 1
4 3
6 5
```

But UNIX-like systems also have full-screen editors, which can also be confusing for beginners. It was developed by Berkeley student Bill Joy for BSD initially as a visual mode for a line editor. It is a very fast and lightweight editor that is part of the Single Unix Specification and the POSIX, which found on every UNIX-like system. The `VI` editor works on all types of terminals and generally requires only a conventional letter keyboard. You can work with it without the arrow keys, PgUp/Down or anything similar. There are actually very small keyboards out there that are optimized for ‘vi’.

But to work on it, you need to understand the basic concept of this editor: it can be in three states (`http://sdn.ifmo.ru/education/courses/`

`free-libre-and-open-source-software/lectures/lecture-6`).

Immediately after launch, we find ourselves in the usual command mode and can switch to editing mode, for example, by pressing the "[Insert]" key. As we can see, the mode status in the lower left corner has changed to '-- INSERT --', and now we can edit our file. Pressing Insert again will change the state mode to '-- REPLACE --' and vice versa. Exit the editing mode by pressing ESC. The third mode can be accessed by pressing the colon key in command mode. This is 'ed' mode. In this mode, we can use the normal 'ed' line editor commands and finish them with ENTER.

In command mode, you can find something with regex by slash and question marks, as in the 'less' viewer. In improved VI (`vim`), you can use very useful visual mode by pressing V. After that you can delete the selection with 'd' or just copy it with 'y' (yank). Then you can paste it anywhere with 'p' (paste). Moreover, you can use [Ctrl-V] to select a vertical box. To exit visual mode, simply press ESC.

Also you may look to `vimtutor` — a guide to Vim can be useful for beginners.

And the second most common editor is `Emacs`. This Richard Stallman's editor was the starting point for the GNU Project, along with GCC and UNIX utilities. EMACS means, for example, "Editing MACroS". An apocryphal hacker koan alleges that the program was named after Emack&Bolio's, a popular Cambridge ice cream store. But overall it is a really very powerful editor with macro extensions, allowing the user to override any keystrokes to launch the editor program. But unlike other editors that support macro-extensions, in Emacs they are implemented using the LISP programming language embedded on editor. At the time, LISP was very popular in artificial intelligence in the United States, and Stallman, who worked at the MIT Artificial Intelligence Lab, chose it as the editor extension language.

This implementation allows many LISP-based applications to be developed, including a user-friendly interface for developers in various programming languages. Usually Emacs is a text editor with a simple graphical interface. But it can only be run in a text environment. The most commonly used keystrokes are:
`C-c C-x` — exit
`C-h t` — tutorial
`C-h i` — info

If you feel overwhelmed by the difficulty of Emaccs, you can see a personal psychoanalyst: `M-x doctor`. It would spoil the fun and hurt your recovery to say too much here about how the doctor works. But when you're ready, you may try to find the well-known Turing-test related AI program ELIZA on WikiPedia.

Also in the UNIX/Linux world, there are many other editors that may be more convenient for you, such as:

- `joe`, `nano` — simple text editors or
- `gedit`, `kate` – text editors from Gnome and KDE projects

Advanced Text Utilities

Searching

If we are talking about text data, finding some text is a common task. And in fact, these are two separate tasks — to find some text inside a file or text stream and to find a file, for example, by name in some directories.

For the first task, we have the '`grep`' utility which print lines matching a pattern.

```
man grep
```

Both fixed strings and regular expressions can be used as a pattern. Also you can do recursive search.

Another commonly used command is '`find`' — search for files in a directory hierarchy.

```
man find
```

You must set the starting point — the directory to start the search or starting points if you are interested in several directories and expressions with search criteria and actions. You may search by name with using of standard shell matching patterns, by time of modification or access, by size, by user and group, by permissins, file type, etc. You can use logical operators such as "and", "or" and "not" in your expressions.

Also you can do some actions when you find something that matches the criteria. The default action is ‘print’. You can also use formatted printing, list of found files, delete them, and execute commands with them. In ‘exec’ actions, you can use curly braces to insert the name of the found file. But keep in mind — you must end your command with a semicolon, and to avoid interpreting this Shell character, you must escape it with a ‘slash’.

But the main drawback of ‘find’ is the long execution time if you are looking in large deep directories. And to speed up this process, you can use the ‘locate’ utility. It finds files by name from databases prepared by ‘updatedb’ and does it incredibly fast. But you have to understand — ‘updatedb’ is started automatically by the cron service at night. And if you only install the ‘lookup’ toolkit or want to find something in the changed filesystem at this time — you have to update this database manually by running ‘updatedb’.

Utilities for Manipulation with a Text Data

Another operation that we often need is comparing files or directories. And we have some tools for this.

```
man cmp
```

The ‘cmp’ utility compares the two files byte-by-byte and reports the position from which we have a difference. By this way we can compare binaries.

To compare text files ‘diff’ utility can be used:

```
man diff
```

We can compare files, directories with the ‘--recursive’ option. We can get the output as a set of commands for the ‘ed’ editor or the ‘patch’ utility. This method of propagating changes was the first in the development of projects in the UNIX ecosystem and is still useful today.

Another important action with text data is sorting, and we have the ‘sort’ utility which sort lines of text files:

```
man sort
```

To eliminate duplicate lines, we have the uniq utility, but first we have to sort our text stream:

```
sort file | uniq
```

We may output the first/last part of files by ‘head’ and ‘tail’ utilities. By default is the first or last 10 lines of standard input, or each FILE from arguments to standard output. You can set another number of lines as an option:

```
tail -15
```

Also in ‘tail’ you can use the ‘--follow’ option to display the appended data as the file grows.

More that, from text lines you can cut some fields, separated by some kind of separators by ‘cut’ utility.

Also you can join lines of two files on a common field by ‘join’ utility and merge lines of files by ‘paste’.

And finally, we have ‘awk’, a scanning and templating language that can do this and other complex work on text files or streams.

Basic Network Utilities

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.

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. For example HTTP:

```
$ 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. 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.