Learning the UNIX/UNIX-Like Command-Line

A Guide for Beginners

Stacey Pellegrino, M.Sc.

Table of Contents

Introduction

Historically, the first version of UNIX was developed by Ken Thompson and Dennis Ritchie at AT&T Bell Labs in 1971. This initial version, written in assembly language (the lowest-level hardware architecture-specific programming language), was limited to a small set of commands. In 1973, the UNIX Operating System (OS) was rewritten in the C programming language, which allowed for source code portability and made it easier for other system software developers to contribute to the project.

UNIX was made available to universities for free in 1974, leading to a period of rapid innovation and development. During the 1980s, UNIX was adopted by many commercial hardware vendors, which further accelerated the development of UNIX. Additionally, the development of the X Window System (X11) helped in making it easier to create Graphical User Interface (GUI) applications.

In the 1990s, the open-source movement gained momentum, leading to the release of UNIX-like systems such as Linux and BSD (Berkeley Software Distribution) variants. This sparked another wave of rapid innovation and development.

UNIX and UNIX-like systems include:

- Microsoft WSL (Windows Subsystem for Linux)
- macOS
- Linux
- FreeBSD
- OpenBSD
- Illumos (namely OpenSolaris and OmniOS)
- Solaris
- AIX
- HP-UX

These systems are widely used in various applications and are considered one of the most important Operating System standards available.

The UNIX/UNIX-like command-line is an extremely powerful tool that can be used to quickly accomplish a wide range of tasks. This introduction to the command-line explains the basic syntax and usage of common commands for managing/manipulating files and programs on UNIX/ UNIX-like systems. It allows users to perform various tasks, such as navigating the filesystem, creating/deleting files and directories, modifying permissions, running programs, and so much more. It is often used by Systems Administrators and Software Developers/Engineers to perform complex tasks efficiently.

So why use the UNIX/UNIX-like command-line? It is an efficient way to interact with a computer, as it allows for quick and direct control of the system. It is also a great way to automate tasks and to access powerful tools and utilities that are not available through a GUI. It is also easier to reproduce sequences of tasks than follow GUI screenshots. In fact, a trace and record of command-line actions can be accessible from the history of commands. Also worth noting is that when there is no GUI on the target system, server access can be done with a remote shell, ideally with SSH (Secure Shell), which has encrypted transport over the network (even the internet). Furthermore, advanced use of the command-line requires a higher degree of knowledge and understanding of the underlying system.

To begin, use a terminal application in order to access a Command-Line Interface (CLI). However, there are many interchangeable terms, such as a Text User-Interface (TUI) in contrast to a Graphical User-Interface (GUI). The following details the alternative terms for a command-line terminal:

- Command prompt
  Console (although this can be both physical via serial RS232 and virtual with direct system access)
- Shell (a command-line interpreter):
  - Korn shell (ksh)
  - C shell (csh)
  - Z shell (zsh)
  - Bourne shell (sh)
  - Bash shell (bash) also known as Bourne Again Shell (most common shell used)

Common Keyboard Shortcuts

Here are some useful CTRL sequence shortcuts for a standard UNIX terminal (performed by holding the CTRL key followed by a character):

- `CTRL–C` - send an interrupt requests to a process with the signal `SIGINT`
- `CTRL–Z` - suspend a process with the signal `SIGTSTP`
- `CTRL–D` - logout (also achieved with the commands `logout` and `exit`)
- `CTRL–L` - clear screen (also achieved with the command `clear`)
- `CTRL–U` - clear line (very useful for correcting typos in non-echoed password prompts)

Shell Basics

Most of the underlying fundamentals of a terminal application featuring a shell (command-line interpreter) have now been introduced. However, further knowledge of UNIX commands is required in order to interface with the OS via a shell in a terminal.

Upon launching the terminal application a prompt is presented. This prompt will typically display the name of the current user, the hostname logged into and the directory they are in. For example, the prompt might look like this:

```
user@hostname:~ $
```

This prompt indicates that the current user is `user`, the hostname is `hostname` and they are in their home directory denoted by the ~ symbol. A typical standard for the end symbol of a prompt is `$` for a regular user and `#` for the superuser. The primary prompt of the shell is denoted by an environment variable `PS1`. This variable can be printed by echoing this variable with the following command:

```
$ echo $PS1
```

The first command to understand and use is `pwd` (print working directory). Suppose the username is john then upon login to a UNIX/UNIX-like system issuing the following command will output as follows:

```
$ pwd
/home/john
```

User Environment

To display all the shell environment variables execute the command `env`. To print the individual variables to the output of the terminal:

```
$ echo $HOME
/home/john

$ echo $PS1
\n\u: \w\n\D{%F %T}\n\n$
```

This custom `PS1` prompt allows for better auditing capability as it displays the username, current working directory, the date and time followed by a new line with the default user prompt character `$`, and displays as follows:

```
spellegrino: /etc
2023-01-19 17:12:01

$
```

Within a shell environment there will typically be an environment variable called `PATH` that allows you to execute a command without giving an absolute or relative path name to the binary command to execute. This can be displayed by issuing the following command:

```
$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
```

To update the default `PATH` environment variable whilst preserving the original variable, then either prepend or append the additional `PATH` (separated by colons as the delimiter) with:

```
$ export PATH=/opt/bin:$PATH
```

In doing so the `PATH` variable is changed for the existing environment session and `echo $PATH` now outputs the following:

```
/opt/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/
snap/bin
```

Bash shell configuration files (which are hidden due to the preceding `.` to the filename) reside in the home directory and can be reread after editing with the source command:

```
$ source .bashrc
$ source .bash_profile
```

The command `source` is interchangeable with the character `.` (period), therefore when executing the command `. ~/.bash_profile` it will also reread the Bash profile into the current shell environment.

The following is a Bash shell option (denoted by the `-o` flag) to enable `vi` (visual editor) inline editing on the command-line:

```
$ set -o vi
```

To make this persistent on subsequent logins thereafter then append this command to `~/.bash_profile` file (which is read to configure the Bash environment upon every login).

To review the history of commands that are written to the `~/.bash_history` file just execute the following:

```
$ history
    1  ifconfig
    2  ip addr show
    3  sudo apt install net-tools
    4  ls
    5  ifconfig
    6  ping 192.168.0.19
    7  ifconfig -a
    8  tree
    9  sudo apt install tree
    10 tree
    11 tree-d-L2/ 12 set-ovi
    13 history
```

The `which` command is used to locate a command. It searches for the given command name in all directories specified in the `PATH` environment variable and displays the absolute path of the executable if found. It is useful for finding out if a command is available in the system and what its exact location is. An example is:

```
$ which ls
/usr/bin/ls
```

The `whereis` command is used to locate source, binary and man (manual) page files for a given command. It searches for the given command name in the standard UNIX directories and displays the absolute path of the executable, source and man page files if found. An example is:

```
$ whereis ls
ls: /usr/bin/ls /usr/share/man/man1/ls.1.gz
```

To run a program not located in the `PATH` environment variable, for example, an executable file called `program` in the current directory, execute the following command (notice the `./` prefix to the program name):

```
$ ./program
```

Run the following command to check the exit status of the previous executed command (whereby a `0` indicates that the program executed successfully, whilst a non-zero value refers to a returned error):

```
$ echo $?
```

Running programs from the command line is straightforward, and checking the exit status helps verify the success or failure of commands.

Help and Support

Some of the best resources are https://superuser.com and https://askubuntu.com. However, UNIX systems typically include a standard documentation system known as man pages, which are in a consistent format. The `man` command is used to view the man page of a particular command in a UNIX system. The man page consists of information about the command such as its syntax, options and usage. To use the `man` command, you can type man <*command*> into a terminal. For example, viewing the man page for the command `ls` would be the command `man ls` accordingly. Also worth exploring is the man page for the `man` command itself, that simply being the command `man man`.

When viewing a manl page of a command it will render in the terminal with the user configured pager (either `more` or `less`). To navigate each man page use the ↑ (up) and ↓ (down) arrows of the keyboard, although this only shifts the output by one line at a time, therefore press the `spacebar` to page forward (within the screen size of the terminal) or to page backward press the `b` key. Within the man page it is possible to search the text accordingly. This is done by either pressing the key `/` for a forward search or pressing the key `?` for a reverse search, whereby the string appended to the keys of `/` or `?` will be used as the search input, for example `/EXAM` will match any uppercase strings, such as EXAMPLE or another example is `/-k` to match a flag of `-k` in the man page being searched. If there are multiple matches of the search string, these can all be referenced by cycling the search in the direction of the search (either forward `/` or reverse `?`) with the lowercase `n` key or the opposite direction with the uppercase `N` key. Furthermore, the use of regular expressions (regex) can also be used for the search string too. Finally, to quit viewing the man page just simply press the `q` key accordingly.

The man -k <*keyword*> command is used to search for man pages related to a particular keyword or phrase. This command is often referred to as `apropos` too. Suppose a man page reference to the keyword `time` is required, then `man -k time` would output as follows:

```
adjtime_config (5)   - information about hardware clock setting and drift factor
bootparam (7)        - introduction to boot time parameters of the Linux kernel
chrt (1)             - manipulate the real-time attributes of a process
date (1)             - print or set the system date and time
dmmp_context_timeout_get (3) - Get IPC timeout.
dmmp_context_timeout_set (3) - Set IPC timeout.
finalrd (1)          - final runtime directory generator for shutdown
hwclock (8)          - time clocks utility
kbdrate (8)          - reset the keyboard repeat rate and delay time
keep-one-running (1) - run just one instance at a time of some command and unique set of argument...
ldconfig (8)         - configure dynamic linker run-time bindings
localtime (5)        - Local timezone configuration file
modules (5)          - kernel modules to load at boot time
openssl-s_time (1ssl) - SSL/TLS performance timing program
openssl-ts (1ssl)    - Time Stamping Authority tool (client/server)
openssl-tsget (1ssl) - Time Stamping HTTP/HTTPS client
pam_time (8)         - PAM module for time control access
pam_timestamp (8)    - Authenticate using cached successful authentication attempts
pam_timestamp_check (8) - Check to see if the default timestamp is valid
rtc (4)              - real-time clock
rtcwake (8)          - enter a system sleep state until specified wakeup time
run-one (1)          - run just one instance at a time of some command and unique set of argument...
run-one-constantly (1) - run just one instance at a time of some command and unique set of argume...
run-one-until-failure (1) - run just one instance at a time of some command and unique set of arg...
run-one-until-success (1) - run just one instance at a time of some command and unique set of arg...
run-this-one (1)     - run just one instance at a time of some command and unique set of argument...
s_time (1ssl)        - SSL/TLS performance timing program
sg_timestamp (8)     - report or set timestamp on SCSI device
slabtop (1)          - display kernel slab cache information in real time
sleep (1)            - delay for a specified amount of time
sudoers_timestamp (5) - Sudoers Time Stamp Format
sysctl (8)           - configure kernel parameters at runtime
systemd-debug-generator (8) - Generator for enabling a runtime debug shell and masking specific u...
systemd-run (1)      - Run programs in transient scope units, service units, or path-, socket-, o...
systemd-time-wait-sync (8) - Wait Until Kernel Time Synchronized
systemd-time-wait-sync.service (8) - Wait Until Kernel Time Synchronized
systemd-timedated (8) - Time and date bus mechanism
systemd-timedated.service (8) - Time and date bus mechanism
systemd-timesyncd (8) - Network Time Synchronization
systemd-timesyncd.service (8) - Network Time Synchronization
systemd-tmpfiles-clean.timer (8) - Creates, deletes and cleans up volatile and temporary files an...
systemd.time (7)     - Time and date specifications
systemd.timer (5)    - Timer unit configuration
tc-etf (8)           - Earliest TxTime First (ETF) Qdisc
tc-taprio (8)        - Time Aware Priority Shaper
time (1)             - run programs and summarize system resource usage
time (3am)           - time functions for gawk
```

```
time (7)             - overview of time and timers
time.conf (5)        - configuration file for the pam_time module
timedatectl (1)      - Control the system time and date
timeout (1)          - run a command with a time limit
timesyncd.conf (5)   - Network Time Synchronization configuration files
timesyncd.conf.d (5) - Network Time Synchronization configuration files
touch (1)            - change file timestamps
ts (1ssl)            - Time Stamping Authority tool (client/server)
tsget (1ssl)         - Time Stamping HTTP/HTTPS client
tzfile (5)           - timezone information
tzselect (1)         - view timezones
tzselect (8)         - select a timezone
uptime (1)           - Tell how long the system has been running.
user-runtime-dir@.service (5) - System units to manage user processes
vcstime (8)          - Show time in upper right hand corner of the console screen
xfs_rtcp (8)         - XFS realtime copy command
zdump (8)            - timezone dumper
zic (8)              - timezone compiler
```

The `apropos` command is an alias for the `man -k` command and is used to search for man pages related to a particular keyword or phrase. Man pages provide detailed information about commands, their options and usage examples, serving as a valuable reference.

UNIX Filesystem Hierarchy

The UNIX filesystem is hierarchical, meaning directories contain files and other directories. The composition of a generic UNIX filesystem hierarchy standard (in alphabetical order) is as follows:

`/`

The root directory is the topmost directory in the UNIX filesystem hierarchy. It is the parent of all other directories and contains the necessary files and directories to boot the system. This directory is often referred to as the *slash* directory and its equivalent in MS Windows is `C:\`.

`/bin`

The `bin` directory (which is shorthand for binary) contains executable files that are used for system administration and maintenance. Command-line utilities such as `ls`, `cp`, `mv`, and `rm` are typically found in this directory or in `/usr/bin` (which it points to as a symbolic link).

`/boot`

This directory contains the files used to boot the system. This includes the initial RAM disk `initrd` that contains the UNIX/UNIX-like kernel and necessary drivers, as well as the boot loader.

`/dev`

The `dev` directory contains device files that represent devices on the system. These files are used by the kernel to access devices and manipulate their drivers.

`/etc`

The `etc` directory contains system configuration files. These files are used to configure the system and can be edited by a privileged user.

`/home`

The `home` directory is used to store user-specific files, such as documents, images, music and videos. Each user on the system typically has a `home` directory, for example, a username of `john` will have their home directory as `/home/john`.

`/lib`

The `lib` directory contains libraries needed by programs in the system. These libraries provide the necessary code to run programs.

`/mnt`

The `mnt` directory is used to mount other file systems, such as file systems on a USB drive.

`/opt`

The `opt` directory is typically used to store optional software packages. This can include non-standard programs or programs that are not included in the Operating System.

`/proc`

The `proc` directory is a virtual file system that contains information about the running system, such as system processors, system memory, processes and devices.

`/root`

The home directory or the privileged user root.

`/sbin`

The `sbin` directory (which is shorthand for static binary) contains executable files for system administration and maintenance.

`/tmp`

The `tmp` directory is used to store temporary files. These files are deleted after a reboot. However, another temporary store is that of `/var/tmp`, which is persistent across reboots.

`/usr`

The `usr` directory is known as UNIX System Resources and contains user-level programs and libraries.

`/var`

The `var` directory contains files that vary in size and content, such as log files (in `/var/log`) or databases (typically in `/var/spool`). This directory is typically used to store data that changes over time.

The following is a visual overview of a typical directory hierarchy structure with the `tree` command. However, it is not typically installed by default and requires installation with the `apt` command (which is covered in more detail later).

```
$ sudo apt update
$ sudo apt install tree

$ sudo tree -d -L 2 /

/
├── bin -> usr/bin
├── boot
│   ├── grub
│   └── lost+found
├── cdrom
├── dev
│   ├── block
│   ├── bsg
│   ├── bus
│   ├── char
│   ├── cpu
│   ├── disk
│   ├── dri
│   ├── fd -> /proc/self/fd
│   ├── hugepages
│   ├── input
│   ├── mapper
│   ├── mqueue
│   ├── net
│   ├── pts
│   ├── shm
│   ├── snd
│   ├── ubuntu-vg
│   └── vfio
├── etc
│   ├── alternatives
```

```
│   ├── apparmor
│   ├── apparmor.d
│   ├── apport
│   ├── apt
│   ├── bash_completion.d
│   ├── binfmt.d
│   ├── byobu
│   ├── ca-certificates
│   ├── calendar
│   ├── cloud
│   ├── console-setup
│   ├── cron.d
│   ├── cron.daily
│   ├── cron.hourly
│   ├── cron.monthly
│   ├── cron.weekly
│   ├── cryptsetup-initramfs
│   ├── dbus-1
│   ├── dconf
│   ├── default
│   ├── depmod.d
│   ├── dhcp
│   ├── dpkg
│   ├── fonts
│   ├── fwupd
│   ├── groff
│   ├── grub.d
│   ├── gss
│   ├── init.d
│   ├── initramfs-tools
│   ├── iproute2
│   ├── iscsi
│   ├── kernel
│   ├── landscape
│   ├── ldap
│   ├── ld.so.conf.d
│   ├── libblockdev
│   ├── libnl-3
│   ├── logcheck
│   ├── logrotate.d
│   ├── lvm
│   ├── mdadm
│   ├── modprobe.d
│   ├── modules-load.d
│   ├── multipath
│   ├── netplan
│   ├── network
│   ├── networkd-dispatcher
│   ├── NetworkManager
│   ├── newt
│   ├── opt
│   ├── PackageKit
│   ├── pam.d
│   ├── perl
│   ├── pki
│   ├── pm
│   ├── polkit-1
│   ├── pollinate
│   ├── profile.d
│   ├── python3
│   ├── python3.8
│   ├── rc0.d
│   ├── rc1.d
│   ├── rc2.d
│   ├── rc3.d
│   ├── rc4.d
│   ├── rc5.d
│   ├── rc6.d
│   ├── rcS.d
│   ├── rsyslog.d
│   ├── security
│   ├── selinux
│   ├── skel
│   ├── sos
│   ├── ssh
│   ├── ssl
│   ├── sudoers.d
│   ├── sysctl.d
│   ├── systemd
│   ├── terminfo
│   ├── thermald
```

```
│   ├── tmp4inyim8q
│   ├── tmpfiles.d
│   ├── ubuntu-advantage
│   ├── udev
│   ├── udisks2
│   ├── ufw
│   ├── update-manager
│   ├── update-motd.d
│   ├── update-notifier
│   ├── UPower
│   ├── vim
│   ├── vmware-tools
│   ├── X11
│   └── xdg
├── home
│   └── spellegrino
├── lib -> usr/lib
├── lib32 -> usr/lib32
├── lib64 -> usr/lib64
├── libx32 -> usr/libx32
├── lost+found
├── media
├── mnt
├── opt
├── proc
│   ├── 1
│   ├── 10
│   ├── 102
│   ├── 105
│   ├── 11
```

...

```
│   ├── 90
│   ├── 91
│   ├── 93
│   ├── acpi
│   ├── asound
│   ├── bus
│   ├── driver
│   ├── fs
│   ├── irq
│   ├── net -> self/net
│   ├── pressure
│   ├── scsi
│   ├── self -> 2258
│   ├── sys
│   ├── sysvipc
│   ├── thread-self -> 2258/task/2258
│   └── tty
├── root
│   └── snap
├── run
│   ├── blkid
│   ├── cloud-init
│   ├── console-setup
│   ├── cryptsetup
│   ├── dbus
│   ├── fsck
│   ├── initramfs
│   ├── lock
│   ├── log
│   ├── lvm
│   ├── mount
│   ├── netns
│   ├── network
│   ├── NetworkManager
│   ├── screen
│   ├── sendsigs.omit.d
│   ├── shm -> /dev/shm
│   ├── snapd
│   ├── sshd
│   ├── sudo
│   ├── systemd
│   ├── tmpfiles.d
│   ├── udev
│   ├── udisks2
│   ├── user
│   └── uuidd
├── sbin -> usr/sbin
├── snap
```

```
│   ├── bin
│   ├── core18
│   ├── lxd
│   └── snapd
├── srv
├── sys
│   ├── block
│   ├── bus
│   ├── class
│   ├── dev
│   ├── devices
│   ├── firmware
│   ├── fs
│   ├── hypervisor
│   ├── kernel
│   ├── module
│   └── power
├── tmp
│   ├── snap.lxd
│   ├── systemd-private-a4de0be3b5a64f69a4da3c11f3fe35b3-systemd-logind.service-3bQrGf
│   ├── systemd-private-a4de0be3b5a64f69a4da3c11f3fe35b3-systemd-resolved.service-pRGARe
│   └── systemd-private-a4de0be3b5a64f69a4da3c11f3fe35b3-systemd-timesyncd.service-oJfrui
├── usr
│   ├── bin
│   ├── config
│   ├── games
│   ├── include
│   ├── lib
│   ├── lib32
│   ├── lib64
│   ├── libexec
│   ├── libx32
│   ├── local
│   ├── sbin
│   ├── share
│   └── src
└── var
    ├── backups
    ├── cache
    ├── crash
    ├── lib
    ├── local
    ├── lock -> /run/lock
    ├── log
    ├── mail
    ├── opt
    ├── run -> /run
    ├── snap
    ├── spool
    └── tmp

320 directories
```

Understanding the filesystem hierarchy is crucial for navigating and managing files. Each standard system directory serves a specific purpose, and knowing these can help you locate files and troubleshoot issues more effectively. Worth noting is that nearly everything in UNIX is a file as such.

Navigating the Filesystem

UNIX systems provide a number of commands for navigating the filesystem, whereby the most commonly used commands for this purpose are `cd` and `ls`. The `cd` command is used to change directories. For example, running the command `cd /etc` will navigate the current working directory to the `/etc` directory. Issuing the command `cd` without any arguments will return the current working directory back to the home directory of the user. The same can be achieved with `cd ~` as well, therefore `cd`, `cd ~` and `cd /home/$USER` are all interchangeable. Also, executing `cd -` (minus) will return back to the previous working directory from the current working directory. To navigate to the parent directory issue the command `cd ..` (dot-dot). The `ls` command is one of the most fundamental and useful commands in UNIX. It is used for listing the contents of a directory. It can be used to list all of the files and directories in the current directory or to list the contents of a specified directory. For example, if in a home directory and the command `ls` is executed then a list of files and directories in that home directory will be presented as output from the command.

Synopsis: `ls <options> <path>`

ls will list the contents of the current directory if no path is specified.

The ls command has many options that can be used to modify the output. Here are some of the most common ones:

-a	list all files and directories, including hidden files and directories (denoted with a **.** prefix to the file or directory name)
-l	list files and directories in long format, with permissions and other information
-h	display file sizes in human-readable format (1K instead of 1024 bytes) when the -l option is used
-R	list all files and directories recursively (including all subdirectories)

To list the contents of the current directory in long format:

```
$ ls -l
```

To list all files and directories (including hidden ones) in the /etc directory:

```
$ ls -a /etc
```

To list all files and directories in the current directory recursively:

```
$ ls -R
```

A very useful file and directory listing that shows a long format in reverse chronological order, therefore the last line of output being the most recent file or directory, is as follows:

```
$ ls -l -t -r
```

The exact same result can be achieved by clustering the flag options together as follows:

```
$ ls -ltr
```

Managing Files and Directories

There are a series of commands to create, delete, move and copy files or directories. The most common commands for the purpose of managing files and directories are:

- To create a file use the command: touch <*filename*>
- To create a directory use the command: mkdir <*directory*>
- To delete a file use the command: rm <*filename*>
- To delete a directory and its contents use the command: rm -r <*directory*>
- To move or rename a file or directory use the command: mv <*source-file*> <*destination-file*>
- To copy a file use the command: cp <*source-file*> <*destination-file*>
- To copy a directory use the command: cp -r <*source-directory*> <*destination-directory*>

The mkdir command is used to create a new directory. For example, if in a home directory and the command mkdir test is executed then a new directory called test will be created in the home directory. However, if specifying more than one directory with subdirectories the mkdir command should be invoked with a -p as a flag option, therefore the command in full will be mkdir -p test1/test2/test3 in order to create the intermediate directories of test3 in test2 in test1. Also, notice that the mkdir command was invoked with what is known as an option, also referred to as a flag or a switch, that being -p (meaning create intermediate directories). More about command flags will be covered later.

The rm command is used to delete files and/or directories. For example, if in a home directory and executing the command rm test the test directory will **not** be deleted accordingly. This is because the rm command needs to be issued with recursion. However, if the directory of concern is empty then issuing the command rmdir test will remove the directory (a safer approach), otherwise if the directory has additional files and directories then issuing rm -r test will recursively remove the directory and all its contents. However, always be careful when using rm -r <*directory*>. This is especially true when issuing rm -r ./* (*dot* preceding *slash* followed by a wildcard glob *asterix*), which could potentially be a typo like rm -r /.* (*slash* preceding *dot* followed by a wildcard glob *asterix*) resulting in potentially damaging the operation of the system. This command was executed with -r meaning a shorthand to invoking the rm command with recursive behaviour, although grouping the flags -r with -i is a safer option as -i means interactive and requires confirmation of deletion. Grouping flags can be done in this case as -ri or interchangeably -ir, therefore the command will be rm -ri <*directory*>.

The `mv` command is used to both move or rename files or directories. For example, if in a home directory and the command `mv test junk` is executed, the `test` directory will be renamed to the `junk` directory. If there is a `junk` directory and file `testfile` in the current working directory, then `mv testfile junk/` will move the file `testfile` to the directory `junk`. Just to reiterate, suppose another directory called `tmp` is created and `mv junk tmp` is executed, then the `junk` directory and its contents are moved into the directory named `tmp`. Worth noting is that UNIX/UNIX-like systems are case-sensitive, therefore `tmp`, `Tmp` and `TMP` will all be different file or directory names!

The `cp` command is similar to the `mv` command but copies rather than move/rename files or directories. To copy a file in the same working directory could be done as `cp file1 file2` or `cp file1 dir1/` or `cp file1 dir1/file1copy`. When copying a directory, the recursive flag `-r` is needed. Therefore, the command will be `cp -r dir1/ tmp/` will recursively copy `dir1` into the `tmp` directory. Worth noting is the `-p` flag, which preserves attributes in the source file being copied.

To search and locate files (for example all files ending in `.conf`) the following command is suitable for that task:

```
$ sudo find / -name *.conf -print
```

With the `find` command it is possible to list those files that are executable or have a `setuid` (executable as a privileged user):

To use the `find` command to locate executable files requires using the following syntax:

```
$ find <path> -type f -executable
```

This command will search the specified `<path>` for any files that are executable and print them to standard output.

To use the find command to locate setuid files requires using the following syntax:

```
$ find <path> -type f -perm -4000
```

This will search the specified `<path>` for any files that have the `setuid` permission bit set, which is represented by the value `4000`. The `setuid` is a permission setting on executable files that allows the file to be run with privileges. This is often used to allow users to access system functions that are normally only available to the root user. An example of this would be updating a user password with the `passwd` command.

`ln` is a command used to create hard and symbolic links.

A hard link is a link to a file on the same filesystem, and a symbolic link is a link to another filesystem, or to a file or directory on the same filesystem.

To create a hard link use the following syntax:

```
$ ln -T <target> <linkname>
```

Where `<target>` is the name of the file to link against and `<linkname>` is the name of the link to create.

To create a symbolic link use the following syntax:

```
$ ln -s <target> <linkname>
```

Where `<target>` is the name of the file or directory to link against and `<linkname>` is the name of the link you want to create.

Use the `-f` option to force the creation of a link if the file or directory already exists.

The `ls -l` command lists the contents of a directory in a long format, which displays additional information about the files and directories. The output of the command will include the permissions, the number of hard links, the owner, the group, the size in bytes, the date and time of the last modification and the name of the file or directory. To read the permissions of an item, the first column of the `ls -l` output will provide the information. Permissions are presented in groups of three. After the first character, the first group of three characters represents the permissions for the owner of the item, then the second group of three characters represents the permissions for the group and the third (last) group of characters represents the permissions for all other users. The characters can be `r` (read), `w` (write), `x` (execute) or `-` (no permission). For example, `rwxr-xr--` represents read, write and execute permission for the owner, read and execute permission for the group, and just read permission for other users (world). A prefixed `d` character, that being the first character of the permissions, indicates that the item is a directory. NOTE: The minimum permissions required to access a directory are both read and execute, most importantly is the execute permission as an absolute requirement regarding directory access per permissions group.

When working with a group of permissions this can be summarised by adding together ALL the octal (base 8) values together in accordance with the their values associated with each `rwx` statement, where read = `4`, write = `2` and execute = `1`. This can be summarised as follows:

```
                      Octal value

r = read                     4
w = write                    2
x = execute                  1
- = no permission            0

u = user
g = group
w = world

        u   g   w     Permission

       rwx rwx rwx           777
       rwx rwx r-x           775
       rwx r-x r-x           755
       rw- rw- r--           664
       rw- r-- r--           644
       rw- --- -             600
```

VERY IMPORTANT NOTE: An octal value that is an odd number (as opposed to even) means the execute permission has been set.

The `chmod` command (known as *change mode*) is used to change the access permission of one or more files and directories. For example, to give a user read, write, and execute permission to a file called `file.txt` then run the following command:

```
$ chmod u+rwx file.txt
```

This command uses the `u` option to indicate that the permissions are being assigned to a user, the `+` symbol to indicate that the permissions should be added, and the `rwx` to indicate that the user should be given read, write and execute permission. Alternatively, the same outcome could be achieved with octal values as follows:

```
$ chmod 777 file.txt
```

To change the file permissions to read and write for the user, read for the group and read for world would be done with:

```
$ chmod 644 file.txt
```

To apply the changes recursively then invoke the `chmod` command with the `-R` flag option.

The chown command (known as *change ownership*) is used to change the owner and/or group of one or more files and directories. For example, to change the ownership of the `file.txt` file to the user `jsmith` then run the following command:

```
$ chown jsmith file.txt
```

Another example is invoking the `chown` command recursively and changing the group as well by executing:

```
$ chown -R jsmith:staff src/
```

What this does is change the username to `jsmith` and group to `staff` for the directory `src` and all files and directories contained within the `src` directory.

Files and Filesystems

The command `file` is used to determine the type of a file. It inspects the file and determines what type of file it is, such as an executable, an image file, a text file, etc. The command is used as follows:

```
$ file <options> <filename>
```

The command `du` (stands for *disk usage*) is used to show the amount of disk space used by a particular directory or file. It can also be used to show the total amount of disk space used by all files in a directory. The command is used as follows

```
$ du <options> <filename>
```

The command `df` (stands for *disk free*) is used to show the amount of block storage space available on the system, as well as the amount of block device space used by each mounted file system. It can be used to show the size of a mounted filesystem or all filesystem mount points. The command is used as follows:

```
$ df <options> <filesystem>
```

Therefore, `df -h /` will show a human-readable output of the root filesystem, including size, as available and use %.

The `watch` command is used to run a command at regular intervals and display the output in the terminal. Therefore, `watch df -h` command is used to monitor the amount of available block storage space on a UNIX system. The `df` command is used to display the amount of available storage space, and the `-h` flag is used to display the output in a human-readable format.

The command `lsof` (stands for *list open files*) and is used to show a list of open files and their associated processes. It can be used to show information about the files and the processes which are using them.

The command `mount` is used to mount a filesystem or device in the system. It is used to make a filesystem available for access by the system and users. The command is used as follows:

```
$ mount <options> <device> <mount-point>
```

The command `hdparm -tT /dev/sda` is used to test the performance of a hard disk drive. The `-t` flag is used to run a read speed test on the drive, and the `-T` flag is used to run a cache speed test on the drive. The argument `/dev/sda` should be replaced by the actual device name of the drive being tested.

The `badblocks -s /dev/sda` command is used to search for bad blocks on a hard disk drive. The `-s` flag is used to enable the creation of a status report of the bad blocks found. The `/dev/sda` argument should be replaced by the actual device name of the drive being tested.

The `lsusb` command is used to list all USB buses and devices connected to the system. It displays information about the devices, including the vendor name, product name and device class.

The command `lspci` is used to list all PCI buses and devices in the system. It displays information about the devices, including the vendor name, device name and device class.

Working with Text Files

The command-line also provides a number of commands for working with text files. The most common commands for this purpose are `cat`, `head`, `less`, `more` and `tail`.

The `cat` command is used to display the contents of a text file. For example, if in a home directory and running the command `cat test.txt` the contents of the `test.txt` file will be displayed. Furthermore, the `cat` command (known as *concatenate*), therefore allows multiple files to be joined together with this command, so an example command demonstrating this would be `cat file1 file2 file3`. This will consolidate all files in the order specified and output like one continuous file. Given this, a simple example of redirection will now be referenced. Basically, to redirect output to a file would involve executing `cat file1 file2 file3 > file4`, therefore creating a new file `file4` with the concatenated contents of the previous three files. Using the `>` character will always create or overwrite the file specified. However, the characters `>>` will append as opposed to overwrite the file specified for redirection. Therefore, the command `echo FUBAR >> file4` will append the string `FUBAR` to the end of the existing file contents of `file4` without overwriting its existing contents. The topic of redirection will be covered in more detail later.
The `head` command is used to display the first several lines of a text file. For example, if in a home directory and running the command `head test.txt` then the first ten lines of the `test.txt` file will be displayed.

The opposite is also true in that the `tail` command is used to display the last several lines of a text file. For example, if in a home directory and running the command `tail test.txt` the last ten lines of the `test.txt` file will be displayed.

A great feature of the tail command is the flag option `-f`, which does not stop at the end-of-file (EOF) but waits for additional text to output from appended future updated input. A great example of this is when monitoring log file updates in near realtime, such as:

```
$ tail -f /var/log/syslog
Jan 19 14:20:07 shroom systemd[1]: Started PackageKit Daemon.
Jan 19 14:21:46 shroom systemd[1]: Starting Ubuntu Advantage APT and MOTD Messages...
Jan 19 14:21:47 shroom systemd[1]: ua-messaging.service: Succeeded.
Jan 19 14:21:47 shroom systemd[1]: Finished Ubuntu Advantage APT and MOTD Messages.
Jan 19 14:27:23 shroom PackageKit: daemon quit
Jan 19 14:27:23 shroom systemd[1]: packagekit.service: Succeeded.
Jan 19 14:33:09 shroom systemd[1]: Starting Cleanup of Temporary Directories...
Jan 19 14:33:09 shroom systemd[1]: systemd-tmpfiles-clean.service: Succeeded.
Jan 19 14:33:09 shroom systemd[1]: Finished Cleanup of Temporary Directories.
```

[Press `ENTER` key several times to scroll output with blank lines as a means to divide old and new output]

On another terminal session execute the command `logger FUBAR` whilst looking at the terminal running with the active `tail -f` command:

```
Jan 19 14:20:07 shroom systemd[1]: Started PackageKit Daemon.
Jan 19 14:21:46 shroom systemd[1]: Starting Ubuntu Advantage APT and MOTD Messages...
Jan 19 14:21:47 shroom systemd[1]: ua-messaging.service: Succeeded.
Jan 19 14:21:47 shroom systemd[1]: Finished Ubuntu Advantage APT and MOTD Messages.
Jan 19 14:27:23 shroom PackageKit: daemon quit
Jan 19 14:27:23 shroom systemd[1]: packagekit.service: Succeeded.
Jan 19 14:33:09 shroom systemd[1]: Starting Cleanup of Temporary Directories...
Jan 19 14:33:09 shroom systemd[1]: systemd-tmpfiles-clean.service: Succeeded.
Jan 19 14:33:09 shroom systemd[1]: Finished Cleanup of Temporary Directories.



Jan 19 15:00:06 jsmithpc jsmith: FUBAR
```

Further updates to the file will output via `tail -f` in near-realtime accordingly. NOTE: This is an important command and `/var/log/syslog` is essential to reference for system log messages.

Both the `more` and `less` commands are known as pagers. They are useful when output is more than the size of the terminal window.

The `more` command is used to view the content of a text file one page at a time when viewing long text files that may exceed the height of the terminal size. An example could be `more /etc/passwd` in order to display the contents of the `/etc/passwd` file one page at a time.

The `less` command is also used to view the content of a text file one page at a time, just like the `more` command. However, it is more versatile than `more` and is often preferred by experienced users. An example could be `less /etc/passwd` in order to display the contents of the `/etc/passwd` file one page at a time.

However, what must be known is the basic navigation commands when using `more` or `less`. Use the ↑ (up) and ↓ (down) arrow keys to scroll through the contents one line at a time. To page the contents one page at a time press the `spacebar` to go forward and the `b` key to go backwards. To navigate to the top of the page then press the `g` key and to the bottom press the `G` key. To search for a specific keyword type `/` to forward search and `?` to search backwards. To cycle the search for more than one keyword match press `n` and `N` keys respectively. If it is a forward search then `n` will be a forward search on the keyword, whereas if it is a backward search then `n` will be a backward search on the keyword. The opposite behaviour to `n` is the `N` key (the opposite of the search direction). Finally regarding pagers is that to quit and exit press the `q` key.

The `script` command is used to record all the output of a terminal session. It is useful for debugging, logging or creating tutorials. It can also be used to create a text file version of output from a command.

Redirection and File Descriptors

I/O (Input/Output) redirection is the process of redirecting the standard I/O of a program to a different source or destination. The `<` and `>` symbols are used to redirect standard input and output, respectively. The `>>` symbol is used to redirect and append the standard output to a file.

An example of redirection for the output of a program to a file use the `>` symbol like `ls > output.txt`. To append at the end of the file contents use `ls >> output.txt`. NOTE: With the `>` symbol this will overwrite the file as opposed to the `>>` symbol, which will append to the end of the file.

In UNIX, the terminal I/O is referred to with the file descriptors `0`, `1` and `2`, which are `stdin` (standard input), `stdout` (standard output) and `stderr` (standard error), respectively. These file descriptors are used by the shell and other UNIX programs to read and write data to the terminal or other I/O devices.

Also, worth noting is that a file descriptor is an integer (whole number) value that is used to reference an open file in UNIX. It is one of the main ways programs interact with files, sockets and other system resources. For example, when a program opens a file, it is assigned a file descriptor that is used to refer to it. The program can then use the file descriptor to read and write data to the file.

Another example of a file descriptor in UNIX is when a program opens a network socket. The socket is assigned a file descriptor, which can be used to read data from and write data to it. This is especially useful for network programming since the file descriptor allows a program to interact with a remote machine. Sockets are treated as files accessed through file descriptors to allow socket operations to use the same system calls, such as `open`, `read`, `write` and `close`, that are used for file operations.

Furthermore, when a program creates a pipe that passes data from one process to another, a file descriptor is used to refer to the pipe. This file descriptor is used to read data from the pipe, write data to the pipe and detect when one of the processes has finished.

`stdin` is defined as standard input and is the file descriptor for the standard input stream. It is used for reading data from the keyboard or from a program to another program. For example, `cat < /dev/stdin` to read from the keyboard.

`stdout` is defined as standard output and is the file descriptor for the standard output stream. It is used for writing data from one program to another. For example, `ls > /dev/stdout` to write to the terminal.

`stderr` is defined as standard error and is the file descriptor for the standard error stream. It is used for writing error messages from one program to another. For example, `ls 2> /dev/stderr` to write error messages to the terminal.

`<command> 2>&1` (`stderr` to `stdout`). This redirects the standard error stream to the standard output stream. This is useful when you want to have both the standard output and standard error messages written to the same destination. For example, if you wanted to write both the standard output and error messages of a program to a file then use `<command> 2>&1 > output.txt`.

Redirection allows management of input and output of a command, which is useful for logging, automation and chaining commands together.

A pipe `|` is used to take the output of one command as input to another. It is most commonly used to take the output of one command and use it as the input of another command. For example, the command below will pipe the output of the `cat` command listing the contents of multiple files, to the `wc -l` command, which will count the number of lines of output:

```
$ cat *.cpp *.h | wc -l
```

Another example is listing the disk usage in kilobytes of all files in the current working directory piped from `du` to the `sort` command in reverse numerical order:

```
$ du -sk ./* | sort -rn
```

The `grep` command (known as *global regular expression print*) is used to search for a pattern or a string (a collection of alphanumeric characters) of text within a file or multiple files. For example, the command below will search for the string `main` within all the `.cpp` and `.h` files in the current directory:

```
$ grep main *.cpp *.h
```

The command syntax for `grep` is as follows:

```
$ grep <options> <pattern> <files>
```

The pattern argument can be a regular expression, which is a way of describing a set of strings (more on regular expressions later). For example, the most basic regular expression `abc` will match any line with the string `abc` in it an dprint it as output.

The most commonly used options are `-i` (ignore case) and `-v` (invert match). The `-i` option causes `grep` to ignore case when searching, while the `-v` option causes it to return only lines that do not match the pattern.

`grep` is powerful and incredibly versatile. It can be used for a variety of tasks that can save time and effort when searching through files or streams of data. For example, it can be used to search for particular words or phrases in log files, to locate files that contain certain text or to find duplicate lines in a file. It can also be used to find the number of occurrences of a particular string in a file.

The `cut` command can be used to extract a particular field from a file, such as the `/etc/passwd` file. This file contains information about each user that is registered on a UNIX system, such as their username, user ID, primary group ID, home directory and shell. The `cut` command can be used to extract any one of these fields from the file.

For example, to extract the username from the `/etc/passwd` file then use the following command:

```
$ cut -d : -f 1 /etc/passwd
```

This command uses the `-d` option to specify that the field delimiter used in the file, which in this case is a `:` (colon) and the `-f` option to specify the desired field number, for example, 1 for the username as the first field in `/etc/passwd`.

Another example to extract the UID field from the `/etc/passwd` file is:

```
$ cut -d : -f 3 /etc/passwd
```

Users and Groups

The `id` command is used to display the user identity, group identity and group membership of a user. It is used to print the user ID (UID) and group ID (GID) of the current user, along with any supplementary group IDs.

The `w` command is used to print information about users who are currently logged into the system. It will display the login name of each user, the time of the login, the load average of the system and the host from which the user is logged in.

The `who` command is used to list information about users who are currently logged into the system. It will display the login name of each user, the time of the login and the host from which the user is logged in.

The `whoami` command is used to display the username of the user who is currently logged in.

```
$ whoami
jsmith

$ sudo -i
# whoami
root
# id
uid=0(root) gid=0(root) groups=0(root)
```

The command `sudo` is to execute a command as another user, typically the root user. The command `su -` means switch to the root user and inherit their environment.

User/group administration commands (see corresponding man pages) on a UNIX/UNIX-like system include:

```
$ sudo useradd <username>
$ sudo userdel -r <username>
$ sudo usermod <option> <username>
$ sudo usermod -aG <groupname> <username>
$ sudo groupadd <groupname>
$ sudo groupdel <groupname>
```

Date and Time

The following shows commands relating to date and time accordingly:

```
$ cal 12 2022
   December 2022
Su Mo Tu We Th Fr Sa
             1  2  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30 31
```

```
$ date
Thu 19 Jan 2023 21:56:42 GMT
```

The system date and time can be set with the `date` command with certain arguments but also via `ntp` (Network Time Protocol), a distributed time service calibrated with an atomic clock.

System Management

System management in UNIX involves monitoring, maintaining and configuring the system to ensure optimal performance and reliability. This section provides detailed descriptions and usage examples for key system management commands.

To display system information:

```
$ uname -a
```

To display the system message buffer for debugging hardware and driver issues use the following:

```
$ dmesg | less
```

To clear the message buffer:

```
$ sudo dmesg -C
```

To show how long the system has been running, the number of users and the system load averages execute:

```
$ uptime
```

To list the last logins of users:

```
$ last
```

To display the last system reboots execute:

```
$ last reboot
```

System shutdown and reboot involves a series of commands, such as performing an immediate shutdown as follows:

```
$ sudo shutdown -h now
```

To schedule a shutdown:

```
$ sudo shutdown -h +10 "System will shutdown in 10 minutes."
```

To reboot the system:

```
$ sudo reboot
```

To power off the system:

```
$ sudo poweroff
```

For log management queries and to display messages from the `systemd` journal issue the following command:

```
$ journalctl -xe
```

To view logs for a specific unit:

```
$ journalctl -u <unit>
```

Adding entries to the system log can be done as follows:

```
$ logger "This is a test log entry"
```

Regarding service management to control a `systemd` system and service manager the following is useful.

To start a service:

```
$ sudo systemctl start <service>
```

To stop a service:

```
$ sudo systemctl stop <service>
```

To restart a service:

```
$ sudo systemctl restart <service>
```

To enable a service to start on boot:

```
$ sudo systemctl enable <service>
```

To disable a service from starting on boot:

```
$ sudo systemctl disable <service>
```

To check the status of a service:

```
$ sudo systemctl status <service>
```

Hardware Information can be accessed with the following series of commands.

To list detailed information about all hardware:

```
$ sudo lshw
```

To list information about block devices:

```
$ lsblk
```

To display information about the CPU architecture:

```
$ lscpu
```

Listing USB devices can be done with:

```
$ lsusb
```

Similarly, to list all PCI devices:

```
$ lspci
```

Disk management tools begins with reporting file system block storage space usage as follows:

```
$ df -h
```

For estimating file space usage use:

```
$ du -sh <directory>
```

To mount a file system to the directory `/mnt` is done as follows:

```
$ sudo mount /dev/sdX1 /mnt
```

To unmount this file system is done as follows:

```
$ sudo umount /mnt
```

To typically check and repairs a file system involves issuing the following command:

```
$ sudo fsck /dev/sdX1
```

To display running tasks on the system can be done with:

```
$ top
```

An extension of `top` as an interactive process viewer for UNIX/UNIX-like systems requires execution of the following command:

```
$ htop
```

Kernel management and to configure kernel parameters at runtime the following can be done with:

```
$ sudo sysctl -a
```

To change a kernel parameter:

```
$ sudo sysctl -w net.ipv4.ip_forward=1
```

To add and remove modules from the kernel is as follows:

```
$ sudo modprobe <module>
```

```
$ sudo modprobe -r <module>
```

To show the status of modules in the kernel:

```
$ lsmod
```

These command-line tools provide the foundation for effective system management on UNIX/UNIX-like systems, ensuring administrators can maintain system performance, security and reliability.

Package Management

Package management is essential for maintaining and updating software on UNIX/UNIX-like systems. This section covers the various commands used in package management, focusing on the `dpkg` format system with APT (Advanced Package Tool) repositories. The `apt` command ensures dependencies are resolved and obsolete packages are removed.

To update the package list of available packages and their versions without installing or upgrading any packages then issue the following command:

```
$ sudo apt update
```

To upgrade the packages after `sudo apt update` has been executed then the following command will install the newest versions of all currently installed out-of-date packages:

```
$ sudo apt upgrade
```

For upgrading to a new distribution release from the software vendor then issue the following command:

```
$ sudo apt dist-upgrade
```

To install a package along with its dependencies then do the following:

```
$ sudo apt install <package>
```

To remove a package but keep its configuration files issue the following command:

```
$ sudo apt remove <package>
```

To remove a package along with its configuration files then execute the following:

```
$ sudo apt purge <package>
```

The following command allows searching for packages in the APT repositories:

```
$ apt search <keyword>
```

To display detailed information about a specific package then execute the following command:

```
$ apt show <package>
```

For listing all installed packages run the following command:

```
$ dpkg -l
```

To show the status of an installed package run:

```
$ dpkg -s <package>
```

To fix broken dependencies run:

```
$ sudo apt --fix-broken install
```

To remove packages that were automatically installed to satisfy dependencies for other packages and are now no longer needed then execute the following command:

```
$ sudo apt autoremove
```

For clearing out the local repository of retrieved package files (which can save block storage device space) then issue the following command:

```
$ sudo apt clean
```

Low-level package management is achieved with the `dpkg` command, whereby to reconfigure an unpacked package then use:

```
$ sudo dpkg --configure <package>
```

To install a package from a local `.deb` file then this can be done as follows:

```
$ sudo dpkg -i <file>.deb
```

To remove an installed package then do the following:

```
$ sudo dpkg -r <package>
```

To purge an installed package, removing configuration files as well, then execute:

```
$ sudo dpkg -P <package>
```

Understanding and using these package management commands ensures that UNIX/UNIX-like systems, in this instance a Linux system due to using APT, are kept up-to-date and secure, with software dependencies properly managed and obsolete packages efficiently removed.

Process Management

The `init` process is the initial process in a UNIX-based Operating System. It is responsible for starting, stopping and managing daemons (background processes) and other system services. When the system is powered up, the `init` process is the first process to be created by the kernel and is given a process ID of 1. It then proceeds to read the system configuration files and execute the configured run levels. During the startup process, `init` reads the `/etc/inittab` configuration file to determine which run level to boot into, and then proceeds to launch the processes associated with that run level. `init` also handles signals sent to the system, such as shutdown and reboot requests. `init` also monitors the health of all other processes running on the system and can restart them if necessary therefore `init` serves as a watchdog to ensure that the system is running properly and to take corrective action if any problems are detected.

```
$ pgrep -lf init
1 systemd
```

`pgrep` is a command-line tool used to search for processes running on a system. It scans through the process table and looks for user-specified patterns. The user can set various criteria including process name, username, terminal or environment variables.

The command `pstree` is a tool used to display the process hierarchy in a tree format. It shows the parent-child relationships between each process and their respective PIDs. It is useful for understanding the inter-process communication within the system.

```
$ sudo pstree -lpsn 1
systemd(1)-+-systemd-journal(359)
           |-systemd-udevd(386)
           |-multipathd(510)-+-{multipathd}(511)
           |                 |-{multipathd}(512)
           |                 |-{multipathd}(513)
           |                 |-{multipathd}(514)
           |                 |-{multipathd}(515)
           |                 `-{multipathd}(516)
           |-systemd-timesyn(557)---{systemd-timesyn}(566)
           |-systemd-network(774)
           |-systemd-resolve(776)
           |-accounts-daemon(801)-+-{accounts-daemon}(807)
           |                      `-{accounts-daemon}(868)
           |-cron(805)
           |-dbus-daemon(806)
           |-networkd-dispat(815)
           |-rsyslogd(817)-+-{rsyslogd}(830)
           |               |-{rsyslogd}(831)
           |               `-{rsyslogd}(832)
           |-snapd(818)-+-{snapd}(921)
           |            |-{snapd}(922)
           |            |-{snapd}(923)
           |            |-{snapd}(924)
           |            |-{snapd}(929)
           |            |-{snapd}(930)
           |            |-{snapd}(948)
           |            |-{snapd}(954)
           |            |-{snapd}(956)
           |            `-{snapd}(962)
           |-systemd-logind(820)
           |-udisksd(822)-+-{udisksd}(827)
           |              |-{udisksd}(869)
           |              |-{udisksd}(895)
           |              `-{udisksd}(905)
           |-atd(823)
           |-unattended-upgr(872)---{unattended-upgr}(919)
           |-polkitd(882)-+-{polkitd}(886)
           |              `-{polkitd}(888)
           |-sshd(1344)
           |-agetty(2611)
           |-login(2910)---bash(3022)---sudo(3422)---pstree(3423)
           `-systemd(3015)---(sd-pam)(3017)
```

The command `ps` is a utility used to display information about currently running processes. It displays the process ID, parent process ID, memory used, CPU time used and command used to run the process. This can be used to monitor processes, troubleshoot performance issues, yet also to stop or kill processes.

```
$ ps -elf
F S UID      PID    PPID  C PRI  NI ADDR SZ WCHAN  STIME TTY          TIME CMD
4 S root       1       0  0  80   0 - 25800 -      14:18 ?        00:00:02 /sbin/init
1 S root       2       0  0  80   0 -     0 -      14:18 ?        00:00:00 [kthreadd]
1 I root       3       2  0  60 -20 -     0 -      14:18 ?        00:00:00 [rcu_gp]
1 I root       4       2  0  60 -20 -     0 -      14:18 ?        00:00:00 [rcu_par_gp]
1 I root       6       2  0  60 -20 -     0 -      14:18 ?        00:00:00 [kworker/0:0H-kblockd]
1 I root       8       2  0  60 -20 -     0 -      14:18 ?        00:00:00 [mm_percpu_wq]
1 S root       9       2  0  80   0 -     0 -      14:18 ?        00:00:00 [ksoftirqd/0]
1 I root      10       2  0  80   0 -     0 -      14:18 ?        00:00:03 [rcu_sched]
1 S root      11       2  0 -40   - -     0 -      14:18 ?        00:00:00 [migration/0]
5 S root      12       2  0   9   - -     0 -      14:18 ?        00:00:00 [idle_inject/0]
```

```
1 S root     14       2  0  80   0 -     0 -      14:18 ?        00:00:00 [cpuhp/0]
5 S root     15       2  0  80   0 -     0 -      14:18 ?        00:00:00 [kdevtmpfs]
1 I root     16       2  0  60 -20 -     0 -      14:18 ?        00:00:00 [netns]
1 S root     17       2  0  80   0 -     0 -      14:18 ?        00:00:00 [rcu_tasks_kthre]
1 S root     18       2  0  80   0 -     0 -      14:18 ?        00:00:00 [kauditd]
1 S root     19       2  0  80   0 -     0 -      14:18 ?        00:00:00 [khungtaskd]
1 S root     20       2  0  80   0 -     0 -      14:18 ?        00:00:00 [oom_reaper]
1 I root     21       2  0  60 -20 -     0 -      14:18 ?        00:00:00 [writeback]
1 S root     22       2  0  80   0 -     0 -      14:18 ?        00:00:00 [kcompactd0]
1 S root     23       2  0  85   5 -     0 -      14:18 ?        00:00:00 [ksmd]
1 S root     24       2  0  99  19 -     0 -      14:18 ?        00:00:00 [khugepaged]
1 I root     70       2  0  60 -20 -     0 -      14:18 ?        00:00:00 [kintegrityd]
1 I root     71       2  0  60 -20 -     0 -      14:18 ?        00:00:00 [kblockd]
1 I root     72       2  0  60 -20 -     0 -      14:18 ?        00:00:00 [blkcg_punt_bio]
1 I root     73       2  0  60 -20 -     0 -      14:18 ?        00:00:00 [tpm_dev_wq]
1 I root     74       2  0  60 -20 -     0 -      14:18 ?        00:00:00 [ata_sff]
1 I root     75       2  0  60 -20 -     0 -      14:18 ?        00:00:00 [md]
1 I root     76       2  0  60 -20 -     0 -      14:18 ?        00:00:00 [edac-poller]
1 I root     77       2  0  60 -20 -     0 -      14:18 ?        00:00:00 [devfreq_wq]
```

The `nice` command is used to change the default priority of a process. It helps to determine how much CPU time a process gets. To use the `nice` command, run it with the desired priority level as an argument, such as:

```
$ nice -n 10 <program>
```

This will set the priority of a new program process to `10`. The priority level range is from `-20` (most favourable process priority) to `19` (least favourable process priority).

The `renice` command is used to change the priority of an existing process. It works in a similar way to `nice` except that you need to specify the PID of the process that you want to change. For example:

```
$ renice -n 10 1234
```

This will change the priority of the process with PID `1234` to `10`.

The `top` command is used to view the resource utilisation of processes running on a UNIX system. It displays a list of processes, along with their CPU and memory usage and a breakdown of their states. It also allows users to kill processes, change their priority, and more.

The `htop` command is similar to `top` but it is more user-friendly and provides more information, such as the command of the process, its environment variables, and more. It also has a good interface and allows users to filter processes, set CPU affinity, an so on.

The `screen` command is used to create virtual terminals on a UNIX system. It allows users to keep multiple programs running at the same time, even if they are disconnected from the terminal. It also allows users to reconnect to their virtual terminals from another location. Screen is a terminal multiplexer, which means that it allows multiple virtual terminals to be open on a single system. Using `screen` in a production environment is ideal for persistent server processes.

The execution of `./<command> &` is used to run a program in the background, asynchronously from the current process. When this command is used, the program is executed without waiting for its completion, allowing the current process to continue uninterrupted. This is especially useful for long running processes that are not dependent on the current process.

The following is an example of executing the bit-nibbler `dd` as an asynchronous process (denoted by the trailing `&` character) and also showing the variety of commands to terminate the background `dd` process:

```
$ dd if=/dev/random of=random.txt bs=1G count=256 &
[1] 3751
$ pgrep -lf random
3751 dd
$ pkill -f random
$ pgrep -lf random
[1]+  Terminated              dd if=/dev/random of=random.txt bs=1G count=256

$ dd if=/dev/random of=random.txt bs=1G count=256 &
[1] 3757
$ pgrep -lf random
3757 dd
$ pkill -9 -f random
```

```
$ pgrep -lf random
[1]+  Killed                  dd if=/dev/random of=random.txt bs=1G count=256

$ dd if=/dev/random of=random.txt bs=1G count=256 &
[1] 3764
$ jobs
[1]+  Running                 dd if=/dev/random of=random.txt bs=1G count=256 &
$ kill %1
$ pgrep -lf random
[1]+  Terminated              dd if=/dev/random of=random.txt bs=1G count=256
$ dd if=/dev/random of=random.txt bs=1G count=256 &
[1] 3766
$ kill 3766
$ pgrep -lf random
[1]+  Terminated              dd if=/dev/random of=random.txt bs=1G count=256

$ head random.txt
ӳ<FA>J<A1>H<B0><D0>}~<BD>1ɗ<C1><BB><ED><F3><B4><97>&<B3>$,\y<CA>=g<98>]
                                                                        '9<BA>iK<EE><D6>f<B1>
<FA>|<84><8A><A3>[<BA><F8>|<A0>l]Nt<8C><88><C0><E6><D9>N<C4>AfW<FA>.<F9><A5>eU<A2>X<86><E1>b
```

Garbled terminal output requires a reset as follows:

```
$ reset
```

The `kill` command is a signal handler for processes.

```
$ kill <signal> <PID>
```

```
$ kill -1 <PID>              - SIGHUP
$ kill -2 <PID>              - SIGINT
$ kill -9 <PID>              - SIGKILL
```

Process signals can be seen in much further detail via:

```
$ man signal
```

The commands `pkill` and `killall` take a process name as an argument in order to invoke the signal.

Networking

Networking in UNIX is managed through a combination of system commands and configuration files. This section covers essential commands and tools for managing network settings, connections and diagnostics.

Hostname and domain information can be displayed with the following commands as follows:

```
$ hostname
```

```
$ domainname
```

Network configuration and diagnostic tools include the series of commands that follow.

The classic way to get and set configuration of network interfaces initially requires the following package to be installed:

```
$ sudo apt install net-tools
```

To display all the network interfaces after installing this package execute the following command:

```
$ ifconfig
```

However, the same can be achieved with the more modern command:

```
$ ip addr show
```

To test connectivity to a host use:

```
$ ping <ip-addr>
```

To capture and analyse network traffic requires using `sudo` to access network interfaces in promiscuous mode with the following well-established network analysis tool:

```
$ sudo tcpdump
```

DNS and lookup utilities include `whois`, `nslookup` and `dig`. To retrieve information about a domain name.on the internet use the command:

```
$ whois <domainname>
```

To query DNS to obtain domain name or IP address mappings.then execute:

```
$ nslookup <domainname>
```

- or -

```
$ dig -x <ip-addr>
```

Network monitoring and statistics can be obtained as follows accordingly.

To display network connections, routing tables, interface statistics, masquerade connections and multicast memberships then execute:

```
$ netstat -a
```

To display the kernel routing table execute:

```
$ netstat -rn
```

To provide detailed information about network sockets use the following command:

```
$ ss -tuln
```

For remote file transfers and remote logins is typically done using the commands `scp` (secure copy) and `ssh` (secure shell).

To securely copy files between hosts across the network from a local host to a remote host execute:

```
$ scp <file> user@hostname:/home/user
```

To copy a file from a remote host:

```
$ scp user@hostname:/home/user/<file>
```

To securely log into a remote machine and execute commands on that host accordingly use the secure shell as follows:

```
$ ssh user@hostname
```

To download files from the web with the command-line use the following:.

```
$ wget <URL>
```

- or -

```
$ curl -O <URL>
```

Additional networking tools include tracking the path packets take to a network host:

```
$ traceroute <hostname>
```

However, combining the functionality of `traceroute` and `ping` in a single network diagnostic tool is as follows:

```
$ mtr <hostname>
```

A really great way to monitor active network connections of a host in near-realtime is with the following combination of commands:

```
$ watch -n 0.1 sudo lsof -nP -iTCP -sTCP:LISTEN
```

Understanding and effectively using these networking command-line tools is crucial for system administration, network troubleshooting and ensuring reliable network operations on UNIX systems.

System Performance Monitoring

The following package contains several system monitoring tools such as `sar`, `iostat`, `vmstat` and `mpstat`.

```
$ sudo apt install sysstat
```

The command `netstat -a` displays all network connections and listening ports on the system. It prints out the source and destination address, port number and protocol information for each connection.

The command `netstat -rn` displays the kernel routing table. It prints out the destination address, gateway address and interface information for each route.

The command `iostat` displays I/O utilisation statistics for all block devices. It prints the number of read/write operations and the amount of data transferred for each device.

The command `vmstat` displays virtual memory utilisation statistics. It prints out the amount of free and used memory, the amount of memory swapped in and out and the amount of CPU time used for system processes.

The command `free -h` is used to display the amount of free and used memory in the system, including physical and swap, as well as the shared memory and buffers used by the kernel. The `-h` flag is used to display the output in a more human-readable format.

The command `mpstat` displays multiprocessor utilisation statistics. It prints the number of tasks running on each processor and the amount of time spent in user and system mode.

Regular Expressions

Regular expressions (or regex) are special strings used to describe a search pattern. They are used for string manipulation, data validation and text processing. It can be used to match strings or parts of strings. For example, using regex to match all emails in a string or to match all words that begin with a specific letter. Also, regex can be used to search for specific text and make changes to it by finding all instances of a certain word and then replace them with a different word. Regex can also be used to validate user input, such as checking if an email address is valid or to check if a phone number is in the correct format. Regex can be used to parse strings into smaller parts, including splitting a string or to extract data from an XML document. Furthermore, regex can be used to analyse data like counting the number of occurrences of a certain word in a set of text or to find the most frequently used words in a document.

The following details most of the syntactical components of regex:

Symbol	Description
`.`	Replace any character.
`^`	Match at start of string.
`$`	Match at end of string.
`*`	Match zero or more occurrences.
`+`	Match one or more occurrences.
`?`	Match exactly one occurrence.
`\`	Escape sequence for special characters.
`[ ]`	Group character classes.
`( )`	Group regex.
`{n}`	Match proceeding character *n* times.
`{n,m}`	Match proceeding character *n* times up to *m* times.
`{n,}`	Match proceeding character *n* times or more.

The following pattern would match any string that contains the letters `abc`:

```
/abc/
```

To make the pattern more specific, then use special characters and symbols to create a more complex pattern. For example, the following pattern would match any string that contains the letters `abc` in that order with the character `a` at the beginning of the string and the character `c` at the end of the string:

```
/^abc$/
```

The caret `^` symbol indicates the beginning of the string and the dollar sign `$` indicates the end of the string.

A further basic example to consider is:

```
/\d{5}/
```

The `\d` symbol indicates any digit and the `{5}` indicates that there must be exactly 5 digits in the pattern.

A more complex example is that of a common regex that validates an email address, which is as follows:

```
^[a-zA-Z0-9.!#$%&’*+/=?^_`{|}~-]+@[a-zA-Z0-9](?:[a-zA-Z0-9-]{0,61}[a-zA-Z0-9])?(?:\.[a-zA-Z0-9](?:
[a-zA-Z0-9-]{0,61}[a-zA-Z0-9])?)*$
```

This regex checks the email address against a specified pattern, ensuring that it contains all the necessary components and meets certain formatting requirements. It starts with the `^` symbol, which indicates the start of the string. It then contains a series of characters that are allowed in the email address, including letters, numbers and various special characters (from the ``[a-zA-Z0-9.!#$%&’*+/=?^_`{|}~-]`` set). This is followed by an `@` symbol, which is a required part of an email address. After the `@` symbol, there is a series of characters that are allowed in the email address, again including letters, numbers and various special characters (from the `[a-zA-Z0-9]` set). The `?` symbol indicates that the previous set of characters is optional. This is followed by a `.` symbol, which is a required part of an email address. After the `.` symbol, there is a series of characters that are allowed in the email address, again including letters, numbers and various special characters (from the `[a-zA-Z0-9]` set). The `?` symbol again indicates that the previous set of characters is optional. The `*` symbol indicates that the previous set of characters can be repeated any number of times (including zero). Finally, the `$` symbol indicates the end of the string.

Essentially, regular expressions are a powerful tool used to search and manipulate text strings.

Visual Editor

The visual editor `vi` is a text editor developed by Bill Joy (a founder and Chief Scientist of Sun Microsystems) in 1976. It is used primarily on UNIX/UNIX-like systems and is still widely used today. It is a modal editor, meaning that it has different modes for different tasks. Also, `vi` is a powerful editor, although it can be difficult to learn due to its modal nature.

A future development of the original `vi` is its clone `vim` (Vi IMproved), which comes with additional features designed to make it easier to use. It is also a modal editor, like the original `vi`. Furthermore, `vim` supports syntax highlighting, auto-completion and macros. A version of `vim` that runs as a GUI application is `gvim`. It provides the same features as `vim`, but with a graphical interface that allows for the look and feel of the editor to be extensively customised. Also, because it has a GUI it is even more easier to use. Furthermore, `gvim` also supports plugins and script languages. It also facilitates syntax highlighting for a huge variety of different programming languages too.

Worth noting in advance is that there are two modes for `vi`, `vim` and `gvim`, that being *command* and *insert* mode. The *command* mode allows for further functionality and efficient editing capability, although when in this mode it is not possible to type input as expected with a text editor. This is where *insert* mode comes in, therefore making it possible to type input accordingly. Initially, upon starting the visual editor the default mode is that of *command* mode. To change from *command* mode to *insert* mode can be done a variety of ways but the most easiest way is by pressing the `i` key. To switch from *insert* mode to *command* mode can be done by pressing the `ESC` key.

To open a file with all variants of the editor:

```
$ vi <filename>
$ vim <filename>
$ gvim <filename>
```

By specifying the flag option `-x` a prompt for the entry of an encryption key is presented, which encrypts the file using blowfish2 as the encryption algorithm.

As previously stated, the default mode upon opening a file is *command* mode. Upon opening a file, press the `i` key to change to *insert* mode, whereby in this mode all the arrow keys move the cursor around the contents of the file. To save the file, entering *command* mode is required, which is done by pressing the `ESC` key followed by typing `:w` and then pressing the `ENTER` key in order to write the file. To quit editing the file, enter *command* mode and type `:q` followed by the `ENTER` key. Both of these *command* mode commands can be used together for more efficiency, therefore upon pressing the `ESC` key the following command `:wq` will write and quit the file. To abort changes to file when quitting press the ESC key followed by the command `:q!` to override accordingly.

To search for a word in a file, simply press `ESC` followed by `/word-to-search` and then `ENTER`. Use the `n` key for cycling a forward search or the `N` key for cycling a search backwards. Also, searching a word allows for regex too. To replace all occurrences of a word in the file requires entering *command* mode followed by:

```
:%s/old-word/new-word/g
```

Then press `ENTER`.

Command	**Description**
`ESC`	Terminate i*nsert* mode and enter *command* mode.
`i`	Insert at cursor (enters *insert* mode).
`a`	Insert after cursor (enters *insert* mode).
`A`	Insert at end-of-line (enters *insert* mode).
`o`	Open newline below current line (enters *insert* mode).
`O`	Open newline above current line (enters *insert* mode).
`u`	Undo.
`CTRL R`	Redo.
`gg`	Move cursor to the first line.
`G`	Move cursor to the last line.
`x`	Delete character at cursor.
`3x`	Delete 3 characters from the cursor.
`r`	Replace character at cursor.
`2r`	Replace 2 characters from the cursor.
`dd`	Delete current line.
`7dd`	Delete 7 lines from current line.
`D`	Delete contents of the current line from the cursor position.
`C`	Delete contents of the current line from the cursor position (enters *insert* mode).
`dw`	Delete word.
`3dw`	Delete 3 words.
`dG`	Delete the current line to the last line.
`cw`	Change word (enters *insert* mode).
`yy`	Yank a line (copy current line into buffer).
`3yy`	Yank 3 lines (copy 3 lines from current line into buffer).
`p`	Paste contents in buffer (NB: Delete operations copy removed content into buffer).
`~`	Change of case alternating from upper-to-lower or lower-to-upper.

Conclusion

This tutorial has provided an introduction to the UNIX/UNIX-like command-line and fundamental commands from basic navigation and file manipulation to more advanced topics, such as system management. This document aims to be an essential guide for the development of necessary skills to efficiently operate within a UNIX environment. Emphasis was placed on understanding the syntax and functionality of commands, ensuring that users can apply them in real-world scenarios with confidence.
Additionally, special attention was given to the visual editor `vi` and regular expressions, tools that significantly enhance text processing and editing capabilities. As a novice user continuing to explore and master the UNIX commands in this tutorial, this will allow for establishing a foundation to becoming more proficient in managing UNIX systems, with the goal to hopefully encourage further learning through exploration, ensuring better preparation in tackling complex tasks and to optimise workflows accordingly. The core aim is to provide a valuable resource for gaining a high-level of proficiency within UNIX environments through powerful use of the command-line.