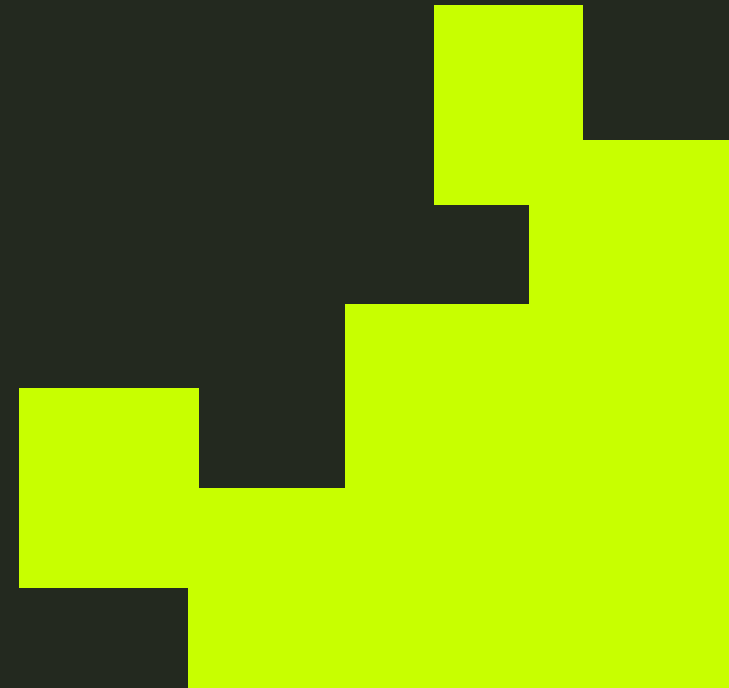


Basic Operations Linux

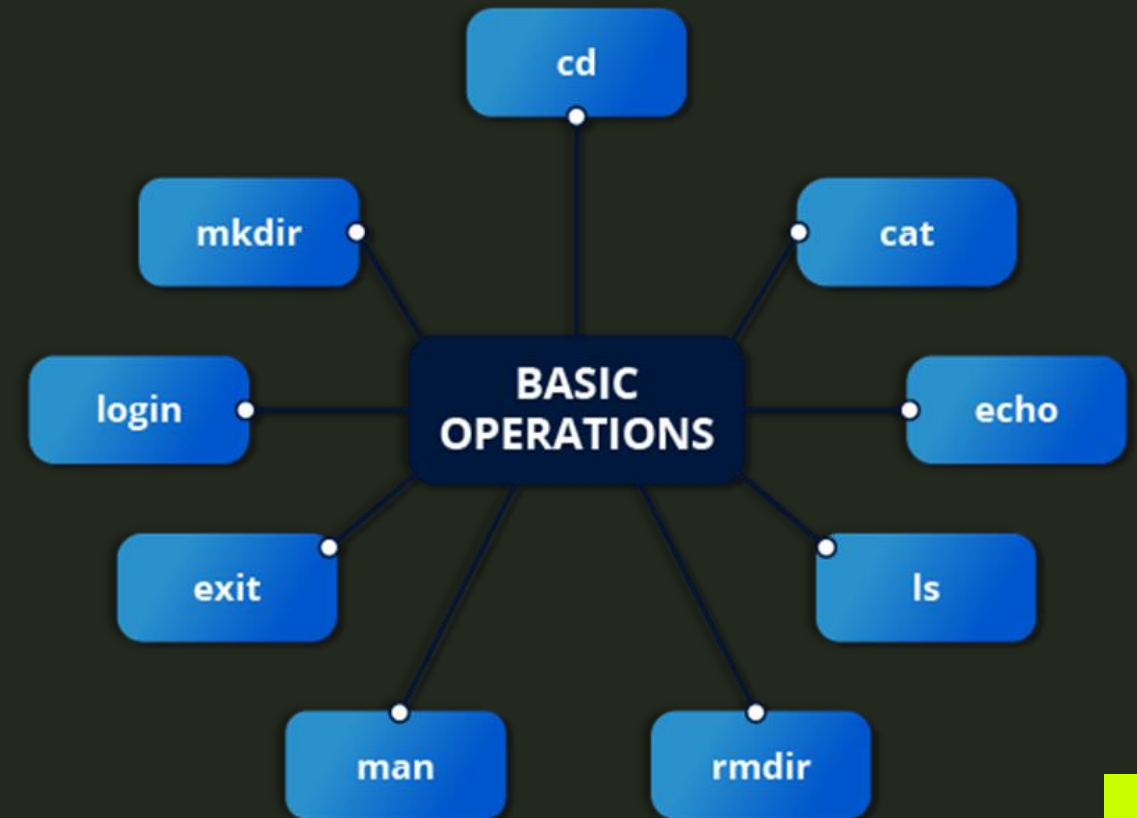
AUGUST 2025



Basic Operations

Basic Operations

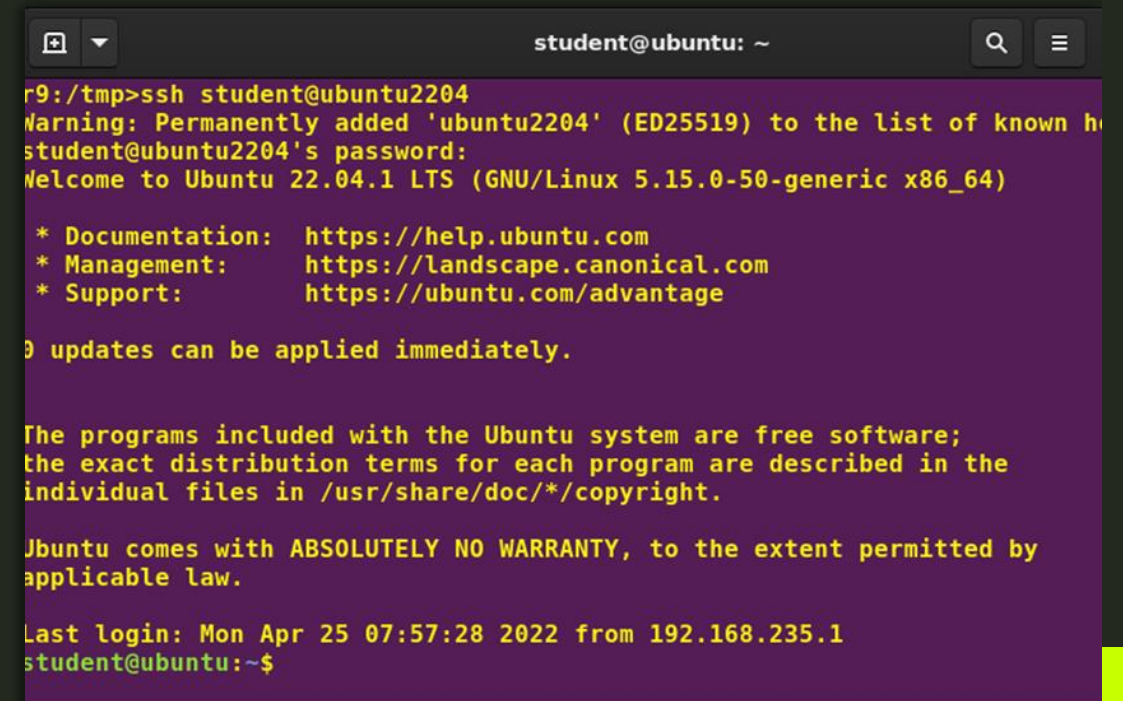
In this section, we will discuss how to accomplish basic operations from the command line. These include how to log in and log out from the system, restart or shut down the system, locate applications, access directories, identify absolute and relative paths, and explore the filesystem.



Logging In and Out

➤ An available text terminal will prompt for a username (with the string login:) and password. When typing your password, nothing is displayed on the terminal (not even a * to indicate that you typed in something), to prevent others from seeing your password. After you have logged into the system, you can perform basic operations.

➤ Once your session is started (either by logging into a text terminal or via a graphical terminal program), you can also connect and log into remote systems by using Secure SHell (SSH). For example, by typing `ssh student@remote-server.com`, SSH would connect securely to the remote machine (remote-server.com) and give student a command line terminal window, using either a password (as with regular logins) or cryptographic key to sign in without providing a password to verify the identity.

A screenshot of a terminal window titled 'student@ubuntu: ~'. The terminal shows the output of an SSH command. It starts with a warning about adding 'ubuntu2204' to the list of known hosts, followed by a password prompt (the password is not visible). The terminal then displays the Ubuntu 22.04.1 LTS welcome message, including documentation, management, and support links. It also shows that 0 updates can be applied immediately. A disclaimer about the Ubuntu system being free software is displayed. The last login information is shown as 'Mon Apr 25 07:57:28 2022 from 192.168.235.1'. The prompt changes to 'student@ubuntu:~\$'.

```
student@ubuntu: ~
r9:/tmp>ssh student@ubuntu2204
Warning: Permanently added 'ubuntu2204' (ED25519) to the list of known hosts
student@ubuntu2204's password:
Welcome to Ubuntu 22.04.1 LTS (GNU/Linux 5.15.0-50-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

0 updates can be applied immediately.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

Last login: Mon Apr 25 07:57:28 2022 from 192.168.235.1
student@ubuntu:~$
```

Rebooting and Shutting Down

↘ GitOps uses Git as the single source of truth for your cluster state. It automates synchronization between Git repositories and Kubernetes clusters. This approach ensures version control and full auditability.

The preferred method to shut down or reboot the system is to use the shutdown command. This sends a warning message, and then prevents further users from logging in. The init process will then control shutting down or rebooting the system. It is important to always shut down properly; failure to do so can result in damage to the system and/or loss of data.

The halt and poweroff commands issue shutdown -h to halt the system; reboot issues shutdown -r and causes the machine to reboot instead of just shutting down. Both rebooting and shutting down from the command line requires superuser (root) access.

When administering a multi-user system, you have the option of notifying all users prior to shutdown, as in:
\$ sudo shutdown -h 10:00
"Shutting down for scheduled maintenance."

```
student@ubuntu: ~  
student@ubuntu:~$ sudo shutdown -h 10:00 "Shutting down for scheduled maintenance"  
Shutdown scheduled for Fri 2017-01-06 10:00:00 CST, use 'shutdown -c' to cancel.  
student@ubuntu:~$
```

Locating Applications

➤ Depending on the specifics of your particular distribution's policy, programs and software packages can be installed in various directories. In general, executable programs and scripts should live in the /bin, /usr/bin, /sbin, /usr/sbin directories, or somewhere under /opt. They can also appear in /usr/local/bin and /usr/local/sbin, or in a directory in a user's account space, such as /home/student/bin.

➤ One way to locate programs is to employ the which utility. For example, to find out exactly where the diff program resides on the filesystem:

```
$ which diff
```

➤ /usr/bin/diff

➤ If which does not find the program, whereis is a good alternative because it looks for packages in a broader range of system directories:

```
$ whereis diff
```

➤ diff: /usr/bin/diff /usr/share/man/man1/diff.1.gz /usr/share/man/man1p/diff.1p.gz

as well as locating source and man files packaged with the program.

```
student@opensuse:~> which firefox
/usr/bin/firefox
student@opensuse:~> whereis firefox
firefox: /usr/bin/firefox /usr/lib64/firefox /usr/share/man/man1/firefox.1.gz
student@opensuse:~>
```

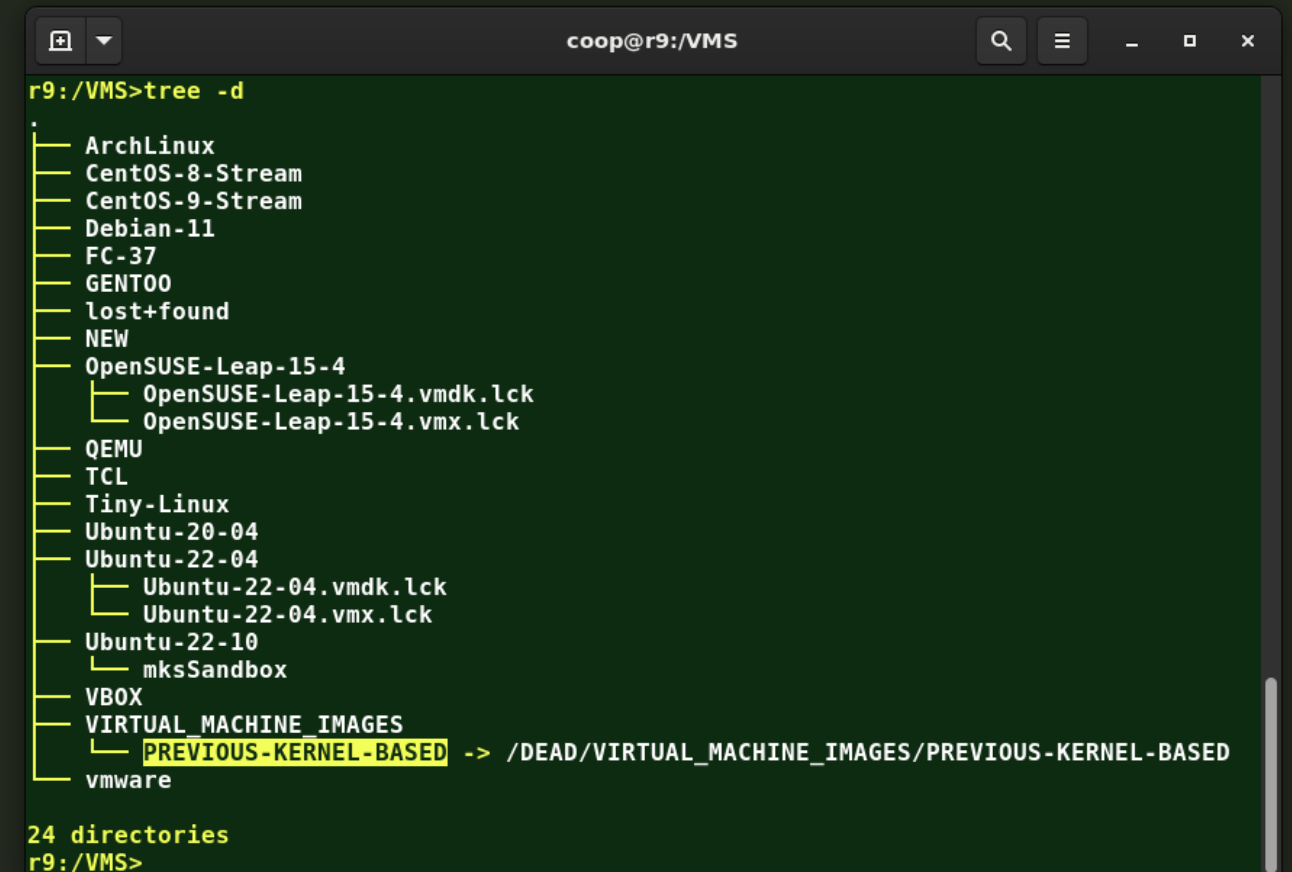
Accessing Directories

↘ When you first log into a system or open a terminal, the default directory should be your home directory. You can see the exact location by typing `echo $HOME`. However, most Linux distributions open new graphical terminals in `$HOME/Desktop` instead.

COMMAND	RESULT
<code>pwd</code>	Displays the present working directory
<code>cd ~</code> or <code>cd</code>	Change to your home directory; shortcut name is <code>~</code> (tilde)
<code>cd ..</code>	Change to parent directory (<code>..</code>)
<code>cd -</code>	Change to previous working directory; <code>-</code> (minus)

Exploring the Filesystem

- ↘ Traversing up and down the filesystem tree can get tedious. The `tree` command is a good way to get a bird's-eye view of the filesystem tree.
- ↘ Use `tree -d` to view just the directories and to suppress listing file names.



```
coop@r9:/VMS
r9:/VMS>tree -d
.
├── ArchLinux
├── CentOS-8-Stream
├── CentOS-9-Stream
├── Debian-11
├── FC-37
├── GENTOO
├── lost+found
├── NEW
├── OpenSUSE-Leap-15-4
│   ├── OpenSUSE-Leap-15-4.vmdk.lck
│   └── OpenSUSE-Leap-15-4.vmx.lck
├── QEMU
├── TCL
├── Tiny-Linux
├── Ubuntu-20-04
├── Ubuntu-22-04
│   ├── Ubuntu-22-04.vmdk.lck
│   └── Ubuntu-22-04.vmx.lck
├── Ubuntu-22-10
│   └── mksSandbox
├── VBox
├── VIRTUAL_MACHINE_IMAGES
│   └── PREVIOUS-KERNEL-BASED -> /DEAD/VIRTUAL_MACHINE_IMAGES/PREVIOUS-KERNEL-BASED
└── vmware

24 directories
r9:/VMS>
```

Hard Links

- The `ln` utility is used to create hard links and (with the `-s` option) soft links, also known as symbolic links or symlinks. These two kinds of links are very useful in UNIX-based operating systems.
- Suppose that `file1` already exists. A hard link, called `file2`, is created with the command:
 - `$ ln file1 file2`
- Note that two files now appear to exist. However, a closer inspection of the file listing shows that this is not quite true.
- `$ ls -li file1 file2`
- The `-i` option to `ls` prints out in the first column the inode number, which is a unique quantity for each file object. This field is the same for both of these files; what is really going on here is that it is only one file, but it has more than one name associated with it, as is indicated by the 2 that appears in the `ls` output. Thus, there was already another object linked to `file1` before the command was executed.
- Hard links are very useful and they save space, but you have to be careful with their use, sometimes in subtle ways. For one thing, if you remove either `file1` or `file2` in the example, the inode object (and the remaining file name) will remain, which might be undesirable, as it may lead to subtle errors later if you recreate a file of that name.
- If you edit one of the files, exactly what happens depends on your editor; most editors, including `vi` and `gedit`, will retain the link by default, but it is possible that modifying one of the names may break the link and result in the creation of two objects.

```
student@ubuntu: /tmp
student@ubuntu:/tmp$ touch file1
student@ubuntu:/tmp$ ln file1 file2
student@ubuntu:/tmp$ ls -li file?
38809 -rw-rw-r-- 2 student student 0 Jan  6 09:18 file1
38809 -rw-rw-r-- 2 student student 0 Jan  6 09:18 file2
student@ubuntu:/tmp$
```


Soft (Symbolic) Links

- Soft (or Symbolic) links are created with the `-s` option, as in:
- `$ ln -s file1 file3`
- `$ ls -li file1 file3`
- Notice file3 no longer appears to be a regular file, and it clearly points to file1 and has a different inode number.
- Symbolic links take no extra space on the filesystem (unless their names are very long). They are extremely convenient, as they can easily be modified to point to different places. An easy way to create a shortcut from your home directory to long pathnames is to create a symbolic link.
- Unlike hard links, soft links can point to objects even on different filesystems, partitions, and/or disks and other media, which may or may not be currently available or even exist. In the case where the link does not point to a currently available or existing object, you obtain a dangling link.

```
student@ubuntu: /tmp
student@ubuntu:/tmp$ touch file1
student@ubuntu:/tmp$ ln file1 file2
student@ubuntu:/tmp$ ls -li file?
38809 -rw-rw-r-- 2 student student 0 Jan  6 09:18 file1
38809 -rw-rw-r-- 2 student student 0 Jan  6 09:18 file2
student@ubuntu:/tmp$ ln -s file1 file3
student@ubuntu:/tmp$ ls -li file?
38809 -rw-rw-r-- 2 student student 0 Jan  6 09:18 file1
38809 -rw-rw-r-- 2 student student 0 Jan  6 09:18 file2
38810 lrwxrwxrwx 1 student student 5 Jan  6 09:22 file3 -> file1
student@ubuntu:/tmp$
```

Navigating Through Directory History

- The `cd` command remembers where you were last, and lets you get back there with `cd -`. For remembering more than just the last directory visited, use `pushd` to change the directory instead of `cd`; this pushes your starting directory onto a list.
- Using `popd` will then send you back to those directories, walking in reverse order (the most recent directory will be the first one retrieved with `popd`). The list of directories is displayed with the `dirs` command.

A terminal window titled 'student@fedora:~' with search, menu, and close buttons. It shows a sequence of commands: 'mkdir /tmp/dir1 /tmp/dir2', 'pushd /tmp/dir1' (outputting '/tmp/dir1 ~'), 'pushd /tmp/dir2' (outputting '/tmp/dir2 /tmp/dir1 ~'), 'popd' (outputting '/tmp/dir1 ~'), 'popd' (outputting '~'), and finally a blank prompt '[student@fedora ~]\$' with a cursor.

```
[student@fedora ~]$ mkdir /tmp/dir1 /tmp/dir2
[student@fedora ~]$ pushd /tmp/dir1
/tmp/dir1 ~
[student@fedora dir1]$ pushd /tmp/dir2
/tmp/dir2 /tmp/dir1 ~
[student@fedora dir2]$ popd
/tmp/dir1 ~
[student@fedora dir1]$ popd
~
[student@fedora ~]$
```

Working with Files

- Linux provides many commands that help you with viewing the contents of a file, creating a new file or an empty file, changing the timestamp of a file, and moving, removing and renaming a file or directory. These commands help you in managing your data and files and in ensuring that the correct data is available at the correct location.
- In this section, you will learn how to manage files.

COMMAND	USAGE
<code>cat</code>	Used for viewing files that are not very long; it does not provide any scroll-back.
<code>tac</code>	Used to look at a file backwards, starting with the last line.
<code>less</code>	Used to view larger files because it is a paging program. It pauses at each screen full of text, provides scroll-back capabilities, and lets you search and navigate within the file. <i>NOTE: Use <code>/</code> to search for a pattern in the forward direction and <code>?</code> for a pattern in the backward direction. An older program named <code>more</code> is still used, but has fewer capabilities: "less is more".</i>
<code>tail</code>	Used to print the last 10 lines of a file by default. You can change the number of lines by doing <code>-n 15</code> or just <code>-15</code> if you wanted to look at the last 15 lines instead of the default.
<code>head</code>	The opposite of <code>tail</code> ; by default, it prints the first 10 lines of a file.

touch

↘ touch is often used to set or update the access, change, and modify times of files. By default, it resets a file's timestamp to match the current time.

↘ However, you can also create an empty file using touch:

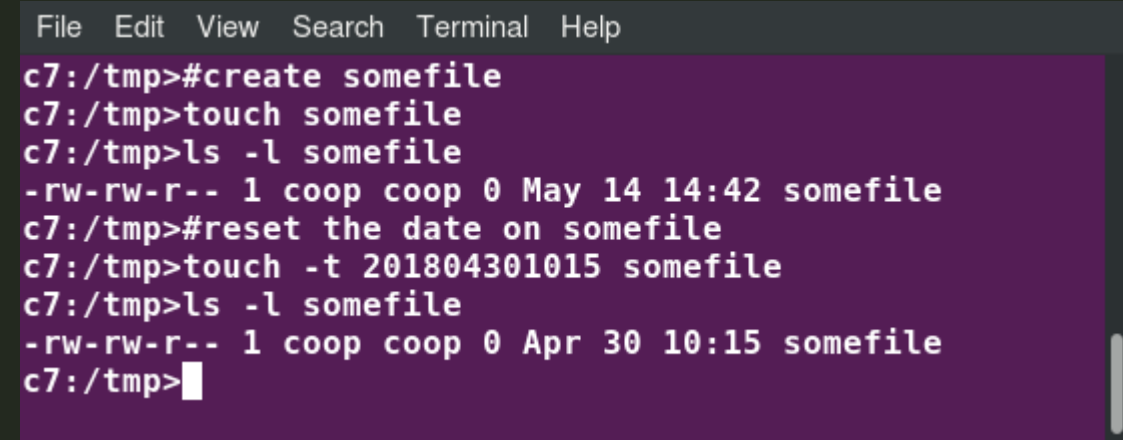
`$ touch <filename>`

↘ This is normally done to create an empty file as a placeholder for a later purpose.

touch provides several useful options. For example, the `-t` option allows you to set the date and timestamp of the file to a specific value, as in:

↘ `$ touch -t 12091600 myfile`

↘ This sets the myfile file's timestamp to 4 p.m., December 9th (12 09 1600).

A terminal window with a dark background and light blue text. The window has a menu bar with 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'. The terminal shows a series of commands and their outputs. A yellow arrow points to the second command.

```
File Edit View Search Terminal Help
c7:/tmp>#create somefile
c7:/tmp>touch somefile
c7:/tmp>ls -l somefile
-rw-rw-r-- 1 coop coop 0 May 14 14:42 somefile
c7:/tmp>#reset the date on somefile
c7:/tmp>touch -t 201804301015 somefile
c7:/tmp>ls -l somefile
-rw-rw-r-- 1 coop coop 0 Apr 30 10:15 somefile
c7:/tmp>
```

mkdir and rmdir

- mkdir is used to create a directory:
- mkdir sampdir
- It creates a sample directory named sampdir under the current directory.
- mkdir /usr/sampdir
- It creates a sample directory called sampdir under /usr.
- Removing a directory is done with rmdir. The directory must be empty or the command will fail. To remove a directory and all of its contents you have to do rm -rf.

```
File Edit View Search Terminal Help
c7:/tmp># Make a directory with a subdirectory
c7:/tmp>mkdir /tmp/somedir/subdir
mkdir: cannot create directory '/tmp/somedir/subdir': No such file or directory
c7:/tmp>
c7:/tmp># now use the -p option and it will work
c7:/tmp>
c7:/tmp>mkdir -p /tmp/somedir/subdir
c7:/tmp>ls -l /tmp/somedir
total 4
drwxrwxr-x 2 coop coop 4096 May 14 14:50 subdir
c7:/tmp>
```

Working with Files

- Moving, Renaming or Removing a File
- Note that mv does double duty, in that it can:
 - Simply rename a file
 - Move a file to another location, while possibly changing its name at the same time.
- If you are not certain about removing files that match a pattern you supply, it is always good to run rm interactively (rm -i) to prompt before every removal.

COMMAND	USAGE
mv	Rename a file
rm	Remove a file
rm -f	Forcefully remove a file
rm -i	Interactively remove a file

Renaming or Removing a Directory

✎ `rmdir` works only on empty directories; otherwise you get an error.

While typing `rm -rf` is a fast and easy way to remove a whole filesystem tree recursively, it is extremely dangerous and should be used with the utmost care, especially when used by root (recall that recursive means drilling down through all sub-directories, all the way down a tree).

COMMAND	USAGE
<code>mv</code>	Rename a directory
<code>rmdir</code>	Remove an empty directory
<code>rm -rf</code>	Forcefully remove a directory recursively

Standard File Streams

- ✚ When commands are executed, by default there are three standard file streams (or descriptors) always open for use: standard input (standard in or **stdin**), standard output (standard out or **stdout**) and standard error (or **stderr**).
- ✚ Usually, **stdin** is your keyboard, and **stdout** and **stderr** are printed on your terminal. **stderr** is often redirected to an error logging file, while **stdin** is supplied by directing input to come from a file or from the output of a previous command through a pipe. **stdout** is also often redirected into a file. Since **stderr** is where error messages (and warning) are written, usually nothing will go there.
- ✚ In Linux, all open files are represented internally by what are called file descriptors. Simply put, these are represented by numbers starting at zero. **stdin** is file descriptor 0, **stdout** is file descriptor 1, and **stderr** is file descriptor 2. Typically, if other files are opened in addition to these three, which are opened by default, they will start at file descriptor 3 and increase from there.
- ✚ On the next page and in the chapters ahead, you will see examples which alter where a running command gets its input, where it writes its output, or where it prints diagnostic (error) messages.

NAME	SYMBOLIC NAME	VALUE	EXAMPLE
standard input	stdin	0	keyboard
standard output	stdout	1	terminal
standard error	stderr	2	log file

I/O Redirection

➤ Through the command shell, we can redirect the three standard file streams so that we can get input from either a file or another command, instead of from our keyboard, and we can write output and errors to files or use them to provide input for subsequent commands.

➤ For example, if we have a program called `do_something` that reads from `stdin` and writes to `stdout` and `stderr`, we can change its input source by using the less-than sign (`<`) followed by the name of the file to be consumed for input data:

➤ `$ do_something < input-file`

➤ If you want to send the output to a file, use the greater-than sign (`>`) as in:

`$ do_something > output-file`

➤ In fact, you can do both at the same time as in:

➤ `$ do_something < input-file > output-file`

➤ Because `stderr` is not the same as `stdout`, error messages will still be seen on the terminal windows in the above example.

➤ If you want to redirect `stderr` to a separate file, you use `stderr`'s file descriptor number (2), the greater-than sign (`>`), followed by the name of the file you want to receive everything the running command writes to `stderr`:

`$ do_something 2> error-file`

➤ NOTE: By the same logic, `do_something 1> output-file` is the same as `do_something > output-file`.

➤ A special shorthand notation can send anything written to file descriptor 2 (`stderr`) to the same place as file descriptor 1 (`stdout`): `2>&1`.

➤ `$ do_something > all-output-file 2>&1`

➤ `bash` permits an easier syntax for the above:

➤ `$ do_something >& all-output-file`

Pipes

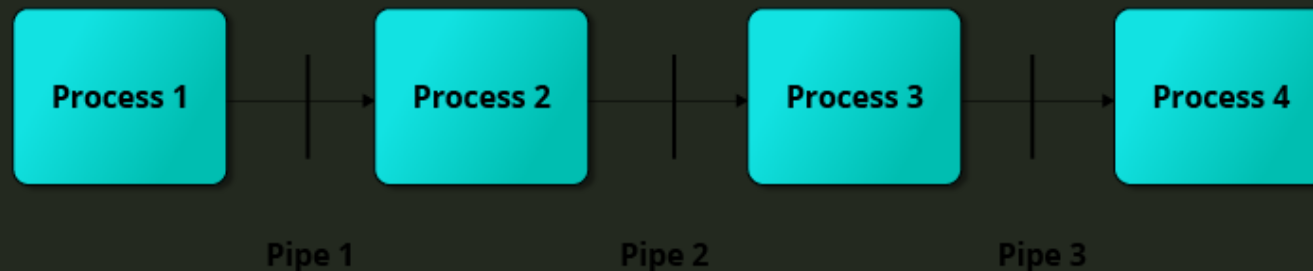
✚ The UNIX/Linux philosophy is to have many simple and short programs (or commands) cooperate together to produce quite complex results, rather than have one complex program with many possible options and modes of operation. In order to accomplish this, extensive use of pipes is made. You can pipe the output of one command or program into another as its input.

In order to do this, we use the vertical-bar, pipe symbol (|), between commands as in:

✚ `$ command1 | command2 | command3`

✚ The above represents what we often call a pipeline, and allows Linux to combine the actions of several commands into one. This is extraordinarily efficient because command2 and command3 do not have to wait for the previous pipeline commands to complete before they can begin processing at the data in their input streams; on multiple CPU or core systems, the available computing power is much better utilized and things get done quicker.

✚ Furthermore, there is no need to save output in (temporary) files between the stages in the pipeline, which saves disk space and reduces reading and writing from disk, which often constitutes the slowest bottleneck in getting something done.



Searching for Files

↳ Being able to quickly find the files you are looking for will save you time and enhance productivity. You can search for files in both your home directory space, or in any other directory or location on the system. The main tools for doing this are the locate and find utilities. We will also show how to use wildcards in bash, in order to specify any file which matches a given generalized request.

locate

➤ The locate utility program performs a search while taking advantage of a previously constructed database of files and directories on your system, matching all entries that contain a specified character string. This can sometimes result in a very long list.

➤ To get a shorter (and possibly more relevant) list, we can use the grep program as a filter. grep will print only the lines that contain one or more specified strings, as in:

➤ `$ locate zip | grep bin`

➤ which will list all the files and directories with both zip and bin in their name. We will cover grep in more detail later. Notice the use of `|` to pipe the two commands together.

➤ locate utilizes a database created by a related utility, updatedb. Most Linux systems run this automatically once a day. However, you can update it at any time by just running updatedb from the command line as the root user.

```
student@ubuntu: ~/Desktop
student@ubuntu:~/Desktop$ locate iproute2
/etc/iproute2
/etc/iproute2/bpf_pinning
/etc/iproute2/ematch_map
/etc/iproute2/group
/etc/iproute2/nl_protos
/etc/iproute2/rt_dsfield
/etc/iproute2/rt_protos
/etc/iproute2/rt_protos.d
/etc/iproute2/rt_realms
/etc/iproute2/rt_scopes
/etc/iproute2/rt_tables
/etc/iproute2/rt_tables.d
/etc/iproute2/rt_protos.d/README
/etc/iproute2/rt_tables.d/README
/usr/include/iproute2
/usr/include/iproute2/bpf_elf.h
/usr/share/doc/iproute2
/usr/share/doc/iproute2/README.Debian
/usr/share/doc/iproute2/changelog.Debian.gz
/usr/share/doc/iproute2/copyright
/usr/share/lintian/overrides/iproute2
/var/lib/dpkg/info/iproute2.conffiles
/var/lib/dpkg/info/iproute2.config
/var/lib/dpkg/info/iproute2.list
/var/lib/dpkg/info/iproute2.md5sums
/var/lib/dpkg/info/iproute2.postinst
/var/lib/dpkg/info/iproute2.postrm
/var/lib/dpkg/info/iproute2.templates
student@ubuntu:~/Desktop$
```

Searching for Files

Wildcards and Matching Filenames

- ✚ You can search for a filename containing specific characters using wildcards.
- ✚ To search for files using the `?` wildcard, replace each unknown character with `?`. For example, if you know only the first two letters are 'ba' of a three-letter filename with an extension of `.out`, type `ls ba?.out`.
- ✚ To search for files using the `*` wildcard, replace the unknown string with `*`. For example, if you remember only that the extension was `.out`, type `ls *.out`.

WILDCARD	RESULT
<code>?</code>	Matches any single character
<code>*</code>	Matches any string of characters
<code>[set]</code>	Matches any character in the set of characters, for example <code>[adf]</code> will match any occurrence of a , d , or f
<code>[!set]</code>	Matches any character not in the set of characters

The find Program

- Find is an extremely useful and often-used utility program in the daily life of a Linux system administrator. It recurses down the filesystem tree from any particular directory (or set of directories) and locates files that match specified conditions. The default pathname is always the present working directory.
- For example, administrators sometimes scan for potentially large core files (which contain diagnostic information after a program fails) that are more than several weeks old in order to remove them.
- It is also common to remove files non-essential or outdated files in /tmp (and other volatile directories, such as those under /var/cache/ containing dispensable cached files) that have not been accessed recently. Many Linux distributions use shell scripts that run periodically (through cron usually) to perform such house cleaning.

```
student@fedora:~/var/log
[student@fedora log]$ sudo find /var/log -name "*.log"
/var/log
/var/log/vmware-network.log
/var/log/dnf.log
/var/log/dnf.rpm.log
/var/log/anaconda/journal.log
/var/log/anaconda/dbus.log
/var/log/anaconda/dnf.librepo.log
/var/log/anaconda/storage.log
/var/log/anaconda/anaconda.log
/var/log/anaconda/ks-script-6wy_62yg.log
/var/log/anaconda/program.log
/var/log/anaconda/hawkey.log
/var/log/anaconda/packaging.log
/var/log/anaconda/ks-script-bgfs3yyj.log
/var/log/anaconda/ks-script-ktr0f9lq.log
/var/log/sss/sss_kcm.log
/var/log/lastlog
/var/log/audit/audit.log
/var/log/dnf.librepo.log
/var/log/vmware-vmsvc-root.log
/var/log/boot.log
/var/log/tallylog
/var/log/hawkey.log
/var/log/vmware-vmtoolsd-root.log
[student@fedora log]$
```

Using find

↘ When no arguments are given, find lists all files in the current directory and all of its subdirectories. Commonly used options to shorten the list include `-name` (only list files with a certain pattern in their name), `-iname` (also ignore the case of file names), and `-type` (which will restrict the results to files of a certain specified type, such as `d` for directory, `l` for symbolic link, or `f` for a regular file, etc.).

↘ Searching for files and directories named gcc:

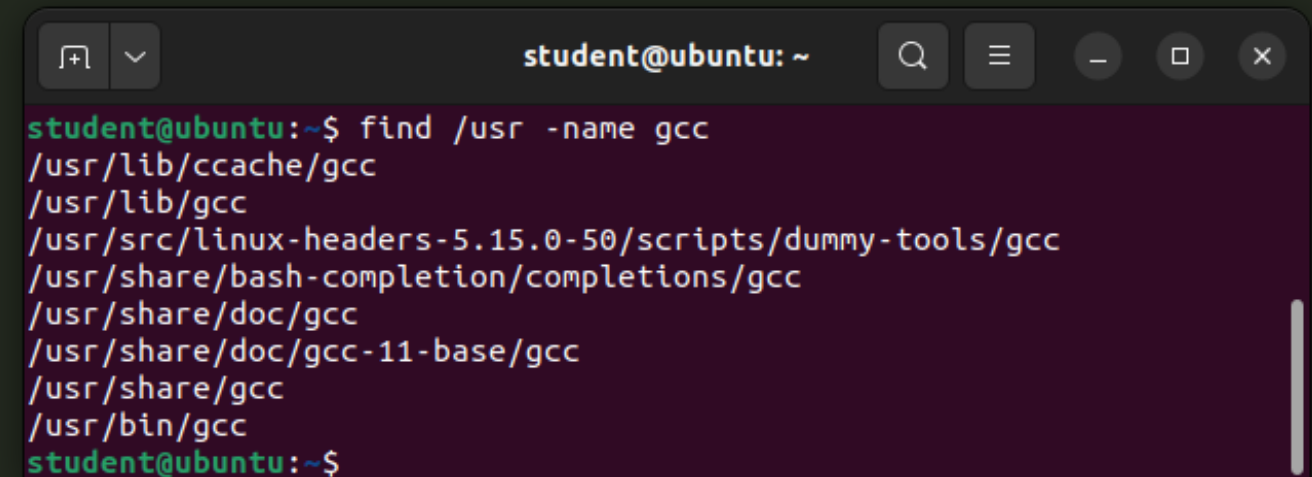
↘ `$ find /usr -name gcc`

↘ Searching only for directories named gcc:

`$ find /usr -type d -name gcc`

↘ Searching only for regular files named gcc:

`$ find /usr -type f -name gcc`

A terminal window titled 'student@ubuntu: ~' with standard window controls. It shows the command 'find /usr -name gcc' being executed. The output lists several paths: /usr/lib/ccache/gcc, /usr/lib/gcc, /usr/src/linux-headers-5.15.0-50/scripts/dummy-tools/gcc, /usr/share/bash-completion/completions/gcc, /usr/share/doc/gcc, /usr/share/doc/gcc-11-base/gcc, /usr/share/gcc, and /usr/bin/gcc. The prompt returns to 'student@ubuntu:~\$' at the bottom.

```
student@ubuntu:~$ find /usr -name gcc
/usr/lib/ccache/gcc
/usr/lib/gcc
/usr/src/linux-headers-5.15.0-50/scripts/dummy-tools/gcc
/usr/share/bash-completion/completions/gcc
/usr/share/doc/gcc
/usr/share/doc/gcc-11-base/gcc
/usr/share/gcc
/usr/bin/gcc
student@ubuntu:~$
```

Using Advanced find Options

- Another good use of find is being able to run commands on the files that match your search criteria. The `-exec` option is used for this purpose.
- To find and remove all files that end with `.swp`:
- `$ find -name "*.swp" -exec rm {} ';'`
- The `{}` (squiggly brackets) is a placeholder that will be filled with all the file names that result from the find expression, and the preceding command will be run on each one individually.
- Please note that you have to end the command with either `';'` (including the single-quotes) or `\;`. Both forms are fine.
- One can also use the `-ok` option, which behaves the same as `-exec`, except that find will prompt you for permission before executing the command. This makes it a good way to test your results before blindly executing any potentially dangerous commands.



Using Advanced find Options

- Another good use of find is being able to run commands on the files that match your search criteria. The `-exec` option is used for this purpose.
- To find and remove all files that end with `.swp`:
- `$ find -name "*.swp" -exec rm {} ';'`
- The `{}` (squiggly brackets) is a placeholder that will be filled with all the file names that result from the find expression, and the preceding command will be run on each one individually.
- Please note that you have to end the command with either `';'` (including the single-quotes) or `\;`. Both forms are fine.
- One can also use the `-ok` option, which behaves the same as `-exec`, except that find will prompt you for permission before executing the command. This makes it a good way to test your results before blindly executing any potentially dangerous commands.



Finding Files Based on Time and Size

➤ It is sometimes the case that you wish to find files according to attributes, such as when they were created, last used, etc., or based on their size. It is easy to perform such searches.

➤ To find files based on time:

`$ find / -ctime 3`

➤ Here, `-ctime` is when the inode metadata (i.e. file ownership, permissions, etc.) last changed; it is often, but not necessarily, when the file was first created. You can also search for accessed/last read (`-atime`) or modified/last written (`-mtime`) times. The number is the number of days and can be expressed as either a number (n) that means exactly that value, `+n`, which means greater than that number, or `-n`, which means less than that number. There are similar options for times in minutes (as in `-cmin`, `-amin`, and `-mmin`).

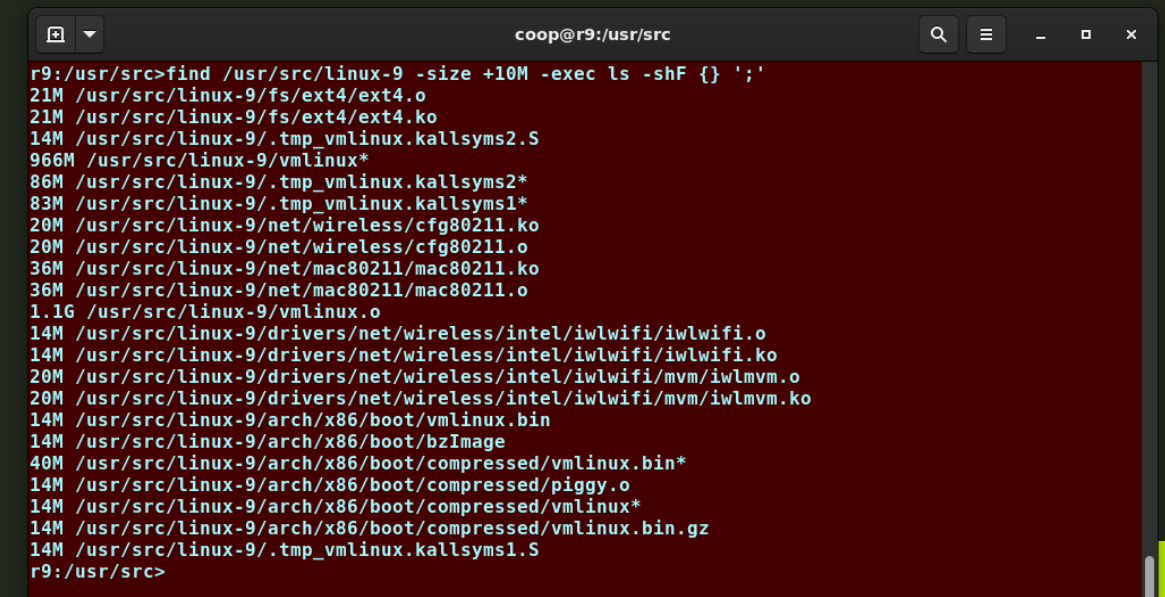
➤ To find files based on sizes:

➤ `$ find / -size 0`

➤ Note the size here is in 512-byte blocks, by default; you can also specify bytes (c), kilobytes (k), megabytes (M), gigabytes (G), etc. As with the time numbers above, file sizes can also be exact numbers (n), `+n` or `-n`. For details, consult the man page for `find`.

➤ For example, to find files greater than 10 MB in size and running a command on those files:

`$ find / -size +10M -exec command {} \;`

A terminal window titled 'coop@r9:/usr/src' showing the output of the command 'find /usr/src/linux-9 -size +10M -exec ls -shF {} ';' . The output lists various files and their sizes, including kernel modules, system binaries, and compressed boot images. The window has a dark background and standard terminal window controls (search, menu, zoom, close).

```
coop@r9:/usr/src
r9:/usr/src>find /usr/src/linux-9 -size +10M -exec ls -shF {} ';'
21M /usr/src/linux-9/fs/ext4/ext4.o
21M /usr/src/linux-9/fs/ext4/ext4.ko
14M /usr/src/linux-9/.tmp_vmlinux.kallsyms2.S
966M /usr/src/linux-9/vmlinux*
86M /usr/src/linux-9/.tmp_vmlinux.kallsyms2*
83M /usr/src/linux-9/.tmp_vmlinux.kallsyms1*
20M /usr/src/linux-9/net/wireless/cfg80211.ko
20M /usr/src/linux-9/net/wireless/cfg80211.o
36M /usr/src/linux-9/net/mac80211/mac80211.ko
36M /usr/src/linux-9/net/mac80211/mac80211.o
1.1G /usr/src/linux-9/vmlinux.o
14M /usr/src/linux-9/drivers/net/wireless/intel/iwlwifi/iwlwifi.o
14M /usr/src/linux-9/drivers/net/wireless/intel/iwlwifi/iwlwifi.ko
20M /usr/src/linux-9/drivers/net/wireless/intel/iwlwifi/mvm/iwlmvm.o
20M /usr/src/linux-9/drivers/net/wireless/intel/iwlwifi/mvm/iwlmvm.ko
14M /usr/src/linux-9/arch/x86/boot/vmlinux.bin
14M /usr/src/linux-9/arch/x86/boot/bzImage
40M /usr/src/linux-9/arch/x86/boot/compressed/vmlinux.bin*
14M /usr/src/linux-9/arch/x86/boot/compressed/piggy.o
14M /usr/src/linux-9/arch/x86/boot/compressed/vmlinux*
14M /usr/src/linux-9/arch/x86/boot/compressed/vmlinux.bin.gz
14M /usr/src/linux-9/.tmp_vmlinux.kallsyms1.S
r9:/usr/src>
```

Summary

- Virtual terminals (VT) in Linux are consoles, or command line terminals that use the connected monitor and keyboard.
- Different Linux distributions start and stop the graphical desktop in different ways.
- A terminal emulator program on the graphical desktop works by emulating a terminal within a window on the desktop.
- The Linux system allows you to either log in via text terminal or remotely via the console.
- When typing your password, nothing is printed to the terminal, not even a generic symbol to indicate that you typed.
- The preferred method to shut down or reboot the system is to use the shutdown command.
- There are two types of pathnames: absolute and relative.
- An absolute pathname begins with the root directory and follows the tree, branch by branch, until it reaches the desired directory or file.
- A relative pathname starts from the present working directory.
- Using hard and soft (symbolic) links is extremely useful in Linux.
- cd remembers where you were last, and lets you get back there with cd -.
- locate performs a database search to find all file names that match a given pattern.
- find locates files recursively from a given directory or set of directories.
- find is able to run commands on the files that it lists, when used with the -exec option.
- touch is used to set the access, change, and edit times of files, as well as to create empty files.
- The Advanced Packaging Tool (apt) package management system is used to manage installed software on Debian-based systems.
- You can use the dnf command-line package management utility for the RPM-based Red Hat Family Linux distributions.
- The zypper package management system is based on RPM and used for openSUSE.

AI STUDIO

info@[aistudio.com.tr](mailto:info@aistudio.com.tr)

aistudio.com.tr

Thank You!

