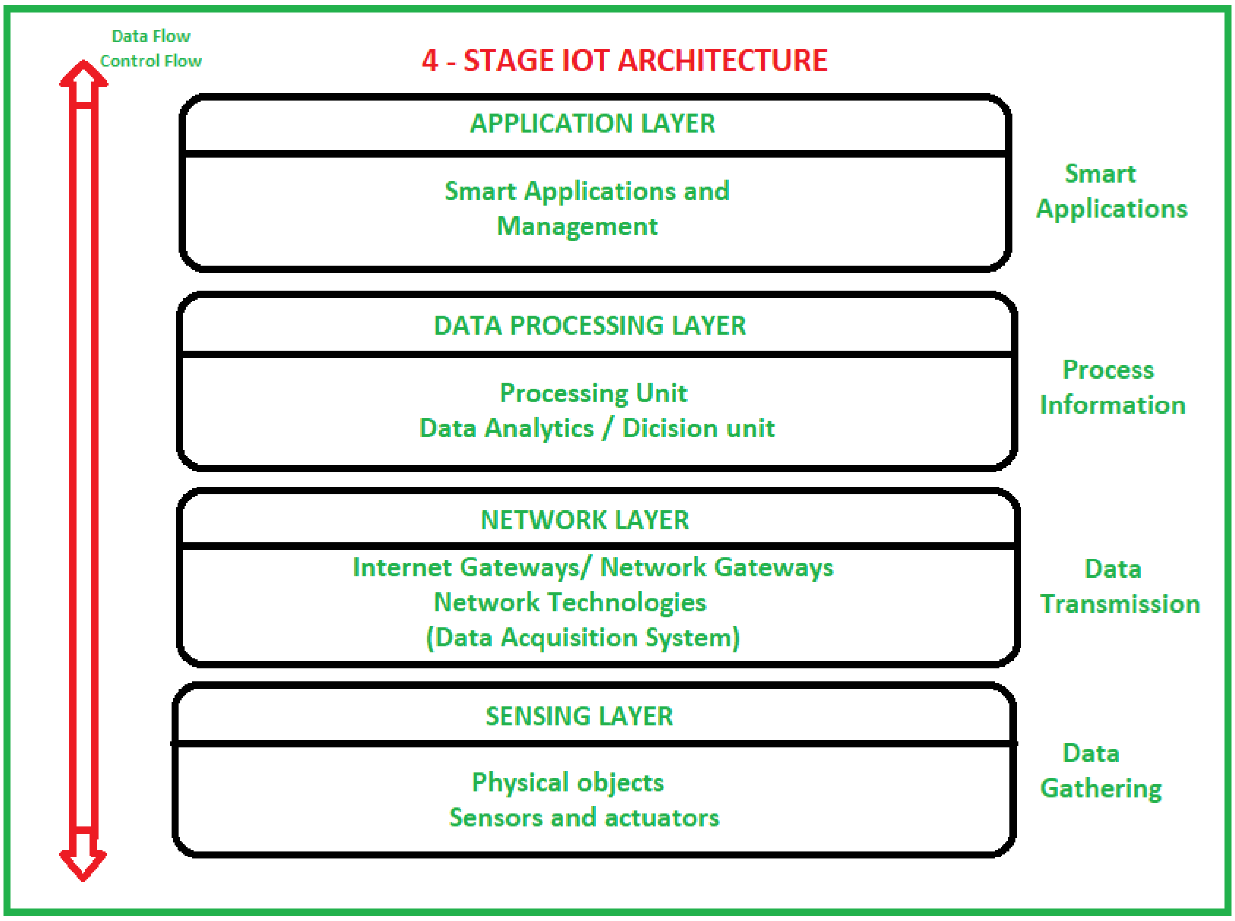
Practical: 1

AIM: Sketch the architecture of IOT and explain each entity in brief.

Solution-

IoT is network of interconnected computing devices which are embedded in everyday objects, enabling them to send and receive data.

Internet of Things (IoT) technology has a wide variety of applications and use of Internet of Things is growing so faster. Depending upon different application areas of Internet of Things, it works accordingly as per it has been designed/developed. But it has not a standard defined architecture of working which is strictly followed universally. The architecture of IoT depends upon its functionality and implementation in different sectors. Still, there is a basic process flow based on which IoT is built.



There is 4 layers are present that can be divided as follows: Sensing Layer, Network Layer, Data processing Layer, and Application Layer.

I. Sensing Layer:

• The sensing layer comprises various types of sensors and actuators that are deployed in the physical environment to collect data. These devices can be temperature sensors, humidity sensors, light sensors, motion sensors, and more.

• Sensors in this layer convert physical parameters into electrical signals or digital data that can be processed and transmitted.

• Actuators, on the other hand, enable the IoT system to interact with the physical environment by controlling devices or triggering actions based on the collected data.

• The sensing layer can employ different communication protocols such as wired (Ethernet, RS-485) or wireless (Bluetooth, Zigbee, LoRaWAN) to connect sensors and actuators to the network layer.

II. Network Layer:

• The network layer facilitates communication and connectivity among IoT devices, enabling them to exchange data with each other and with the wider internet.

• It encompasses various network technologies and protocols to establish reliable and secure connections.

• Wireless technologies like WiFi, Bluetooth, Zigbee, and cellular networks (4G, 5G) are commonly used for IoT device communication.

• Gateways and routers play a crucial role in the network layer, acting as intermediaries between devices and the internet. They facilitate data routing, protocol translation, and provide security functionalities.

• Security measures, including encryption, authentication, and access control, are implemented at the network layer to protect data and devices from unauthorized access.

III. Data Processing Layer:

• The data processing layer handles the collected raw data from the sensing layer and performs necessary processing and analysis.

• It involves data management systems that store and organize the data for further processing.

• Analytics platforms and machine learning algorithms are applied to extract meaningful insights from the collected data.

• Advanced techniques such as data aggregation, filtering, and data fusion may be employed to enhance the accuracy and efficiency of data processing.

• The data processing layer can also include edge computing capabilities, where some processing tasks are performed closer to the data source, reducing latency and network bandwidth usage.

IV. Application Layer:

• The application layer is the interface through which end-users interact with the IoT system and its devices.

• It provides user-friendly interfaces, such as mobile apps, web portals, or desktop applications, for users to monitor and control IoT devices.

• Middleware services play a role in enabling seamless communication and integration between different devices and systems.

• Advanced analytics and processing capabilities within the application layer allow for data visualization, predictive analytics, and automation.

• This layer enables end-users to make informed decisions, take actions,

Practical: 2

AIM: Explain the Working Of Arduino

Solution-

What is Arduino?

Arduino is an open-source electronics platform that consists of both hardware and software components. It is designed to provide an easy and accessible way for people, including beginners and hobbyists, to create and prototype their own electronic projects.

It's Working -

• Arduino Board: The heart of the Arduino platform is the Arduino board, which is a programmable microcontroller. The most commonly used board is the Arduino Uno, but there are several other variations available with different features and capabilities.

• Programming Language: Arduino uses its own programming language, which is a simplified version of C++ and is similar to Wiring. The Arduino software, known as the Integrated Development Environment (IDE), provides a user-friendly interface for writing and uploading code to the Arduino board.

• Writing and Uploading Code: To create a program for Arduino, you write code in the Arduino IDE using its syntax. The code consists of two main parts: the setup() function, which runs once when the board powers up or resets, and the loop() function, which runs repeatedly. In these functions, you can define variables, perform calculations, control digital and analog pins, and interact with sensors, actuators, and other electronic components.

• Connecting Components: Arduino boards have a set of digital and analog input/output (I/O) pins that allow you to connect various electronic components such as sensors, buttons, LEDs, motors, and more. You use jumper wires or shields (additional boards that extend the functionality) to connect these components to the Arduino board.

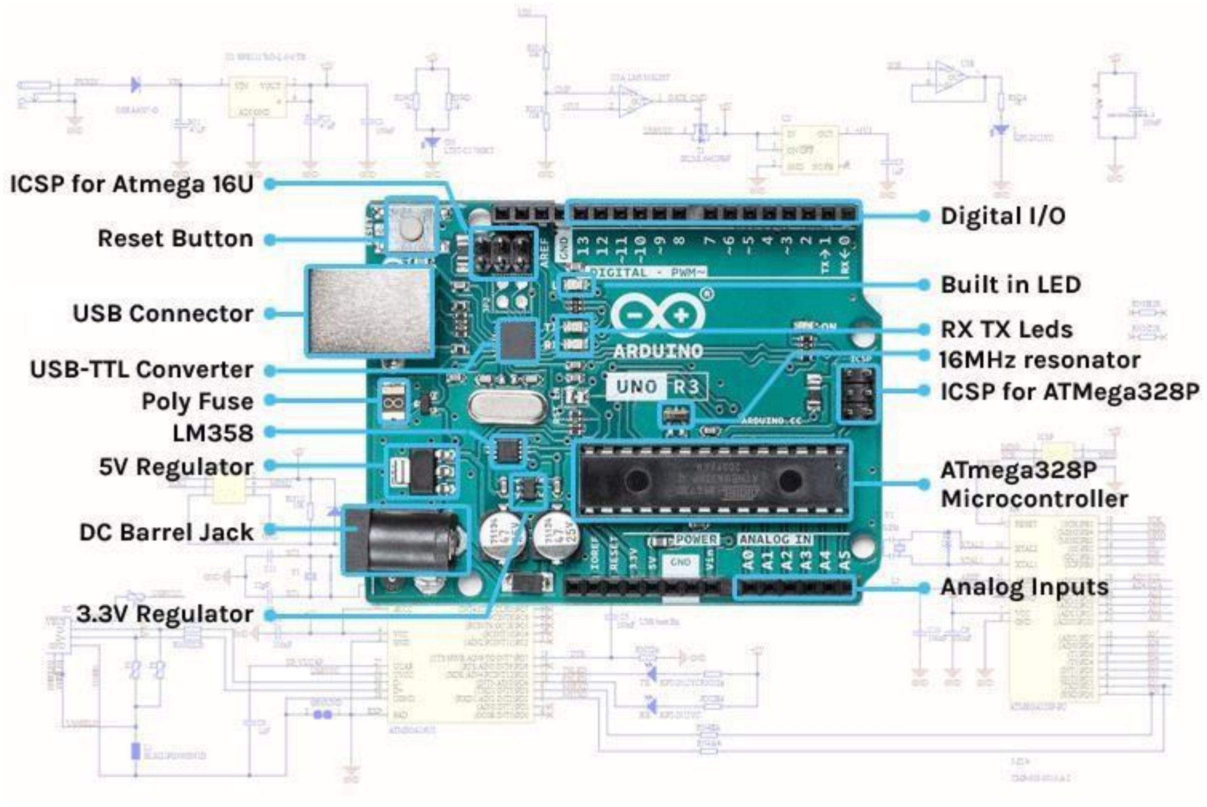
• Powering the Arduino: Arduino boards can be powered either through a USB connection or an external power supply. The power supply provides the necessary voltage and current to operate the board and connected components.

• Uploading Code: Once you have written your code in the Arduino IDE, you can upload it to the Arduino board via a USB connection. The IDE compiles the code, converts it into machine-readable instructions, and sends it to the microcontroller on the Arduino board.

• Execution: After the code is uploaded, the Arduino board executes the instructions in the setup() function once, and then continuously runs the instructions in the loop() function until it is powered off or reset.

By following this process, you can control and interact with electronic components using Arduino, making it a versatile platform for a wide range of projects, from simple blinking LED lights to more complex robotics and automation systems.

Image



Advantages of Arduino :

• User-friendly platform, suitable for beginners and hobbyists.

• Open-source nature allows customization and modification.

• Active community support with a wealth of online resources.

• Compatibility with a wide range of sensors and components.

• Cross-platform support (Windows, macOS, Linux).

• Cost-effective compared to other microcontroller platforms.

Disadvantages of Arduino:

• Limited processing power and memory compared to more advanced microcontrollers.

• Not ideal for resource-intensive applications or real-time tasks.

• Lack of built-in networking capabilities, requiring additional modules for internet connectivity.

• Relatively slower execution speed due to the interpreted nature of the Arduino programming language.

• Limited number of input/output pins on some Arduino boards, restricting the number of connected components.

• May require additional circuitry or shields for complex projects or specific functionalities.

Practical: 3

AIM: Explain the working Of Raspberry Pi.

Solution-

What is Raspberry Pi?

•Raspberry Pi is a series of single-board computers (SBCs) designed to provide an affordable and accessible computing platform. It was created by the Raspberry Pi Foundation, a non-profit organization based in the United Kingdom, with the aim of promoting computer science education and enabling people to learn, experiment, and create with technology.

•The Raspberry Pi boards are small in size, typically the size of a credit card, and consist of all the essential components of a basic computer integrated onto a single circuit board. These components include a microprocessor (CPU), memory (RAM), storage (usually in the form of a microSD card), various ports for connectivity (USB, HDMI, Ethernet), and GPIO (General Purpose Input/Output) pins for interfacing with external devices and sensors.

•Raspberry Pi boards run on various Linux-based operating systems, with the most popular being Raspbian, a version of the Debian operating system optimized for the Raspberry Pi. These operating systems provide a user-friendly interface and support for a wide range of applications and programming languages.

•Due to its affordability, compact size, and versatility, Raspberry Pi has found applications in various fields such as education, robotics, home automation, Internet of Things (IoT), media centers, retro gaming, and much more. It has become a popular tool for learning programming, electronics, and computer science, as well as a platform for prototyping and creating innovative projects.

Working

The Raspberry Pi operates in a similar manner to a traditional computer, albeit with a more compact form factor. Here's a breakdown of the working of Raspberry Pi:

I. Power Supply: The Raspberry Pi requires a power source, typically a micro USB cable connected to a power adapter or a USB port with sufficient power output. Once powered on, the board initializes its components.

II. Booting Process: The Raspberry Pi's operating system is stored on a microSD card inserted into the board's dedicated slot. When the power is supplied, the board reads the bootloader from the microSD card, which then starts the booting process.

III. Operating System: The Raspberry Pi supports various operating systems, with Raspbian (based on Debian) being the most commonly used. The selected operating system is loaded into the memory (RAM) from the microSD card.

IV. User Interface: Once the operating system is loaded, the user interacts with the Raspberry Pi through a display connected via the HDMI port, or remotely through SSH (Secure Shell) for headless operation. The graphical user interface (GUI) allows users to navigate and run applications.

V. Software Applications: Raspberry Pi supports a wide range of software applications. Users can browse the web, write programs, play media, create documents, or use specialized software for specific purposes. The CPU processes instructions and executes these applications.

VI. GPIO Interfacing: One of the key features of Raspberry Pi is its GPIO pins. These pins allow users to connect and interact with external hardware components, such as sensors, LEDs, motors, and more. Users can write code in programming languages like Python to control and receive input from these external devices.

VII. Connectivity: Raspberry Pi boards come with various ports, including USB ports, Ethernet ports, and Wi-Fi capabilities. These connectivity options enable the device to connect to networks, peripherals, and the internet, facilitating data transfer and communication.

VIII. Storage and File Management: The Raspberry Pi uses a microSD card as its primary storage medium. Users can install and store applications, data, and files on the microSD card. Additionally, external storage devices, such as USB drives or network-attached storage (NAS), can be connected for expanded storage options.

IX. Power-Off: When you want to power off the Raspberry Pi, it is recommended to properly shut down the operating system. This can be done through the graphical interface or by using command-line instructions. After the shutdown process, the power supply can be disconnected.

Advantages

- Affordability compared to traditional computers.
- Compact size and portability.
- Low power consumption.
- Flexibility and customizability through GPIO pins.
- Active community and extensive resources for support and learning.

Disadvantages

- Limited processing power compared to high-end computers.
- Limited RAM capacity.
- Storage limitations with microSD cards.
- Lack of native x86 architecture compatibility.
- Some applications may not be directly compatible.

Applications

- Prototyping with Raspberry Pi. The old-school methodology for prototyping a new device involved an electrical engineer and building an ugly green PCB to test mechanical or software functionality.
- Low-volume/higher-cost production. Raspberry Pi is a great hardware option for low-volume (up to thousands), higher-priced products.
- Raspberry Pi test bench.
- Cloud-to-Bluetooth server.
- Side-car server

Practical: 4

AIM: Connect Raspberry Pi with your existing system components.

Solution-

What is Raspberry Pi?

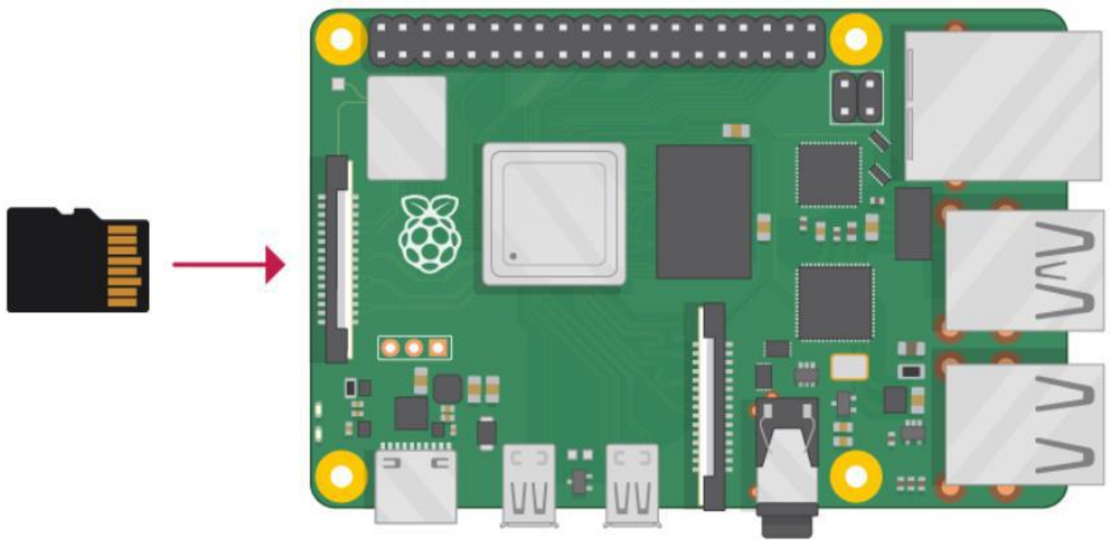
• Raspberry Pi is a series of small, single-board computers developed by the Raspberry Pi Foundation. It was designed to provide an affordable and accessible platform for learning programming and computer science, as well as for creating various projects and applications.

• It is a series of small, affordable, and versatile single-board computers designed for learning, tinkering, and creating various projects. It provides a complete computer system on a credit card-sized board and supports a wide range of applications, from programming and robotics to home automation and IoT devices. It has gained popularity for its accessibility, GPIO pins for connecting electronic components, and its role in promoting computer education.

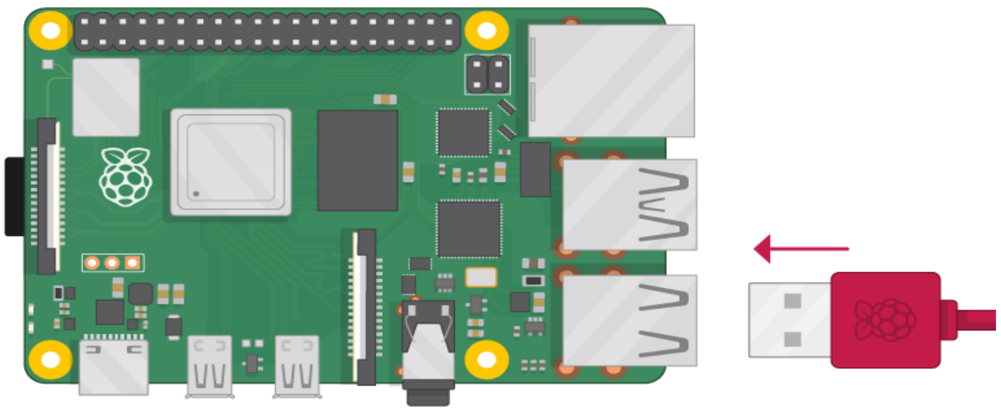
Connect your Raspberry Pi

Let's connect up your Raspberry Pi and get it running.

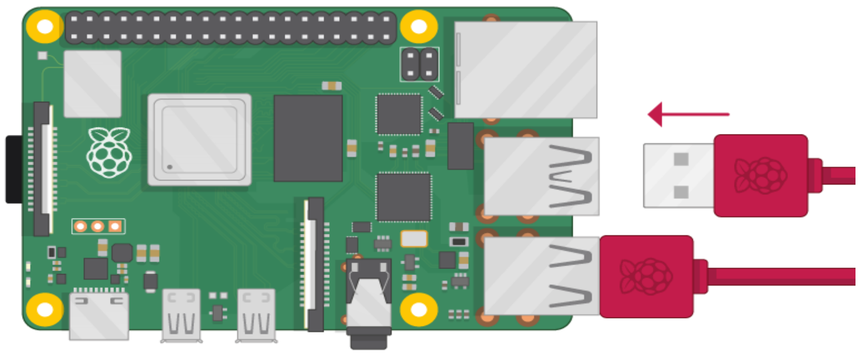
o Check the slot on the underside of your Raspberry Pi to see whether an SD card is inside. If no SD card is there, then insert an SD card with Raspbian installed (via NOOBS).



o Find the USB connector end of your mouse’s cable, and connect the mouse to a USB port on your Raspberry Pi (it doesn’t matter which port you use).



o Connect the keyboard in the same way.

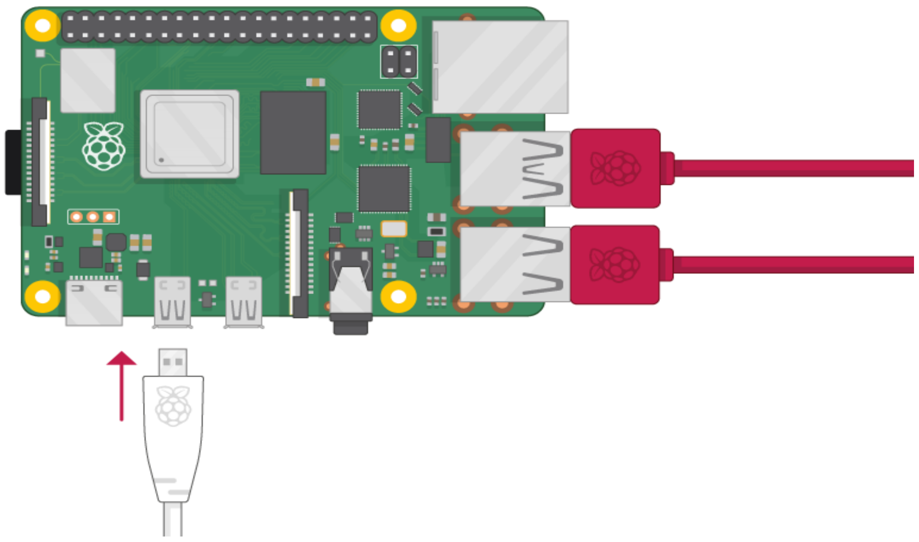


o Look at the HDMI port(s) on your Raspberry Pi — notice that they have a flat side on top.

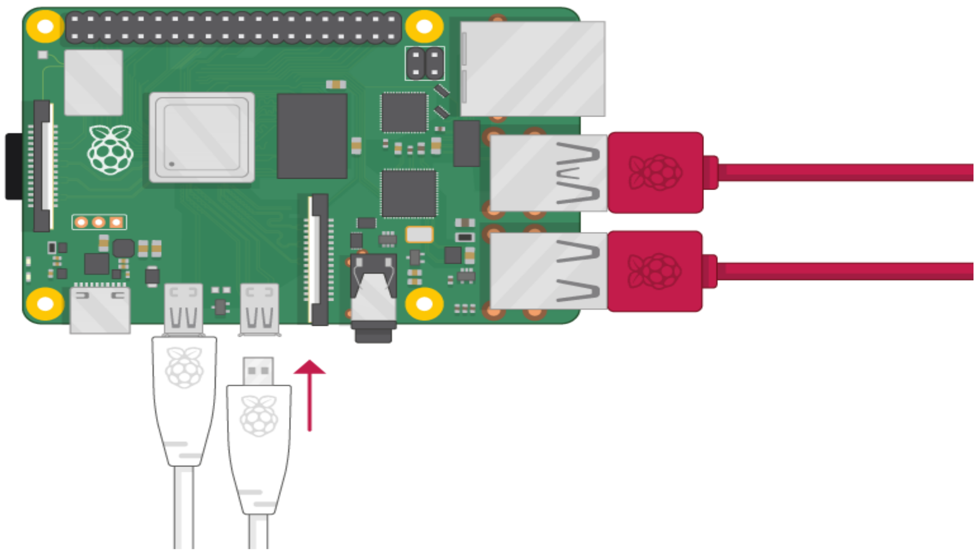
o Use a cable to connect the screen to the Raspberry Pi's HDMI port — use an adapter if necessary.

Raspberry Pi 4

o Connect your screen to the first of Raspberry Pi 4's HDMI ports, labelled HDMI0.

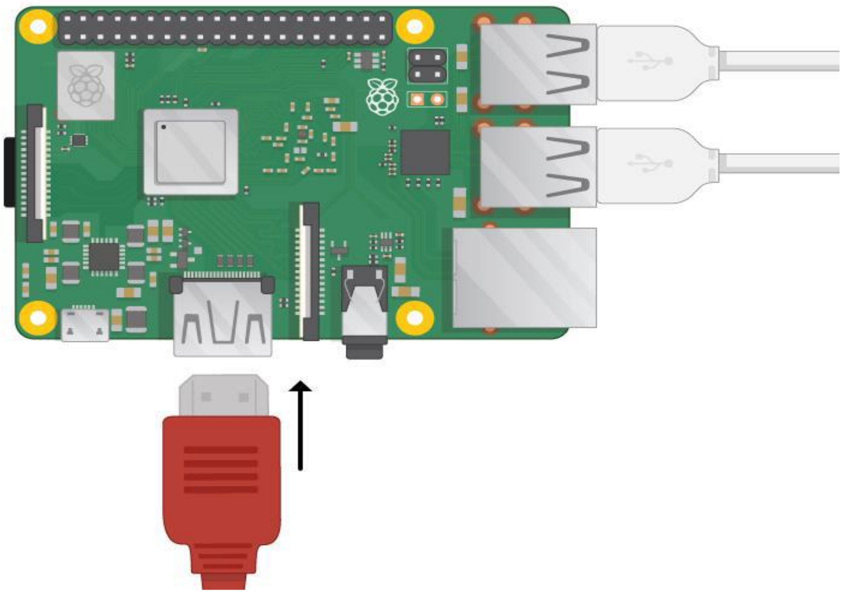


o You could connect an optional second screen in the same way.

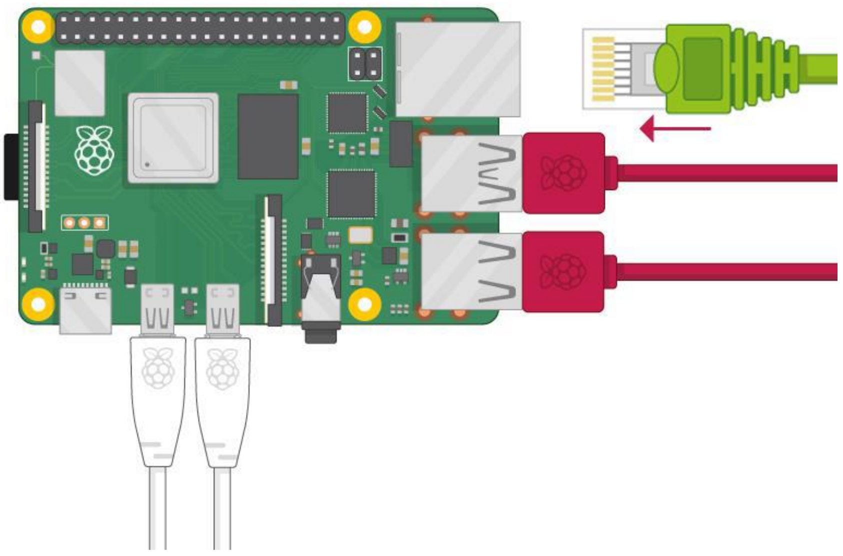


Raspberry Pi 1, 2, 3

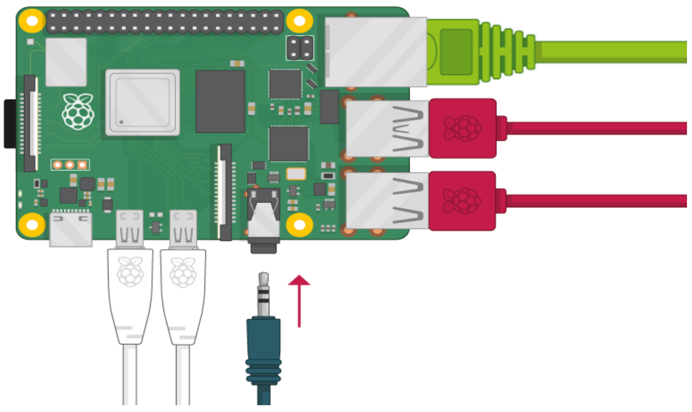
o Connect your screen to the single HDMI port.



o If you want to connect the Pi to the internet via Ethernet, use an Ethernet cable to connect the Ethernet port on the Raspberry Pi to an Ethernet socket on the wall or on your internet router. You don't need to do this if you want to use wireless connectivity, or if you don't want to connect to the internet.

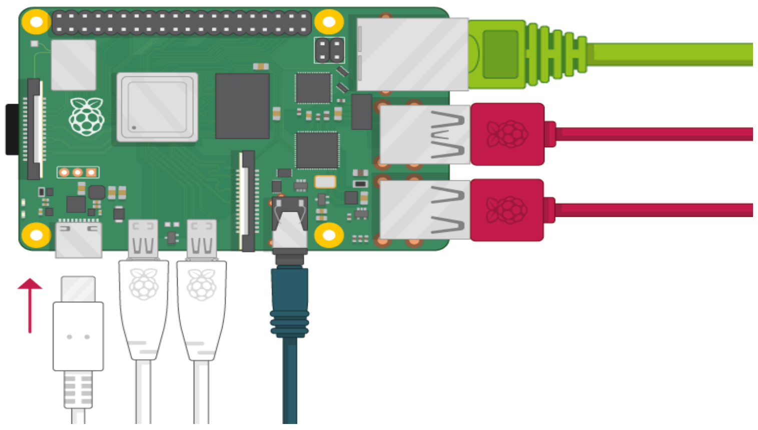


o If your screen has speakers, your Raspberry Pi can play sound through these. Or you could connect headphones or speakers to the audio port.

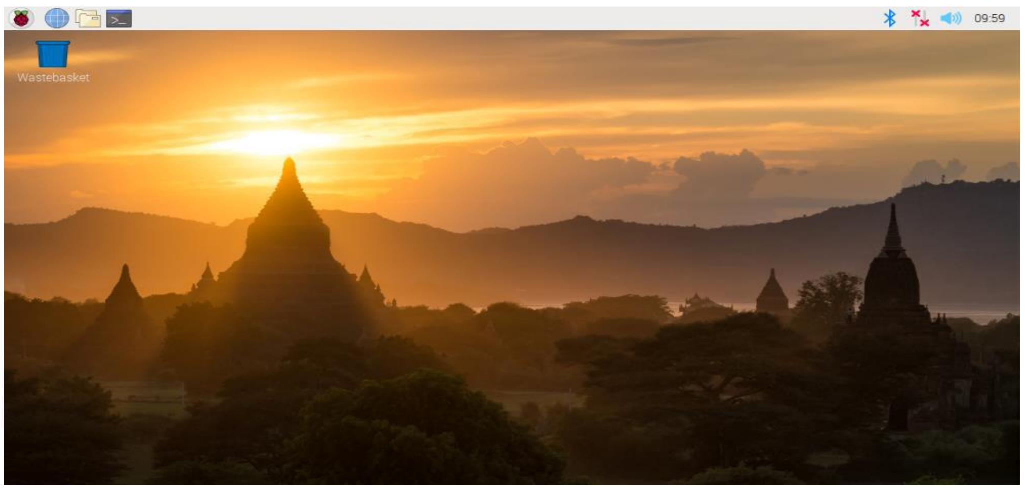


o Plug the power supply into a socket and then connect it to your Raspberry Pi's USB power port.

o You should see a red light on your Raspberry Pi and raspberries on the monitor.



o Your Raspberry Pi then boots up into a graphical desktop.



Practical – 5

Aim – Give overview of Zetta.

Solution-

Zetta is an API first platform for the Internet of Things. This is a general overview of core concepts that are important to a Zetta developer.

Contents

Core Concepts

1. Server

The Zetta server is the highest level of abstraction in Zetta. The Zetta server contains Drivers, Scouts, Apps, and Server Extensions. A Zetta server will typically run on a hardware hub such as a BeagleBone Black, Intel Edison, or Raspberry Pi. The server itself coordinates interactions between all of the contained components to communicate with devices, and generate HTTP APIs that an API consumer can interact with.

2. Linking

Zetta servers allow for establishing a secure tunneled link between two servers. This connection takes care of network configurations, and firewalls that make cloud connected IoT solutions difficult to maintain.

3. Drivers Zetta drivers are state machine representations of devices. Drivers are primarily responsible for modeling devices, and interacting with the device on the physical level. These device models are then taken, and used to generate HTTP and JS APIs for use in Zetta.

4. Scouts

Zetta scouts serve as a discovery mechanism for devices that may be on the network, or require system resources to speak a specific protocol. Scouts will search for devices on a particular protocol, and report back to Zetta when they've been found. Scouts can also use identifying information about devices (e.g. a devices MAC address) to identify whether or not Zetta has interacted with the device before, and ensure any relevant data or security credentials are maintained when interacting with that device an additional time.

5. Apps

Apps are interactions between devices written in javascript. Zetta allows developers to create local interactions based on sensor streams, or changes in devices. These apps will function regardless of inter-Zetta connectivity, and allow for quick response times to certain events in the system.

6. Server Extensions

Zetta follows a pluggable model for extending functionality. Most of these will be server extensions. Server extensions will deal with API management, defining addtional APIs, or even adding security to your API.

7. Registry

The registry is a persistence layer for Zetta. It's a small database that lives in the server context, and holds information about devices connected to the server itself.

Zetta Deployment

A typical Zetta deployment will look something like below.

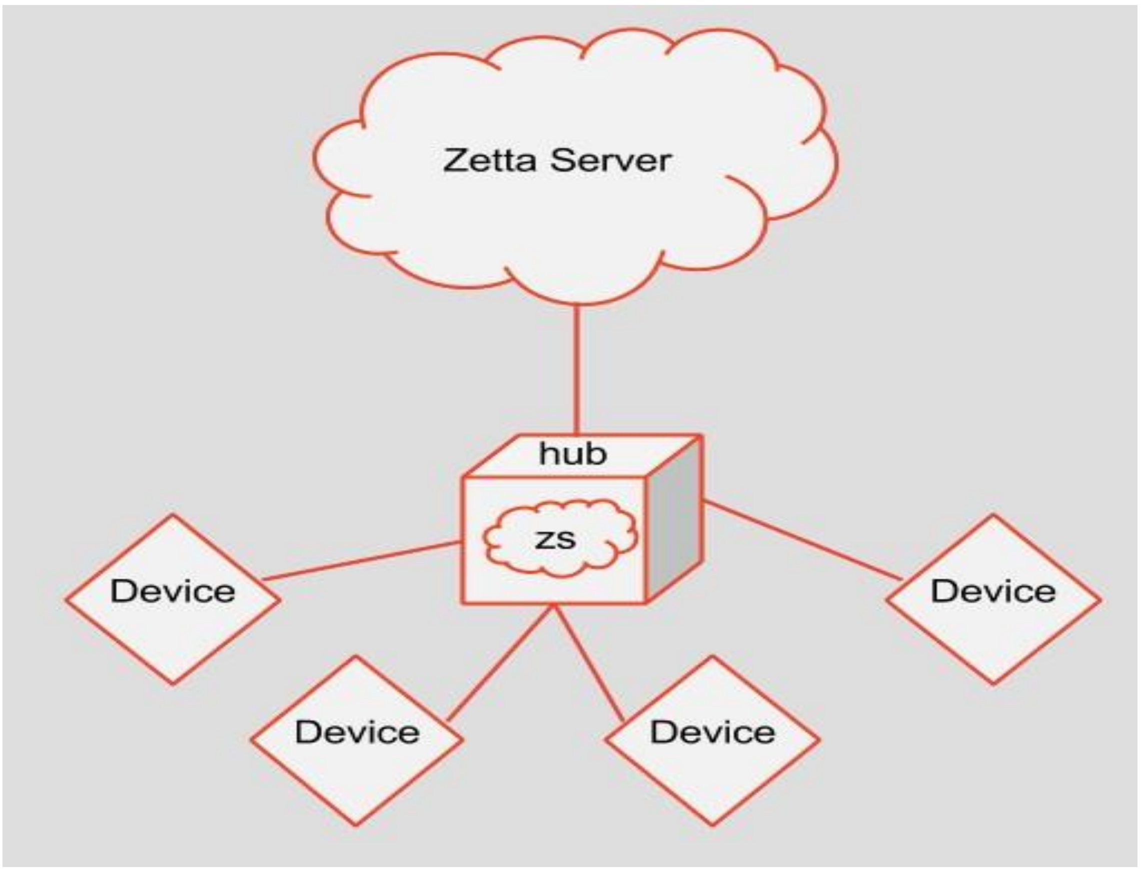
1. One Zetta server will live on a hardware hub. This hub is typically something like a BeagleBone Black, Intel Edison, or Raspberry Pi.

   The Zetta hub will connect to devices. Zetta will mediate from HTTP to the particular protocols used in a deployment.

2. Another Zetta server will live in the cloud. This server will use the exact same node package as Zetta on the hub.

   The Zetta sever on the hardware hub will connect to the server in the cloud.

3. Zetta will then expose an API to any consumers at the cloud endpoint.



Where do my APIs live?

APIs are on each instance of a Zetta server. Zetta uses hypermedia to expose a walkable set of links for navigating the API from a response, and affordances for streaming and interacting with devices. We conform to the Siren

1. Querying for devices on a particular server.

2. Setting up links between servers.

3. Interacting with Devices.

4. Streaming sensor data with websockets.

5. Registering hubless devices.

Practical – 6

Aim – Give overview of Putty.

Solution-

PuTTY is a free and open-source terminal emulator, serial console and network file transfer application. It supports several network protocols, like SCP, SSH, Telnet, rlogin, and raw socket connection. It can also connect to a serial port. The name "PuTTY" has no official meaning.

PuTTY can thus be said as a free implementation of SSH (and telnet) for PCs running Microsoft Windows (it also includes an xterm terminal emulator). PuTTY is also useful if you want to access an account on a Unix or other multi-user system from a PC (for example your own or one in an internet cafe).

PuTTY is actually an SSH and telnet client, developed originally by Simon Tatham for the Windows platform. It is open source software that is available with source code and is developed and supported by a group of volunteers.

Getting started with PuTTY

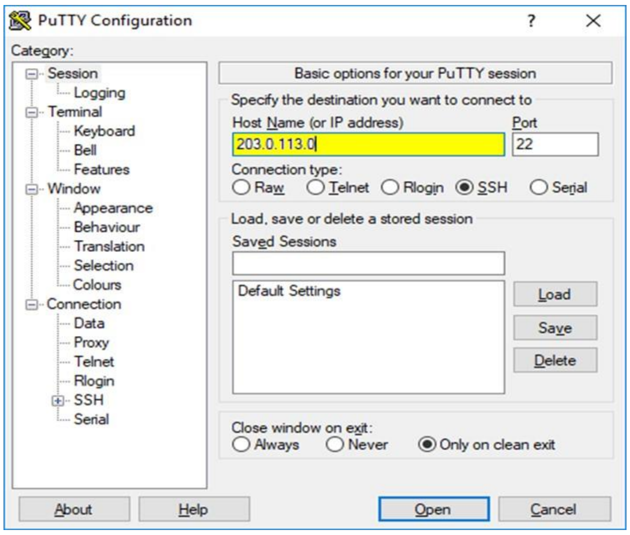
Starting a session

When you start PuTTY, you will see a dialog box. This dialog box allows you to control everything PuTTY can do enter a few basic parameters.

In the ‘Host Name’ box, enter the Internet host name of the server you want to connect to given by the provider

Now select a login protocol to use, from the ‘Connection type’ controls. For a login session, you should select SSH, Telnet, Rlogin, or SUPDUP.

Once you have filled in the ‘Host Name’, ‘Connection type’, and possibly ‘Port’ settings, you are ready to connect. Press the ‘Open’ button at the bottom of the dialog box, and PuTTY will begin trying to connect you to the server.



Specify the destination you want to connect to
Host Name (or IP address): 203.0.113.0
Port: 22
Connection type: Raw Telnet Rlogin [✓ SSH] Serial
Close window on exit: Always Never [✓ Only on clean exit]

Verifying the host key (SSH only)

When using SSH to connect to a server for the first time, you will probably see a message looking something like this:

The host key is not cached for this server: ssh.example.com (port 22) You have no guarantee that the server is the computer you think it is. The server's ssh-ed25519 key fingerprint is:

ssh-ed25519

255 SHA256:TddlQk20DVs4LRcAsIfDN9pInKpY06D+h4kSHwWAj4w

If you trust this host, press "Accept" to add the key to PuTTY's cache and carry on connecting.

If you want to carry on connecting just once, without adding the key to the cache, press "Connect Once".

If you do not trust this host, press "Cancel" to abandon the connection.

This is a feature of the SSH protocol. It is designed to protect you against a network attack known as spoofing: secretly redirecting your connection to a different computer, so that you send your password to the wrong machine. Using this technique, an attacker would be able to learn the password that guards your login account, and could then log in as if they were you and use the account for their own purposes.

To prevent this attack, each server has a unique identifying code, called a host key. These keys are created in a way that prevents one server from forging another server's key. So if you connect to a server and it sends you a different host key from the one you were expecting, PuTTY can warn you that the server may have been switched and that a spoofing attack might be in progress.

PuTTY records the host key for each server you connect to, in the Windows Registry. Every time you connect to a server, it checks that the host key presented by the server is the same host key as it was the last time you connected. If it is not, you will see a warning, and you will have the chance to abandon your connection before you type any private information (such as a password) into it.

When you connect to a server you have not connected to before, PuTTY has no way of telling whether the host key is the right one or not. So it gives the warning shown above, and asks you whether you want to trust this host key or not.

Whether or not to trust the host key is your choice. If you are connecting within a company network, you might feel that all the network users are on the same side and spoofing attacks are unlikely, so you might choose to trust the key without checking it. If you are connecting across a hostile network (such as the Internet), you should check with your system administrator, perhaps by telephone or in person. (When verifying the fingerprint, be careful with letters and numbers that can be confused with each other: 0/O, 1/I/l, and so on.)

Many servers have more than one host key. If the system administrator sends you more than one fingerprint, you should make sure the one PuTTY shows you is on the list, but it doesn't matter which one it is.

If you don't have any fingerprints that look like the example (SHA256: followed by a long string of characters), but instead have pairs of characters separated by colons like a4:db:96:a7:..., try pressing the ‘More info...’ button and see if you have a fingerprint matching the ‘MD5 fingerprint’ there. This is an older and less secure way to summarise the same underlying host key; it's possible for an attacker to create their own host key with the same fingerprint; so you should avoid relying on this fingerprint format unless you have no choice. The ‘More info...’ dialog box also shows the full host public key, in case that is easier to compare than a fingerprint.

Logging in

After you have connected, and perhaps verified the server's host key, you will be asked to log in, probably using a username and a password.

Your system administrator should have provided you with these. (If, instead, your system administrator has asked you to provide, or provided you with, a ‘public key’ or ‘key file’ text,

PuTTY will display a window (the ‘terminal window’ – it will have a black background unless you've changed the defaults), and prompt you to type your username and password into that window. (These prompts will include the PuTTY icon, to distinguish them from any text sent by the server in the same window.)

Enter the username and the password, and the server should grant you access and begin your session. If you have mistyped your password, most servers will give you several chances to get it right.

While you are typing your password, you will not usually see the cursor moving in the window, but PuTTY is registering what you type, and will send it when you press Return. (It works this way to avoid revealing the length of your password to anyone watching your screen.)

If SSH is used , we have to be careful SSH servers do not permit to make two login attempts. If username is wrong, you must close PuTTY and start again.

After logging in

After you log in to the server, what happens next is up to the server! Most servers will print some sort of login message and then present a prompt, at which you can type commands which the server will carry out. Some servers will offer you on- line help; others might not. If you are in doubt about what to do next, consult your system administrator.

Logging out

When the session is finished ,you should log out by typing the server's own logout command. This might vary between servers; if in doubt, try logout or exit, or consult a manual or the system administrator. When the server processes the logout command, the PuTTY window should close itself automatically.

<u>It's Components-</u>

PuTTY consists of several components:

1.PuTTY

the Telnet, rlogin, and SSH client itself, which can also connect to a serial port
2.PSCP

an SCP client, i.e. command-line secure file copy. Can also use SFTP to perform transfers

3.PSFTP

an SFTP client, i.e. general file transfer sessions much like FTP

4.PuTTYtel

a Telnet-only client

5.Plink

a command-line interface to the PuTTY back ends. Usually used for SSH Tunneling

6.Pageant

an SSH authentication agent for PuTTY, PSCP and Plink

7.PuTTYgen

an RSA, DSA, ECDSA and EdDSA key generation utility

8.pterm

(Unix version only) an X11 client which supports the same terminal emulation as PuTTY

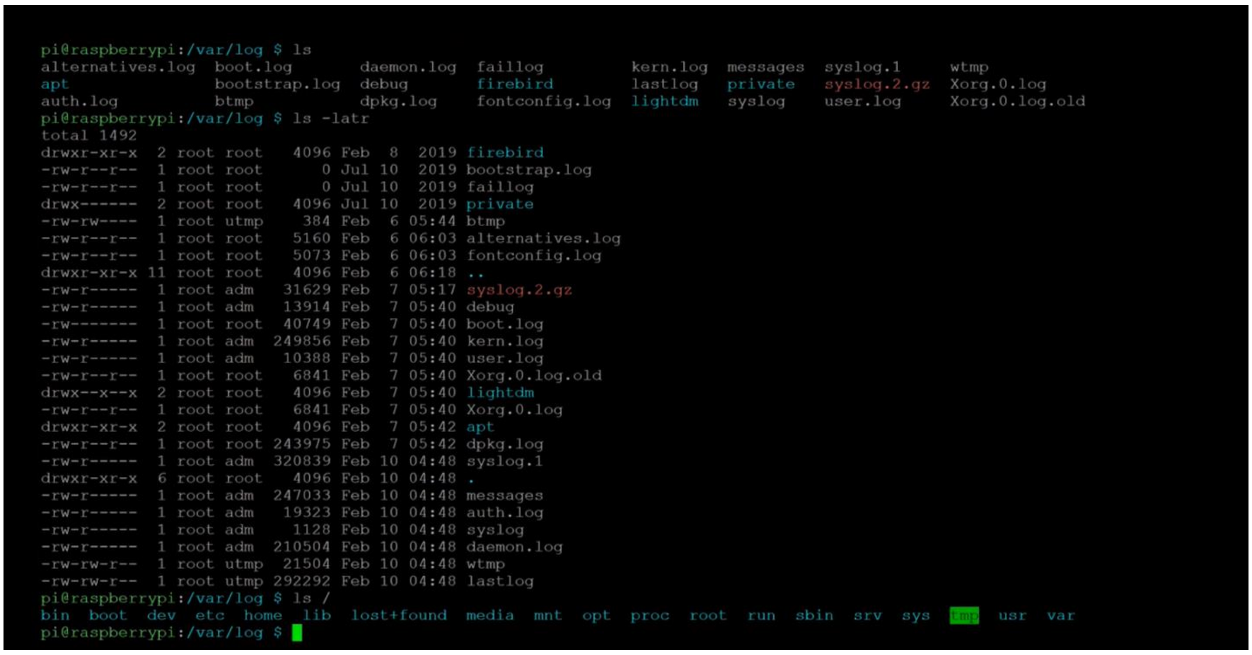
Practical: 7

Aim: Start Raspberry Pi and try various Linux commands in command terminal window: ls, cd, touch, mv, rm, man, mkdir, rmdir, tar, gzip, cat, more, less, ps, sudo, cron, chown,chgrp, ping etc.

Solution-

ls: List files and directories in the current directory.

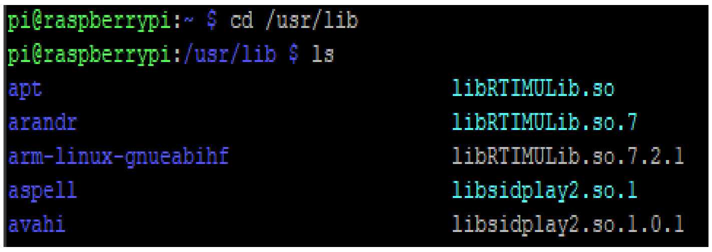
-ls



```
pi@raspberrypi:/var/log $ ls
alternatives.log  boot.log       daemon.log  faillog         kern.log  messages  syslog.1     wtmp
apt               bootstrap.log  debug       firebird        lastlog   private   syslog.2.gz  Xorg.0.log
auth.log          btmp           dpkg.log    fontconfig.log  lightdm   syslog    user.log     Xorg.0.log.old
pi@raspberrypi:/var/log $ ls -latr
total 1492
drwxr-xr-x  2 root root   4096 Feb  8  2019 firebird
-rw-r--r--  1 root root      0 Jul 10  2019 bootstrap.log
-rw-r--r--  1 root root      0 Jul 10  2019 faillog
drwx------  2 root root   4096 Jul 10  2019 private
-rw-rw----  1 root utmp    384 Feb  6 05:44 btmp
-rw-r--r--  1 root root   5160 Feb  6 06:03 alternatives.log
-rw-r--r--  1 root root   5073 Feb  6 06:03 fontconfig.log
drwxr-xr-x 11 root root   4096 Feb  6 06:18 ..
-rw-r-----  1 root adm   31629 Feb  7 05:17 syslog.2.gz
-rw-r-----  1 root adm   13914 Feb  7 05:40 debug
-rw-------  1 root root  40749 Feb  7 05:40 boot.log
-rw-r-----  1 root adm  249856 Feb  7 05:40 kern.log
-rw-r-----  1 root adm   10388 Feb  7 05:40 user.log
-rw-r--r--  1 root root   6841 Feb  7 05:40 Xorg.0.log.old
drwx--x--x  2 root root   4096 Feb  7 05:40 lightdm
-rw-r--r--  1 root root   6841 Feb  7 05:40 Xorg.0.log
drwxr-xr-x  2 root root   4096 Feb  7 05:42 apt
-rw-r--r--  1 root root 243975 Feb  7 05:42 dpkg.log
-rw-r-----  1 root adm  320839 Feb 10 04:48 syslog.1
drwxr-xr-x  6 root root   4096 Feb 10 04:48 .
-rw-r-----  1 root adm  247033 Feb 10 04:48 messages
-rw-r-----  1 root adm   19323 Feb 10 04:48 auth.log
-rw-r-----  1 root adm    1128 Feb 10 04:48 syslog
-rw-r-----  1 root adm  210504 Feb 10 04:48 daemon.log
-rw-rw-r--  1 root utmp  21504 Feb 10 04:48 wtmp
-rw-rw-r--  1 root utmp 292292 Feb 10 04:48 lastlog
pi@raspberrypi:/var/log $ ls /
bin  boot  dev  etc  home  lib  lost+found  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
pi@raspberrypi:/var/log $ 
```

cd: Change the current directory.

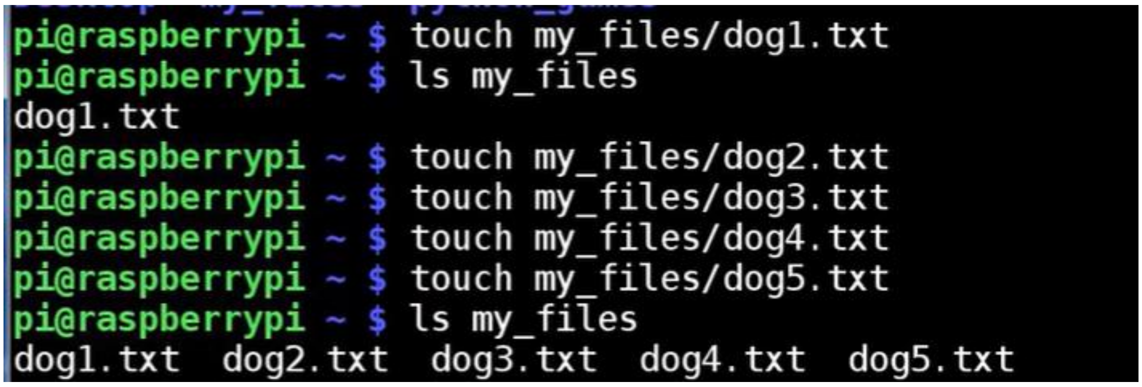
-cd /path/to/directory



```
pi@raspberrypi:~ $ cd /usr/lib
pi@raspberrypi:/usr/lib $ ls
apt                                libRTIMULib.so
arandr                             libRTIMULib.so.7
arm-linux-gnueabihf                libRTIMULib.so.7.2.1
aspell                             libsidplay2.so.1
avahi                              libsidplay2.so.1.0.1
```

touch: Create an empty file.

-touch filename.txt

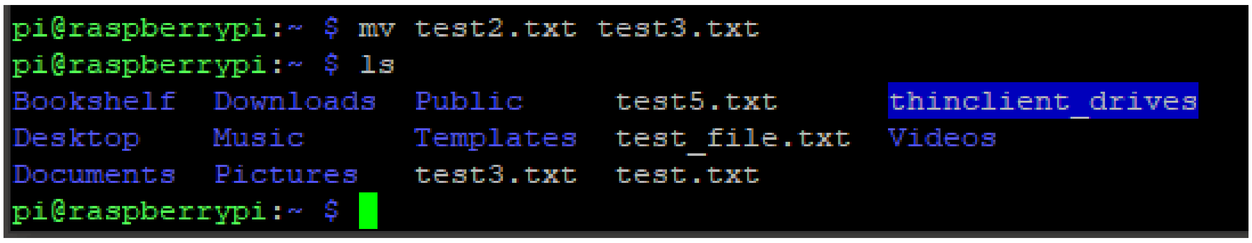


```
pi@raspberrypi ~ $ touch my_files/dog1.txt
pi@raspberrypi ~ $ ls my_files
dog1.txt
pi@raspberrypi ~ $ touch my_files/dog2.txt
pi@raspberrypi ~ $ touch my_files/dog3.txt
pi@raspberrypi ~ $ touch my_files/dog4.txt
pi@raspberrypi ~ $ touch my_files/dog5.txt
pi@raspberrypi ~ $ ls my_files
dog1.txt  dog2.txt  dog3.txt  dog4.txt  dog5.txt
```

mv: Move or rename files and directories.

-mv oldfile.txt newfile.txt

mv directory1 directory2

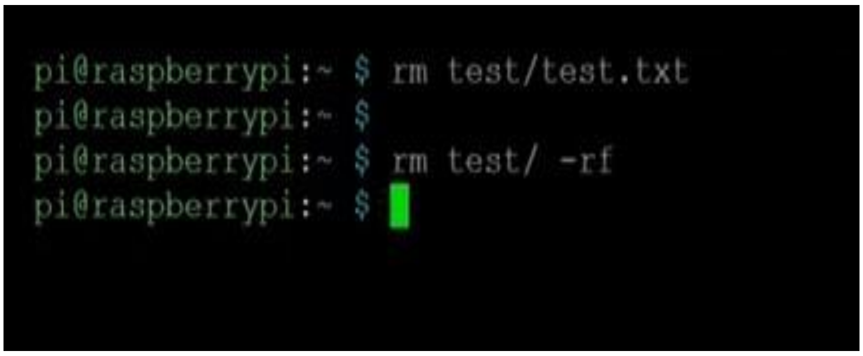


```
pi@raspberrypi:~ $ mv test2.txt test3.txt
pi@raspberrypi:~ $ ls
Bookshelf  Downloads  Public     test5.txt      thinclient_drives
Desktop    Music      Templates  test_file.txt  Videos
Documents  Pictures   test3.txt  test.txt
pi@raspberrypi:~ $
```

rm: Remove files or directories.

-rm filename.txt

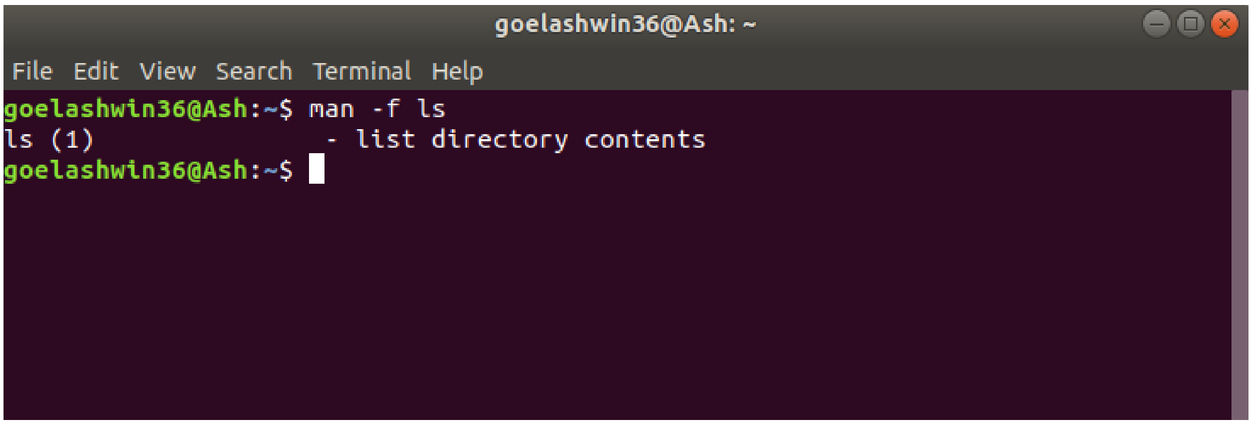
rm -r directory



```
pi@raspberrypi:~ $ rm test/test.txt
pi@raspberrypi:~ $
pi@raspberrypi:~ $ rm test/ -rf
pi@raspberrypi:~ $
```

man: Access the manual pages for commands.

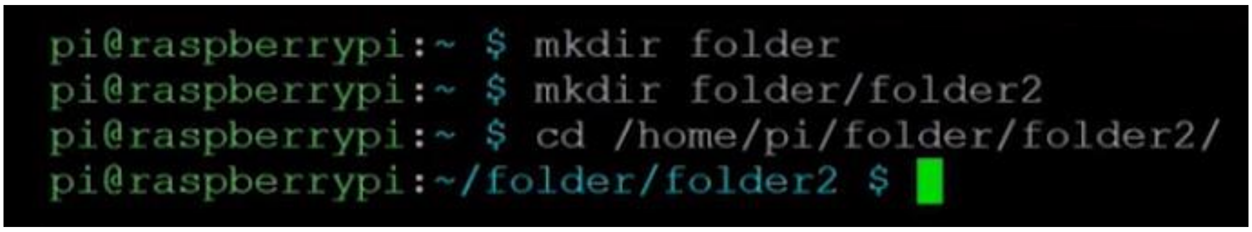
-man ls



```
goelashwin36@Ash:~$ man -f ls
ls (1)               - list directory contents
goelashwin36@Ash:~$
```

mkdir: Create a new directory.

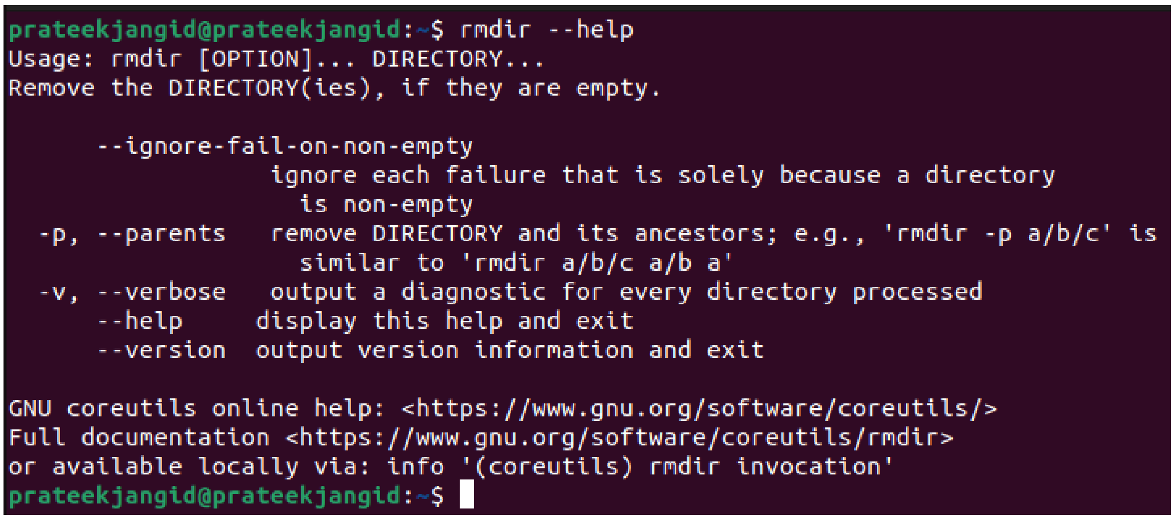
-mkdir new_directory



```
pi@raspberrypi:~ $ mkdir folder
pi@raspberrypi:~ $ mkdir folder/folder2
pi@raspberrypi:~ $ cd /home/pi/folder/folder2/
pi@raspberrypi:~/folder/folder2 $
```

rmdir: Remove an empty directory.

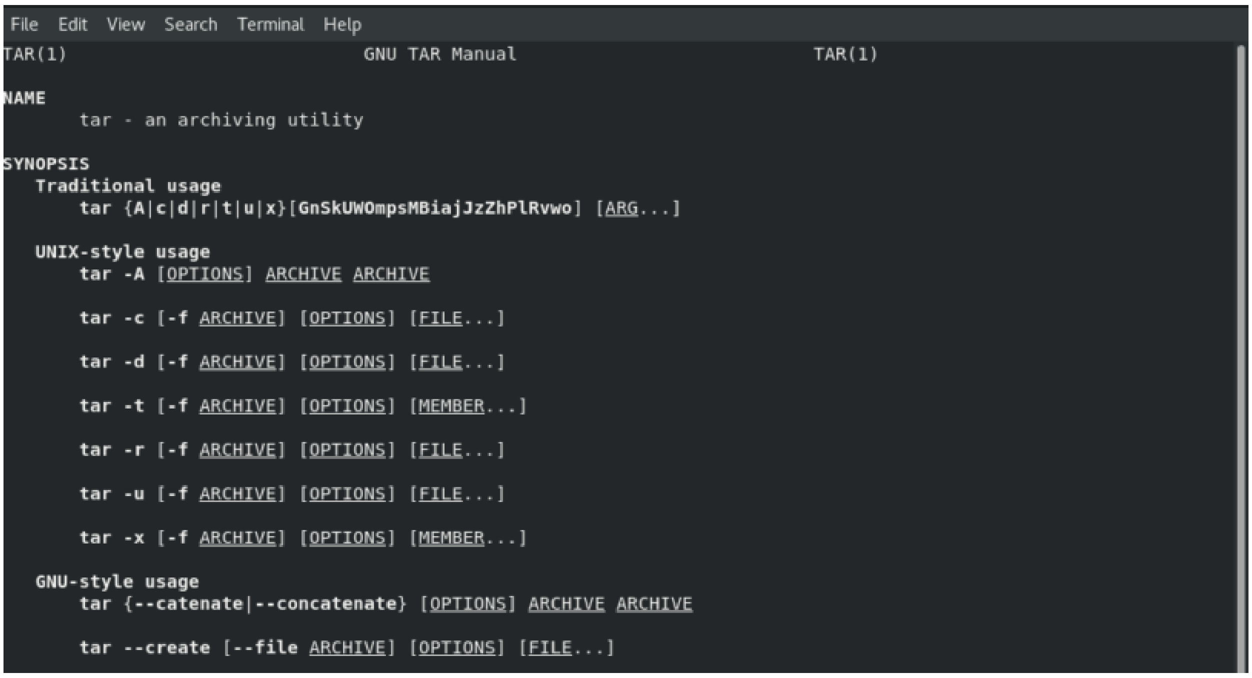
-rmdir empty_directory



```
prateekjangid@prateekjangid:~$ rmdir --help
Usage: rmdir [OPTION]... DIRECTORY...
Remove the DIRECTORY(ies), if they are empty.

      --ignore-fail-on-non-empty
                  ignore each failure that is solely because a directory
                    is non-empty
  -p, --parents   remove DIRECTORY and its ancestors; e.g., 'rmdir -p a/b/c' is
                    similar to 'rmdir a/b/c a/b a'
  -v, --verbose   output a diagnostic for every directory processed
      --help     display this help and exit
      --version  output version information and exit

GNU coreutils online help: <https://www.gnu.org/software/coreutils/>
Full documentation <https://www.gnu.org/software/coreutils/rmdir>
or available locally via: info '(coreutils) rmdir invocation'
prateekjangid@prateekjangid:~$
```

tar: Archive files and directories.

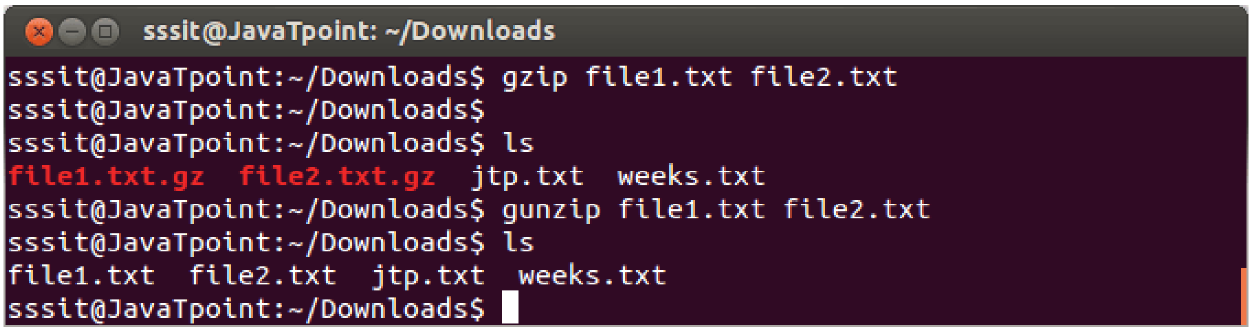
tar -cvzf archive.tar.gz directory/



```
TAR(1)                          GNU TAR Manual                          TAR(1)

NAME
       tar - an archiving utility

SYNOPSIS
   Traditional usage
       tar {A|c|d|r|t|u|x}[GnSkUWOmpsMBiajJzZhPlRvwo] [ARG...]

   UNIX-style usage
       tar -A [OPTIONS] ARCHIVE ARCHIVE

       tar -c [-f ARCHIVE] [OPTIONS] [FILE...]

       tar -d [-f ARCHIVE] [OPTIONS] [FILE...]

       tar -t [-f ARCHIVE] [OPTIONS] [MEMBER...]

       tar -r [-f ARCHIVE] [OPTIONS] [FILE...]

       tar -u [-f ARCHIVE] [OPTIONS] [FILE...]

       tar -x [-f ARCHIVE] [OPTIONS] [MEMBER...]

   GNU-style usage
       tar {--catenate|--concatenate} [OPTIONS] ARCHIVE ARCHIVE

       tar --create [--file ARCHIVE] [OPTIONS] [FILE...]
```

gzip: Compress files.

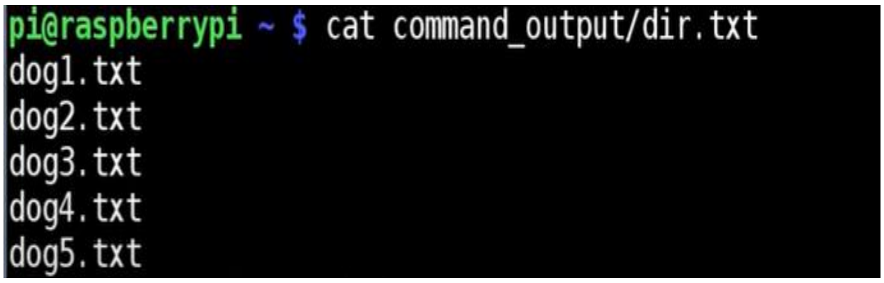
-gzip filename.txt



```
sssit@JavaTpoint:~/Downloads$ gzip file1.txt file2.txt
sssit@JavaTpoint:~/Downloads$
sssit@JavaTpoint:~/Downloads$ ls
file1.txt.gz  file2.txt.gz  jtp.txt  weeks.txt
sssit@JavaTpoint:~/Downloads$ gunzip file1.txt file2.txt
sssit@JavaTpoint:~/Downloads$ ls
file1.txt  file2.txt  jtp.txt  weeks.txt
sssit@JavaTpoint:~/Downloads$
```

cat: Display the contents of a file.

-cat filename.txt



```
pi@raspberrypi ~ $ cat command_output/dir.txt
dog1.txt
dog2.txt
dog3.txt
dog4.txt
dog5.txt
```

more: View the contents of a file one screen at a time.

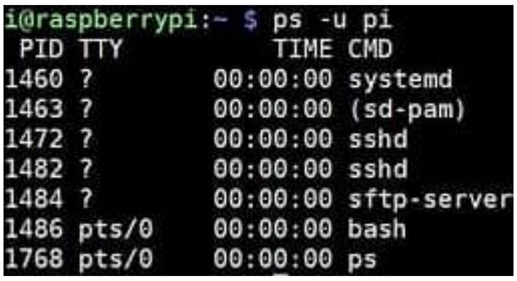
-more longfile.txt

less: View the contents of a file with backward navigation.

-less largefile.txt

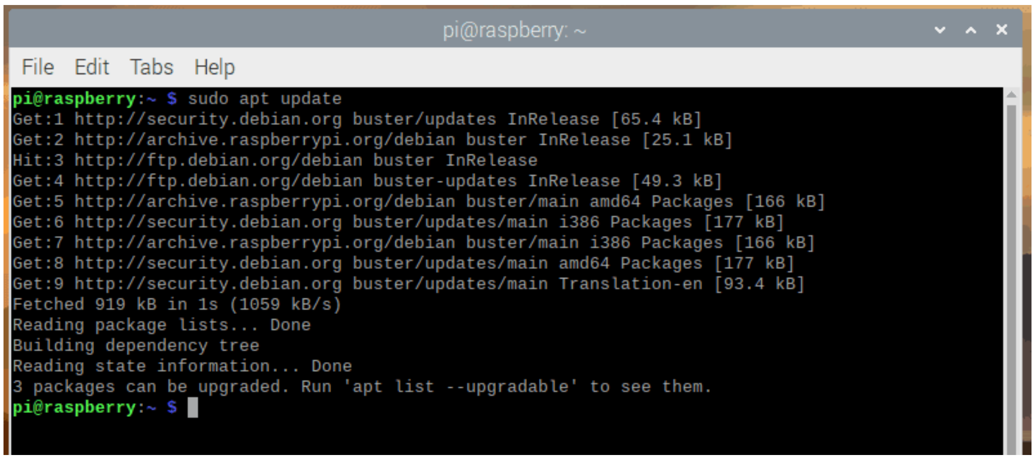
ps: List running processes.

-ps aux


```
i@raspberrypi:~ $ ps -u pi
 PID TTY          TIME CMD
1460 ?        00:00:00 systemd
1463 ?        00:00:00 (sd-pam)
1472 ?        00:00:00 sshd
1482 ?        00:00:00 sshd
1484 ?        00:00:00 sftp-server
1486 pts/0    00:00:00 bash
1768 pts/0    00:00:00 ps
```

sudo: Execute a command with superuser (root) privileges.

-sudo apt update

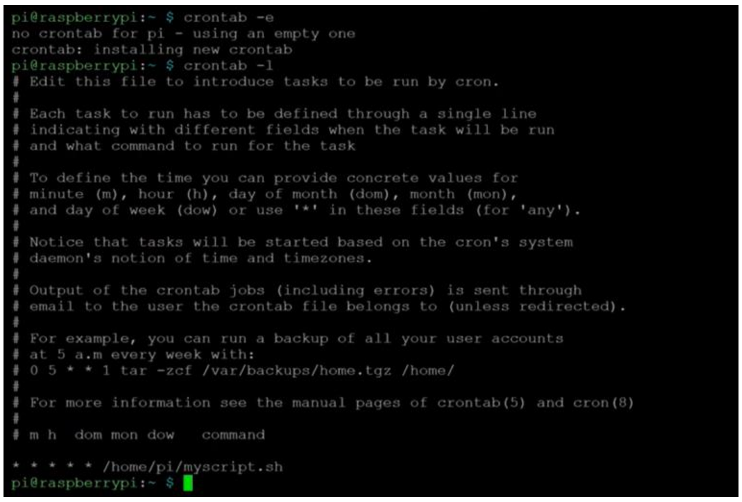


```
pi@raspberry:~ $ sudo apt update
Get:1 http://security.debian.org buster/updates InRelease [65.4 kB]
Get:2 http://archive.raspberrypi.org/debian buster InRelease [25.1 kB]
Hit:3 http://ftp.debian.org/debian buster InRelease
Get:4 http://ftp.debian.org/debian buster-updates InRelease [49.3 kB]
Get:5 http://archive.raspberrypi.org/debian buster/main amd64 Packages [166 kB]
Get:6 http://security.debian.org buster/updates/main i386 Packages [177 kB]
Get:7 http://archive.raspberrypi.org/debian buster/main i386 Packages [166 kB]
Get:8 http://security.debian.org buster/updates/main amd64 Packages [177 kB]
Get:9 http://security.debian.org buster/updates/main Translation-en [93.4 kB]
Fetched 919 kB in 1s (1059 kB/s)
Reading package lists... Done
Building dependency tree
Reading state information... Done
3 packages can be upgraded. Run 'apt list --upgradable' to see them.
pi@raspberry:~ $ 
```

There are other sudo commands too.

cron: Schedule tasks to run at specific times.

-crontab -e



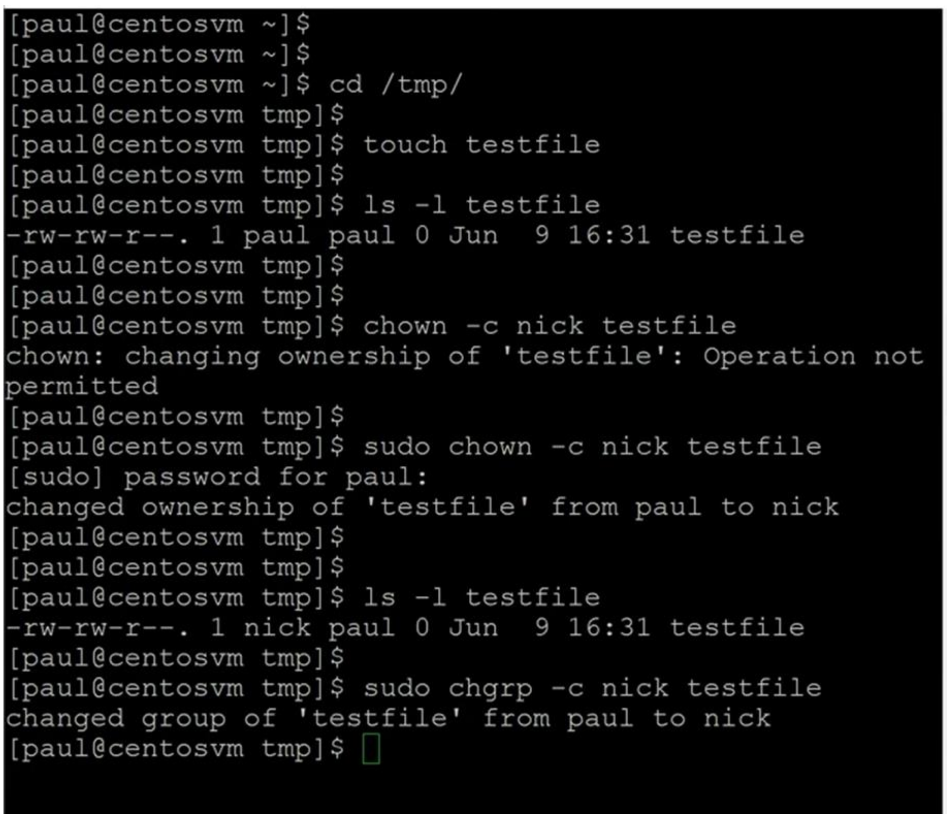
```
pi@raspberrypi:~ $ crontab -e
no crontab for pi - using an empty one
crontab: installing new crontab
pi@raspberrypi:~ $ crontab -l
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').
#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command

* * * * * /home/pi/myscript.sh
pi@raspberrypi:~ $
```

chown: Change file or directory ownership.

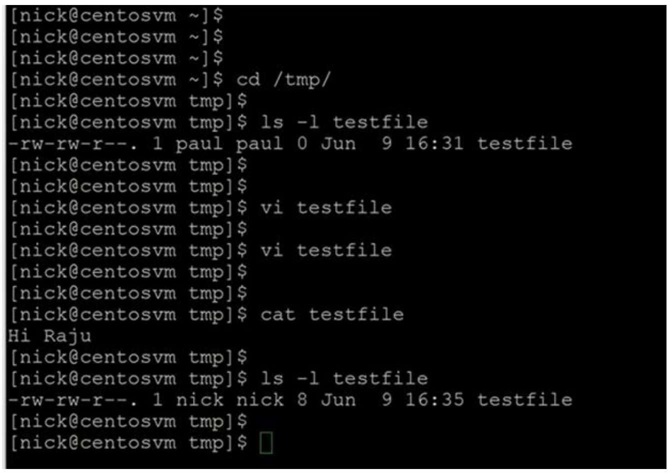
-sudo chown username:groupname filename.txt

chgrp: Change group ownership of files.

-sudo chgrp groupname filename.txt



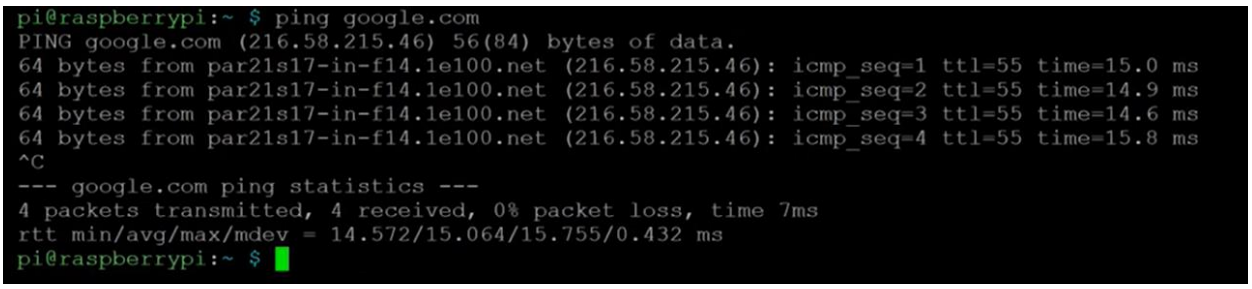
```
[paul@centosvm ~]$
[paul@centosvm ~]$
[paul@centosvm ~]$ cd /tmp/
[paul@centosvm tmp]$
[paul@centosvm tmp]$ touch testfile
[paul@centosvm tmp]$
[paul@centosvm tmp]$ ls -l testfile
-rw-rw-r--. 1 paul paul 0 Jun  9 16:31 testfile
[paul@centosvm tmp]$
[paul@centosvm tmp]$
[paul@centosvm tmp]$ chown -c nick testfile
chown: changing ownership of 'testfile': Operation not
permitted
[paul@centosvm tmp]$
[paul@centosvm tmp]$ sudo chown -c nick testfile
[sudo] password for paul:
changed ownership of 'testfile' from paul to nick
[paul@centosvm tmp]$
[paul@centosvm tmp]$
[paul@centosvm tmp]$ ls -l testfile
-rw-rw-r--. 1 nick paul 0 Jun  9 16:31 testfile
[paul@centosvm tmp]$
[paul@centosvm tmp]$ sudo chgrp -c nick testfile
changed group of 'testfile' from paul to nick
[paul@centosvm tmp]$
```



```
[nick@centosvm ~]$
[nick@centosvm ~]$
[nick@centosvm ~]$
[nick@centosvm ~]$ cd /tmp/
[nick@centosvm tmp]$
[nick@centosvm tmp]$ ls -l testfile
-rw-rw-r--. 1 paul paul 0 Jun  9 16:31 testfile
[nick@centosvm tmp]$
[nick@centosvm tmp]$
[nick@centosvm tmp]$ vi testfile
[nick@centosvm tmp]$
[nick@centosvm tmp]$ vi testfile
[nick@centosvm tmp]$
[nick@centosvm tmp]$
[nick@centosvm tmp]$ cat testfile
Hi Raju
[nick@centosvm tmp]$
[nick@centosvm tmp]$ ls -l testfile
-rw-rw-r--. 1 nick nick 8 Jun  9 16:35 testfile
[nick@centosvm tmp]$
[nick@centosvm tmp]$
```

ping: Check network connectivity to a host.

-ping google.com



```
pi@raspberrypi:~ $ ping google.com
PING google.com (216.58.215.46) 56(84) bytes of data.
64 bytes from par21s17-in-f14.1e100.net (216.58.215.46): icmp_seq=1 ttl=55 time=15.0 ms
64 bytes from par21s17-in-f14.1e100.net (216.58.215.46): icmp_seq=2 ttl=55 time=14.9 ms
64 bytes from par21s17-in-f14.1e100.net (216.58.215.46): icmp_seq=3 ttl=55 time=14.6 ms
64 bytes from par21s17-in-f14.1e100.net (216.58.215.46): icmp_seq=4 ttl=55 time=15.8 ms
^C
--- google.com ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 7ms
rtt min/avg/max/mdev = 14.572/15.064/15.755/0.432 ms
pi@raspberrypi:~ $
```

Practical: 8

Aim: Write Run some c++ programs on arduino like:

a) Read your name and print Hello message with name.

b) Read two numbers and print their sum, difference, product and division.

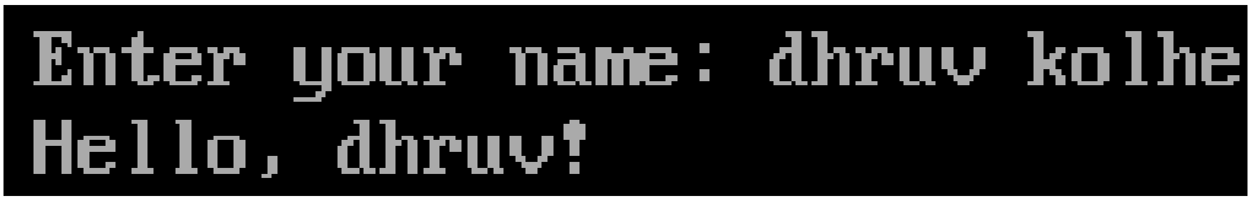
c) Word and character count of a given string.

Solution-

a)

```
void setup() {
  Serial.begin(9600);  // Initialize Serial communication
}
void loop() {
  String name;
  Serial.println("Enter your name: ");
  while (!Serial.available()) {
    // Wait for user input
  }
  // Read the name from Serial Monitor
  name = Serial.readString();
  // Print the "Hello" message with the name
  Serial.print("Hello, ");
  Serial.println(name);
  // Wait for a moment before repeating
  delay(1000);  // Delay for 1 second
}
```

Output-

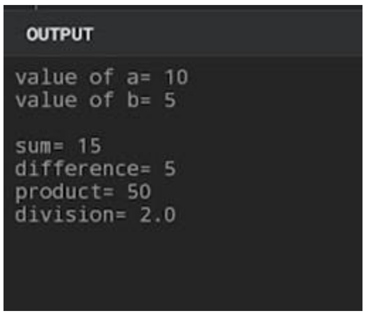


```
Enter your name: dhruv kolhe
Hello, dhruv!
```

b)

```
void setup() {
  Serial.begin(9600);
  double a = 10.0;
  double b = 5.0;
  Serial.print("Value of a (a): ");
  Serial.println(a);
  Serial.print(Value of b (b): ");
  // Uncomment the following lines to read 'b' from Serial input
  // while (!Serial.available()) {}
  // b = Serial.parseFloat();
  Serial.println(b);
  double sum = a + b;
  double difference = a - b;
  double product = a * b;
  if (b != 0) {
    double division = a / b;
    Serial.print("Sum (a + b): ");
    Serial.println(sum);
    Serial.print("Difference (a - b): ");
    Serial.println(difference);
    Serial.print("Product (a * b): ");
    Serial.println(product);
    Serial.print("Division (a / b): ");
    Serial.println(division);
  } else {
    Serial.println("Error: Division by zero is not allowed.");
  }
}
void loop() {
}
```

Output-

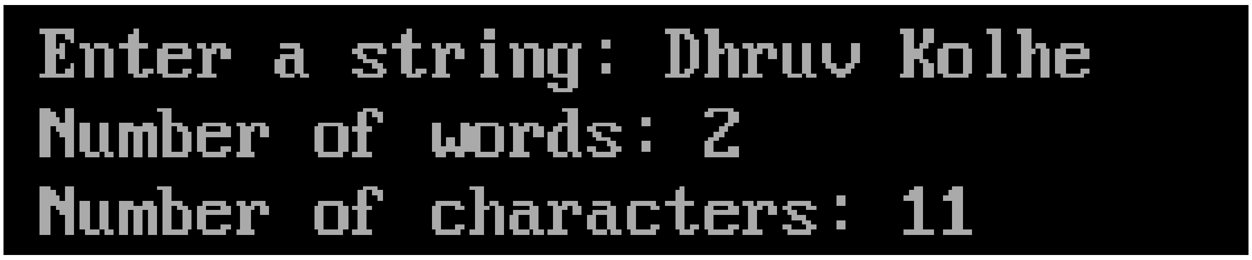


```
OUTPUT
value of a= 10
value of b= 5

sum= 15
difference= 5
product= 50
division= 2.0
```

c)

```
void setup() {
  Serial.begin(9600);
  while (!Serial) {
  }
}
void loop() {
  char str[100];
  int i, count_words = 0, count_chars = 0;
  Serial.println("Enter a string:");
  while (Serial.available() == 0) {
  }
  i = 0;
  while (Serial.available() > 0) {
    char c = Serial.read();
    if (c == '\n' || i == sizeof(str) - 1) {
      str[i] = '\0';
      break;
    }
    str[i] = c;
    i++;
  }
  for (i = 0; str[i] != '\0'; i++) {
    if (str[i] == ' ' || str[i] == '\t' || str[i] == '\n') {
      if (i > 0 && (str[i - 1] != ' ' && str[i - 1] != '\t' && str[i - 1] != '\n')) {
        count_words++;
      }
    }
    count_chars++;
  }
  if (count_chars > 0 && (str[count_chars - 1] != ' ' && str[count_chars - 1] != '\t'
&& str[count_chars - 1] != '\n')) {
    count_words++;
  }
  Serial.print("Number of words: ");
  Serial.println(count_words);
  Serial.print("Number of characters: ");
  Serial.println(count_chars);
  delay(1000);
}
```

Output-



```
Enter a string: Dhruv Kolhe
Number of words: 2
Number of characters: 11
```

Practical: 9

Aim: Run some c++ programs on arduino like: a) Print a name 'n' time, where name and n are read from standard input, using for and b) while loops.

c) Handle Divided by Zero Exception.

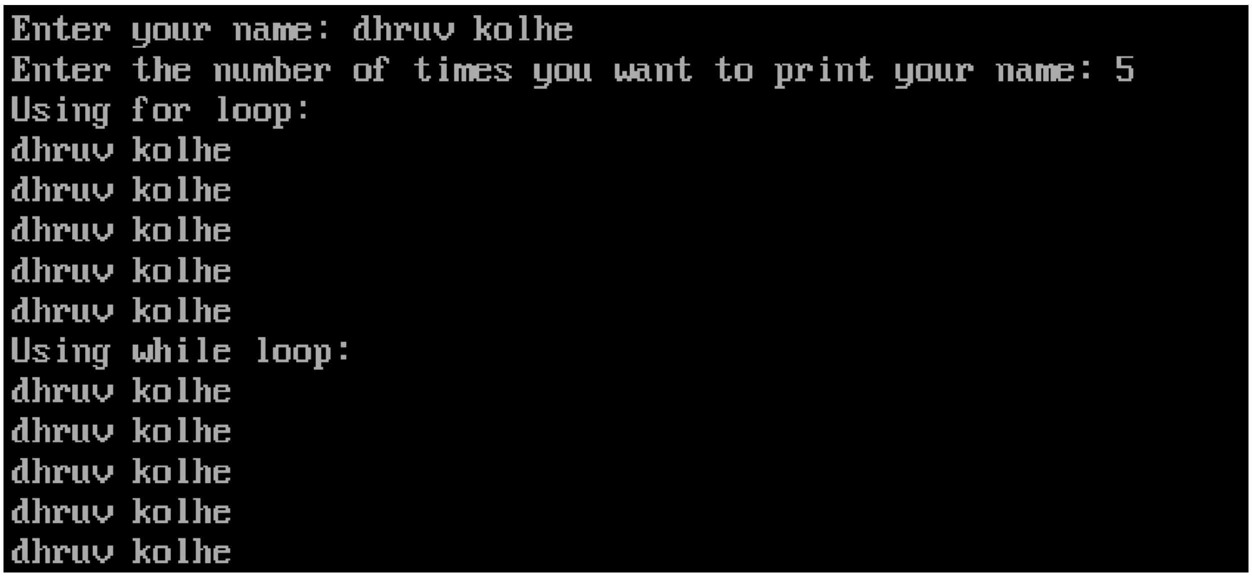
d) Print current time for 10 times with an interval of 10 seconds.

Solution-

a)

```
#include <iostream>

using namespace std;

int main() {

  string name;

  int n;

  cout << "Enter your name: ";

  cin >> name;

  cout << "Enter the number of times you want to print your name: ";

  cin >> n;

  cout << "Using for loop:" << endl;

  for (int i = 0; i < n; i++) {

    cout << name << endl;

  }

  cout << "Using while loop:" << endl;

  int i = 0;
  while (i < n) {
    cout << name << endl;
    i++;
  }
  return 0;
}
```

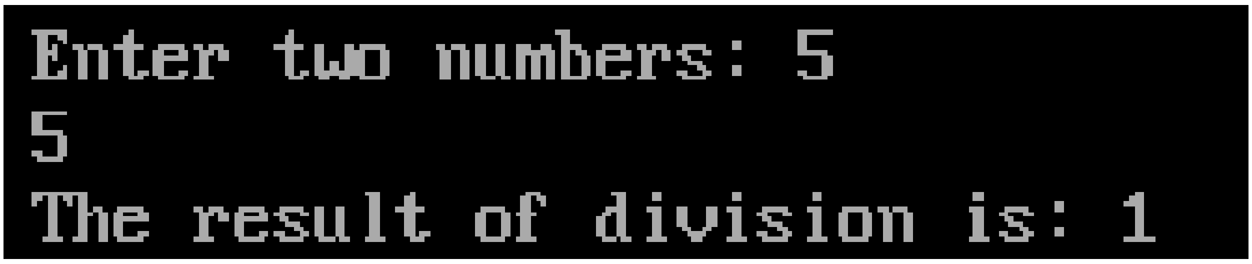
Output-



```
Enter your name: dhruv kolhe
Enter the number of times you want to print your name: 5
Using for loop:
dhruv kolhe
dhruv kolhe
dhruv kolhe
dhruv kolhe
dhruv kolhe
Using while loop:
dhruv kolhe
dhruv kolhe
dhruv kolhe
dhruv kolhe
dhruv kolhe
```

```
// c) Handle Divided by Zero Exception.
#include <iostream>
using namespace std;
int main() {
  int a, b;
  cout << "Enter two numbers: ";
  cin >> a >> b;
  try {
    if (b == 0) {
      throw runtime_error("Division by zero");
    }
    cout << "The result of division is: " << a / b << endl;
  } catch (runtime_error& e) {
    cout << e.what() << endl;
  }
  return 0;
}
```

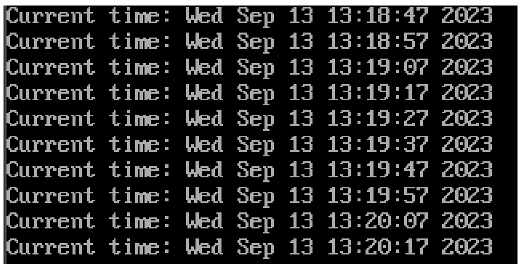
Output-



```
Enter two numbers: 5
5
The result of division is: 1
```

```
// d) Print current time for 10 times with an interval of 10 seconds.
#include <iostream>
#include <chrono>
#include <thread>
using namespace std;
int main() {
  for (int i = 0; i < 10; i++) {
    auto now = chrono::system_clock::now();
    time_t t = chrono::system_clock::to_time_t(now);
    cout << "Current time: " << ctime(&t) << endl;
    this_thread::sleep_for(chrono::seconds(10));
  }
  return 0;
}
```

Output-



```
Current time: Wed Sep 13 13:18:47 2023
Current time: Wed Sep 13 13:18:57 2023
Current time: Wed Sep 13 13:19:07 2023
Current time: Wed Sep 13 13:19:17 2023
Current time: Wed Sep 13 13:19:27 2023
Current time: Wed Sep 13 13:19:37 2023
Current time: Wed Sep 13 13:19:47 2023
Current time: Wed Sep 13 13:19:57 2023
Current time: Wed Sep 13 13:20:07 2023
Current time: Wed Sep 13 13:20:17 2023
```

Practical: 10

Aim: LED bilking using Arduino on Proteus using Flowchart.

Solution-

Components-

- 1 × Breadboard
- 1 × Arduino Uno R3
- 1 × LED
- 1 × 330Ω Resistor
- 2 × Jumpe

Procedure

Here's a step-by-step procedure for simulating LED blinking using an Arduino in Proteus:

i. Open Proteus: Launch the Proteus Design Suite on your computer.

ii. Create a New Project: Start a new project or open an existing one.

iii. Add Components: In the Proteus schematic view, add the following components to the workspace:
- Arduino Board (e.g., Arduino Uno)
- LED
- Resistor (220-330 Ohms)
- Wires (Jumper wires for connections)

iv. Connect Components: Wire the components together following this schematic:
- Connect one end of the resistor to the LED's shorter lead (cathode).
- Connect the other end of the resistor to the GND (ground) pin on the Arduino.
- Connect the LED's longer lead (anode) to one of the digital pins on the Arduino (e.g., pin 13).
- Connect the other end of the LED to the GND (ground) on the Arduino.

v. Upload Arduino Code: Write the Arduino code for LED blinking (as provided in the previous response) in the Arduino IDE on your computer.
   - Compile the code to check for errors.
   - Upload the compiled code to the Arduino board in Proteus.
   - Ensure that the Arduino board is selected in Proteus (click on it), and in the Properties panel, browse for the HEX file generated by the Arduino IDE.

vi. Configure Power Supply : If your Arduino is not USB-powered and requires an external power supply, configure the power supply in Proteus accordingly.

vii. Simulate the Circuit: Start the simulation in Proteus.
   - You should see the LED connected to the Arduino blinking according to the code you uploaded.

viii. Observation: Observe the LED's behavior in the simulation to verify that it blinks at the specified intervals.

ix. Save and Close: Save your Proteus project for future reference.
   - Close the simulation when you are finished.

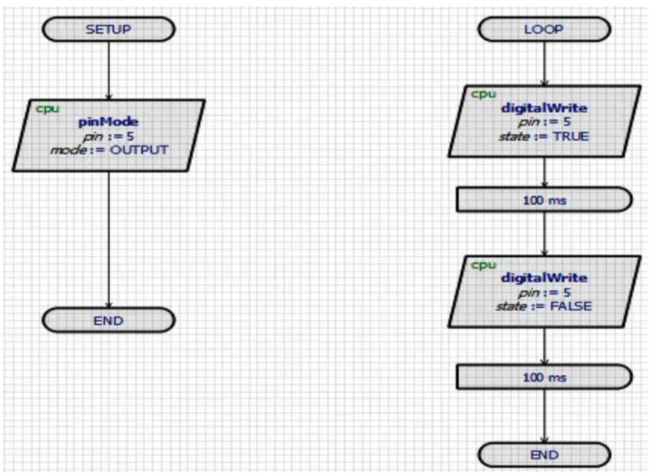
By following these steps, you can simulate LED blinking using an Arduino in Proteus with the specified components.

Code-

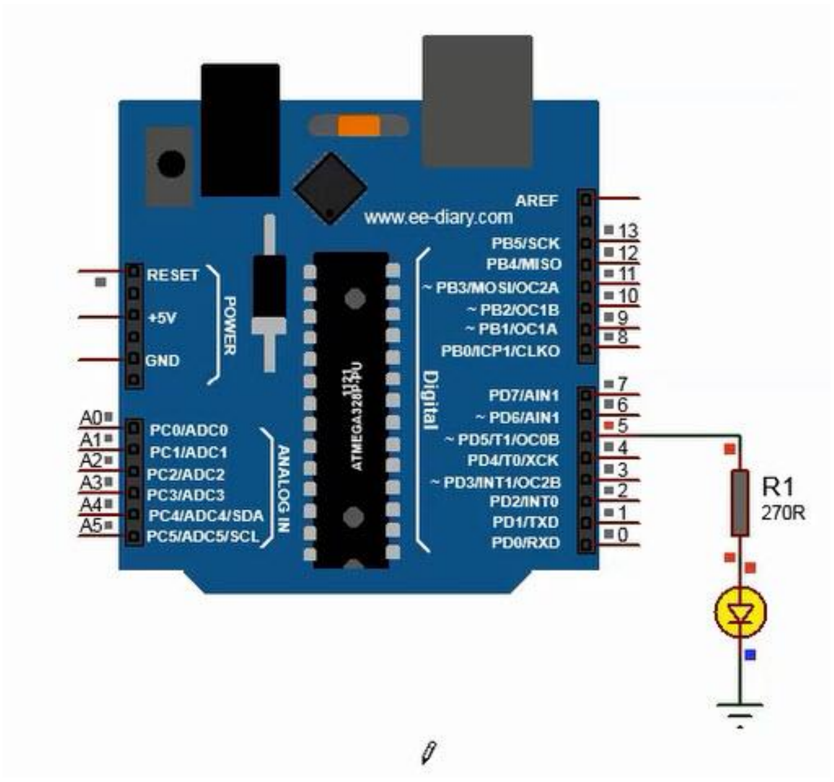
```
import RPi.GPIO as GPIO

import time

# GPIO pin for the LED (change to the appropriate pin)

led_pin = 13

# Initialize GPIO settings

GPIO.setmode(GPIO.BCM)

GPIO.setup(led_pin, GPIO.OUT)
```

```
# Initialize variables
led_state = GPIO.LOW  # Initial LED state is OFF
try:
    while True:
        # Check LED State
        if led_state == GPIO.LOW:
            # If LED is OFF, turn it ON
            GPIO.output(led_pin, GPIO.HIGH)
            led_state = GPIO.HIGH
        else:
            # If LED is ON, turn it OFF
            GPIO.output(led_pin, GPIO.LOW)
            led_state = GPIO.LOW
        # Delay for 1 second
        time.sleep(1)
except KeyboardInterrupt:
    # Clean up GPIO on program exit
    GPIO.cleanup()
```

Flowchart-



On the schematic editor we can run the simulation and we will see that the LED turns on and off with 100ms delay.



Practical: 11

Aim: LED bilking without push button using Arduino on Proteus using Flowchart.

Solution-

Here's a step-by-step procedure for simulating LED blinking without a push button using an Arduino in Proteus:

i. Open Proteus: Launch Proteus Design Suite on your computer.

ii. Create a New Project: Start a new project or open an existing one.

iii. Add Components: In the Proteus schematic view, add the following components to the workspace:
   - Arduino Board (e.g., Arduino Uno)
   - LED
   - Resistor (220-330 Ohms)
   - Wires (Jumper wires for connections)

iv. Connect Components:
   - Wire the components together following this schematic:
     - Connect one end of the resistor to the LED's shorter lead (cathode).
     - Connect the other end of the resistor to the GND (ground) pin on the Arduino.
     - Connect the LED's longer lead (anode) to one of the digital pins on the Arduino (e.g., pin 13).
     - Connect the other end of the LED to the ground (GND) on the Arduino.

v. Upload Arduino Code:
   - Write the Arduino code for LED blinking (as provided earlier) in the Arduino IDE on your computer.
   - Compile the code to check for errors.
   - Upload the compiled code to the Arduino board in Proteus.

- Ensure that the Arduino board is selected in Proteus (click on it), and in the Properties panel, browse for the HEX file generated by the Arduino IDE.

vi. Configure Power Supply: If your Arduino is not USB-powered and requires an external power supply, configure the power supply in Proteus accordingly.

vii. Simulate the Circuit: Start the simulation in Proteus.

- You should see the LED connected to the Arduino blinking according to the code you uploaded.

Observation: Observe the LED's behavior in the simulation to verify that it blinks at the specified intervals.

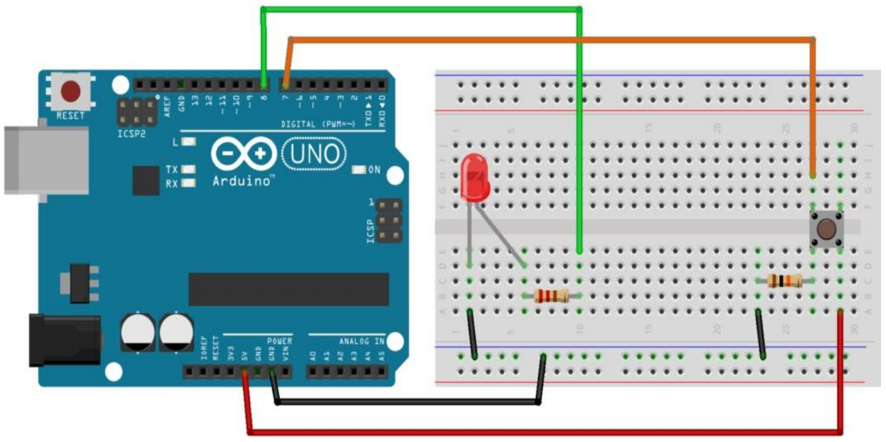
Save and Close: Save your Proteus project for future reference.

Close the simulation when you are finished.

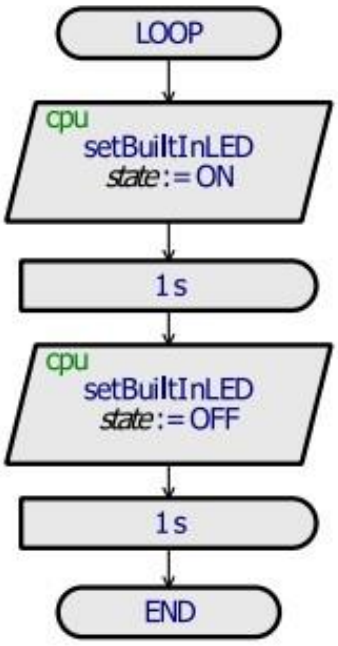
By following these steps, you can simulate LED blinking without a push button using an Arduino in Proteus.

Code-

```
int ledPin = 13; // LED connected to digital pin 13

void setup() {

  pinMode(ledPin, OUTPUT); // Set the LED pin as an output

}

void loop() {

  digitalWrite(ledPin, HIGH); // Turn on the LED

  delay(1000); // Wait for 1 second (1000 milliseconds)
  digitalWrite(ledPin, LOW); // Turn off the LED

  delay(1000); // Wait for 1 second (1000 milliseconds)

}
```



Flowchart-



Practical-12

Aim-Design a Flowchart Weather Station with Arduino.

Solution-

What is Arduino?

Arduino is an open-source electronics platform based on easy-to-use hardware and software. It consists of a microcontroller board and a development environment, allowing users to create interactive electronic projects.

Advantages:

• **Ready to use structure**: Arduino comes in a complete package form that includes the 5V regulator, a burner, an oscillator, a micro-controller, serial communication interface, LED, and headers for the connections.

• **Library of examples**: Arduino has a library of examples present inside the software of Arduino. This library makes it easy to learn how to use the various components of the board.

• **Effortless functions**: During coding of Arduino, you will notice some functions which make life so easy. Another advantage of Arduino is its automatic unit conversion capability.

• **Large community**: There are many forums present on the internet in which people are talking about the Arduino. Engineers, hobbyists and professionals are making their projects through Arduino. You can easily find help about everything.

• **Low cost**: Arduino boards are cheaper than other microcontroller platforms like Raspberry Pi or BeagleBone, making them more affordable for hobbyists and students.

• **Open-source platform**: Being open-source has a plethora of advantages- anyone can access the design and build of the device and make improvements; anyone can use the same hardware design to create their product lineup.

• **Limited processing power**: The limited processing power of Arduino makes it ideal for small-scale projects that do not require complex computations.

Disadvantages:

• **Limited processing power**: The limited processing power of Arduino can be a disadvantage for large-scale projects that require complex computations.

• **No memory safety checks**: Unlike other programming languages like Java or Python, C++ (the language used by Arduino) does not have memory safety checks built-in, which can lead to memory leaks and other issues.

• **Expensive for the CPU power and memory**: Compared to other microcontroller platforms like Raspberry Pi or BeagleBone, Arduino boards can be expensive for their CPU power and memory.

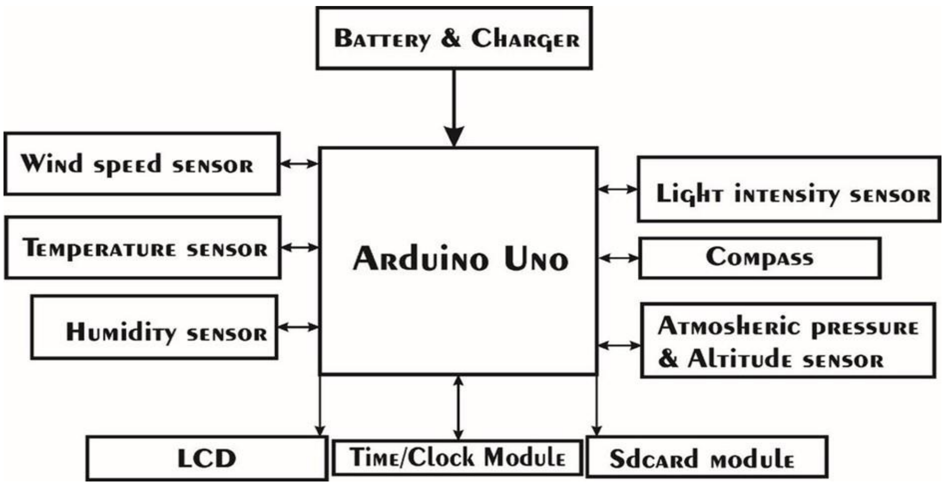
• **Lack of built-in functionality**: Unlike Raspberry Pi or BeagleBone, Arduino does not have built-in functionality like Wi-Fi or Bluetooth connectivity.

• **Cannot run more than one program at the same time**: An Arduino can only run a single sketch or app at a time.

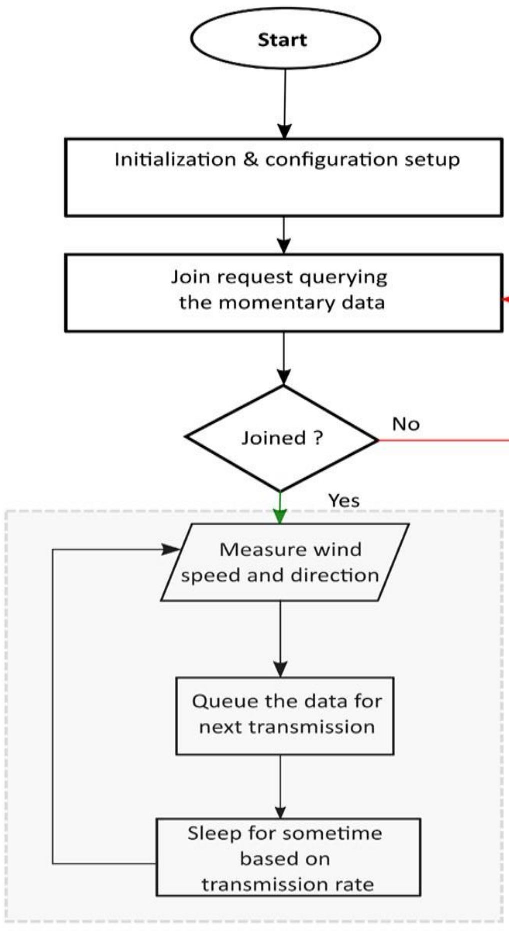
• **No external programmer or power supply needed**: While this is an advantage in some cases, it can also be a disadvantage as it limits the flexibility of the board.

• **Limited memory capacity**: The limited memory capacity of Arduino can be a disadvantage for large-scale projects that require storing large amounts of data.

Flowchart-



`


Start
Initialization & configuration setup
Join request querying the momentary data
Joined ?
No
Yes
Measure wind speed and direction
Queue the data for next transmission
Sleep for sometime based on transmission rate