INDEX:-

PRACTICAL NO: 1

What is Unix Operating System? What are its components? What are its Features?

Unix (officially trademarked as UNIX®, sometimes also written as Unix or Unix® with small caps) is a computer operating system originally developed in 1969 by a group of AT&T employees at Bell Labs including Ken Thompson, Dennis Ritchie and Douglas McIlroy. Today's Unix systems are split into various branches, developed over time by AT&T as well as various commercial vendors and non-profit organizations.

As of 27, the owner of the trademark UNIX® is The Open Group, an industry standards consortium. Only systems fully compliant with and certified to the Single UNIX Specification qualify as "UNIX®" (others are called "Unix system-like" or "Unix-like").

During the late 1970s and early 1980s, Unix's influence in academic circles led to large-scale adoption of Unix (particularly of the BSD variant, originating from the University of California, Berkeley) by commercial startups, the most notable of which is Sun Microsystems. Today, in addition to certified Unix systems, Unix-like operating systems such as Linux and BSD derivatives are commonly encountered.

Sometimes, "traditional Unix" may be used to describe a Unix or an operating system that has the characteristics of either Version 7 Unix or UNIX System V.

Overview of the Features of Unix

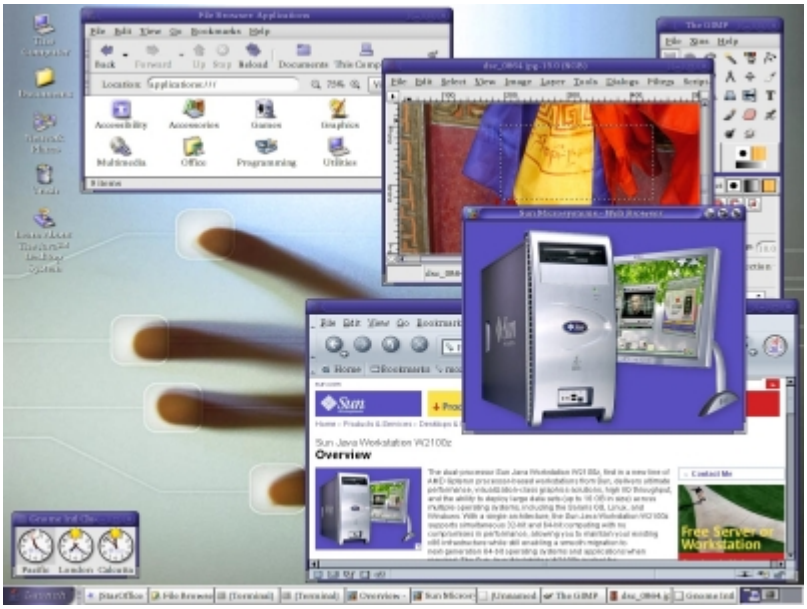
Unix operating systems are widely used in both servers and workstations. The Unix environment and the client-server program model were essential elements in the development of the Internet and the reshaping of computing as centered in networks rather than in individual computers.

Both Unix and the C programming language were developed by AT&T and distributed to government and academic institutions, causing both to be ported to a wider variety of machine families than any other operating system. As a result, Unix became synonymous with "open systems".

Unix was designed to be portable, multi-tasking and multi-user in a time-sharing configuration. Unix systems are characterized by various concepts: the use of plain text for storing data; a hierarchical file system; treating devices and certain types of inter-process communication (IPC) as files; and the use of a large number of small programs that can be strung together through a command line interpreter using pipes, as opposed to using a single monolithic program that includes all of the same functionality. These concepts are known as the Unix philosophy.

Under Unix, the "operating system" consists of many of these utilities along with the master control program, the kernel. The kernel provides services to start and stop programs, handle the file system and other common "low level" tasks that most programs share, and, perhaps most importantly, schedules access to hardware to avoid conflicts if two programs try to access the same resource or device simultaneously. To mediate such

access, the kernel was given special rights on the system, leading to the division between user-space and kernel-space.
The microkernel tried to reverse the growing size of kernels and return to a system in which most tasks were completed by smaller utilities. In an era when a "normal" computer consisted of a hard disk for storage and a data terminal for input and output



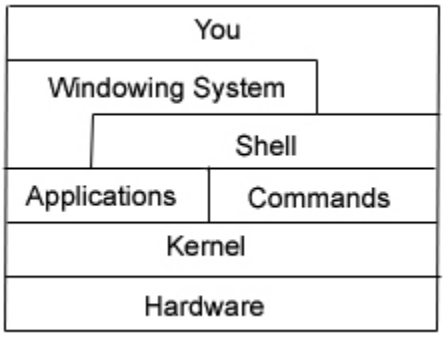
(I/O), the Unix file model worked quite well as most I/O was "linear". However, modern systems include networking and other new devices. Describing a graphical user interface driven by mouse control in an "event driven" fashion didn't work well under the old model. Work on systems supporting these new devices in the 1980s led to facilities for non-blocking I/O, forms of inter-process communications other than just pipes, as well as moving functionality such as network protocols out of the kernel.
Components:

The Unix system is composed of several components that are normally packaged together. By including — in addition to the kernel of an operating system — the development environment, libraries, documents, and the portable, modifiable source-code for all of these components, Unix was a self-contained software system. This was one of the key reasons it emerged as an important teaching and learning tool and has had such a broad influence.
The inclusion of these components did not make the system large — the original V7 UNIX distribution, consisting of copies of all of the compiled binaries plus all of the source code and documentation occupied less than 10Mb, and arrived on a single 9-track magtape. The printed documentation, typeset from the on-line sources, was contained in two volumes.

The names and file system locations of the Unix components has changed substantially across the history of the system. Nonetheless, the V7 implementation is considered by many to have the canonical early structure:

- **Kernel** — source code in /usr/sys, composed of several sub-components:
  - conf — configuration and machine-dependent parts, including boot code
  - dev — device drivers for control of hardware (and some pseudo-hardware)
  - sys — operating system "kernel", handling memory management, process scheduling, system calls, etc.
  - h — header files, defining key structures within the system and important system-specific invariables
- **Development Environment** — Early versions of Unix contained a development environment sufficient to recreate the entire system from source code:
  - cc — C language compiler (first appeared in V3 Unix)
  - as — machine-language assembler for the machine
  - ld — linker, for combining object files
  - lib — object-code libraries (installed in /lib or /usr/lib) libc, the system library with C run-time support, was the primary library, but there have always been additional libraries for such things as mathematical functions (libm) or database access. V7 Unix introduced the first version of the modern "Standard I/O" library stdio as part of the system library. Later implementations increased the number of libraries significantly.
  - make - build manager (introduced in PWB/UNIX), for effectively automating the build process
  - include — header files for software development, defining standard interfaces and system invariants
  - Other languages — V7 Unix contained a Fortran-77 compiler, a programmable arbitrary-precision calculator (bc, dc), and the awk "scripting" language, and later versions and implementations contain many other language compilers and toolsets. Early BSD releases included Pascal tools, and many modern Unix systems also include the GNU Compiler Collection as well as or instead of a proprietary compiler system.
  - Other tools — including an object-code archive manager (ar), symbol-table lister (nm), compiler-development tools (e.g. lex & yacc), and debugging tools.
- **Commands** — Unix makes little distinction between commands (user-level programs) for system operation and maintenance (e.g. cron), commands of general utility (e.g. grep), and more general-purpose applications such as the text formatting and typesetting package. Nonetheless, some major categories are:
  - sh — The "shell" programmable command-line interpreter, the primary user interface on Unix before window systems appeared, and even afterward (within a "command window").
  - Utilities — the core tool kit of the Unix command set, including cp, ls, grep, find and many others. Subcategories include:
    - System utilities — administrative tools such as mkfs, fsck, and many others
    - User utilities — environment management tools such as passwd, kill, and others.

- Document formatting — Unix systems were used from the outset for document preparation and typesetting systems, and included many related programs such as nroff, troff, tbl, eqn, refer, and pic. Some modern Unix systems also include packages such as TeX and GhostScript.



- **Documentation** — Unix was the first operating system to include all of its documentation online in machine-readable form. The documentation included:
  - man — manual pages for each command, library component, system call, header file, etc.

doc — longer documents detailing major subsystems, such as the C language and tradeoff

PRACTICAL NO: 2

Briefly Explain the basic Unix Commands?

Basic commands

There are a number of *Unix* commands which the user should become familiar with from the outset:

`passwd`

This command allows you to change your login *password*. You are prompted to enter your current password, and then prompted (twice) to enter your new password. On *Linux* systems (like ***Magrathea***) passwords should exceed 6 characters in length, and contain at least one non-alphanumeric character (such as **`#`**, **`%`**, **`*`**, **`^`**, **`[`**, or **`@`** etc.)

`cd`

This command, as in DOS, changes directories. You can use **`..`** to represent the **directory above** the current directory. You can use **`~`** to represent your **root directory** (also called your **home** or **top** directory). Example: **`cd maindir`** to move into the **`maindir`** directory, **`cd ..`** to move to the directory above, or **`cd ~`** to move to your root directory.

`pwd`

This command tells you which directory you are currently working in. Your home directory is represented by the tilde **`~`** symbol. To go to your home directory from anywhere, type **`cd ~`**, however typing **`cd`** without the **`~`** also works on *Linux* systems.

`ls`

This gives you a listing of all files in a directory. You can't tell which are files and which are directories

`ls -F`

This shows which files are normal files (they have no special symbols at the end), which are directories (they end in a **`/`** character), which are links (they end in a **@** symbol) and which are executables (they end in a **`*`** character). These special symbols are NOT part of the file name.

`ls -l`

"Long" format. Gives more details about files and directories in the current directory.

`ls -a`

Lists "hidden" files in current directory (those starting with a **`.`** character).

`ls -la`

Options may usually be combined. This particular combination would list both hidden and unhidden files in the long format

---

`mv`

The "move" command is how you rename files. Example: **`mv oldfile.txt newfile.txt`**

`cp`

Allows you to copy one or more files. Example: **`cp myfile.c backup.c`**

`rm`

Deletes a file. BE CAREFUL!! There's no "undelete" command. Example: **`rm janfiles.*`**

`cat`

Sends the contents of a file to *stdout* (usually the display screen). The name comes from "con***`cat`***enate." Example: **`cat index.html`**

`more`

Like **`cat`** but displays a file one page at a time. Example: **`more long_file.txt`**

wc

Counts the number of lines, words, and characters in a file. Example: **wc essay.rtf**

wc

Counts the number of lines in a file. Example: **wc –l essay.rtf**

mkdir

Creates a new directory, located below the present directory. (Use **pwd** first to check where you are!) Example: **mkdir new_dir**

rmdir

Deletes a directory. Example: **rmdir old_dir**

---

man

The **most important** *Unix* command! It displays the manual pages for a chosen *Unix* command. Press **[Enter]** to advance one line, **[Spacebar]** to advance one page, and the **[Q]** key to quit and return to the *Unix* prompt. Example: **man ls**

man -k

Displays all *Unix* commands related to a given keyword. Example: **man -k date** will list all *Unix* commands whose man pages contain a reference to the word `date`

---

date

Shows the current time and date.

---

who

```
Shows the list of number of user online at a time
```

who am i

```
Shows the current user who run this command
```

---

logout

Terminates the current login session, (and returns you to your telnet client, if that is how you established the session originally).

I/O Redirection

<

Input redirection. This allows you to take input from a file rather than *stdin*. Example: **`tr a z <text`**

>

Output redirection. This allows you to send output to a file rather than *stdout*. Example: **`ls -l >listing`**

|

Pipe. This allows you to connect *stdout* from one command with *stdin* of another. Example: **`ls -la | more`**

Unix operating system is warehouse of hundreds of more commands; it is very difficult to use all those commands in simple programming practice. The above mentioned commands are those commands which are most often used by Unix Beginners

PRACTICAL NO: 3

Write a Program to Find Given Pattern in Given List?

Program:

```
clear
echo “Enter some text(Press ctrl+d to stop)”
cat>f1
echo
echo “Enter the pattern”
read pat
if grep “$pat” “f1”
then
echo
echo “Pattern found and shown above”
else
echo “Pattern not found”
fi
```

Explanation

Abstract:

Motive of this Program is to be familiar with the grep command

NAME

grep, egrep, fgrep - print lines matching a pattern

SYNOPSIS

grep [*options*] *PATTERN* [*FILE*...]
grep [*options*] [**-e** *PATTERN* | **-f** *FILE*] [*FILE*...]

DESCRIPTION

Grep searches the named input *FILE*s (or standard input if no files are named, or the file name - is given) for lines containing a match to the given *PATTERN*. By default, **grep** prints the matching lines.

In addition, two variant programs **egrep** and **fgrep** are available. **Egrep** is the same as **grep -E**. **Fgrep** is the same as **grep -F**.

FEW OPTIONS

-A *NUM*, --after-context=*NUM*

Print *NUM* lines of trailing context after matching lines. Places a line containing -- between contiguous groups of matches.

-a, **--text**

Process a binary file as if it were text; this is equivalent to the **--binary-files=text** option.

-B *NUM*, --before-context=*NUM*

Print *NUM* lines of leading context before matching lines. Places a line containing -- between contiguous groups of matches.

-C *NUM*, --context=*NUM*

Print *NUM* lines of output context. Places a line containing -- between contiguous groups of matches.

-b, **--byte-offset**

Print the byte offset within the input file before each line of output.

PRACTICAL NO: 4

Write a Program for merging Contents of Two Files and Place Result in third File?

Program

```
clear
echo “Enter the first source file name.”
read f1
echo “Enter the second source file name.”
read f2
echo
echo “Contents of file $f1 are:”
cat $f1
echo
echo “Contents of file f2 are:”
cat $f2
echo
echo “Enter the name of third file where you want to place the merged data”
read f3
cat $f1 $f2 >$f3
echo
echo”Contents of $f1 and $f2 are merged and sent to $f3”
echo
echo”Contents of file $f3 after merging are:”
cat $f3
```

Explanation

The intent of this Program is to merge contents of two files and then placing result in third file to do so we make use of cat command which help in viewing content of the file After Reading the content of files using read command, with use of command line

cat $f1 $f2 >$f3
we concatenate the content of file f1 and f2 and place the concatenated Result in third File

PRACTICAL NO: 5

Write a Program that being Equivalent to cp Linux command

Program

```
clear
echo”Enter the file name whose contents you want to copy”
read f1
echo
echo”Contents of file $f1 are:”
cat $f1
echo
echo “In which file you want to copy(a non existing file)”
read f2
if[-e $f2 ]
then
echo “File already exists. Please enter a new file name”
read f2
fi
cat $f1> $f2
echo
echo “Now contents of file $f1 are copied to file $f2 and are shown below:”
cat $f2
```

Explanation

Basic intent of this program is to simulate the behavior of cp command used in linux with help of other commands

cp is the command entered in a Unix shell to copy a file from one place to another, possibly on a different filesystem. The original file remains unchanged, and the new file may have the same or a different name.

The following sections refer to the AIX version of cp. Versions of the cp command from other operating systems may have different usage and flags

Usage

To Copy a File to another File

cp [-f] [-H] [-i] [-p][--] SourceFile TargetFile To Copy a File to a Directory

cp [-f] [-H] [-i] [-p] [-r | -R] [--] SourceFile ... TargetDirectory

To Copy a Directory to a Directory

cp [-f] [-H] [-i] [-p] [--] { -r | -R } SourceDirectory ... TargetDirectory

By making use of echo command we try to visualize the effect of cp command

PRACTICAL NO: 6

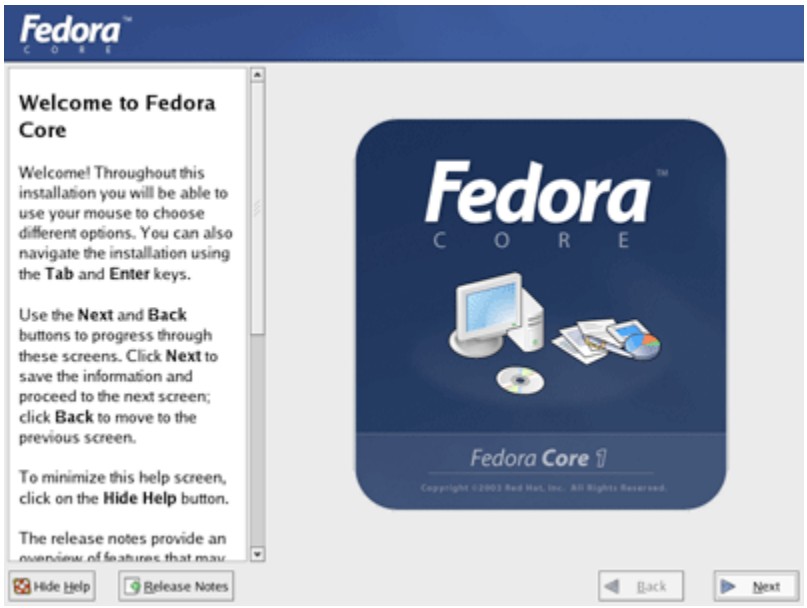
Learning installation and up gradation of the Linux operating system

Installing Fedora Linux

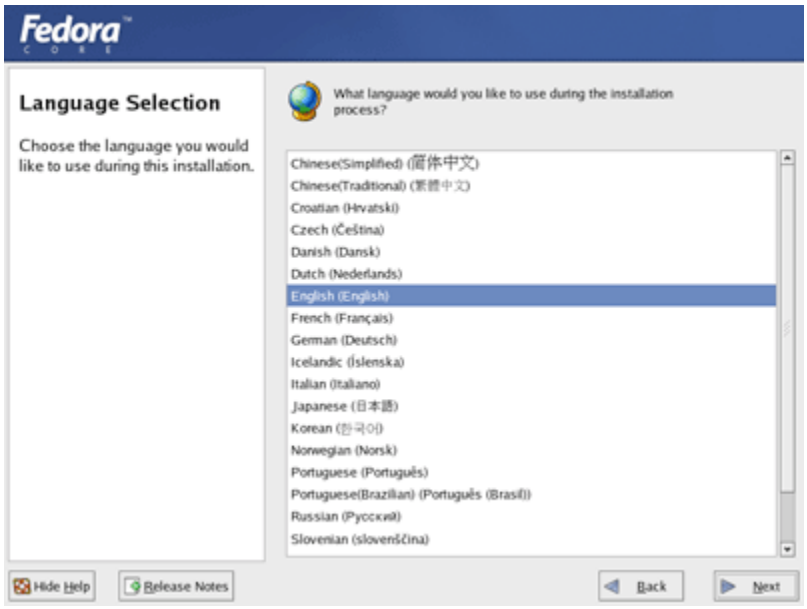
Hardware detection in modern versions of Linux is as good as, if not even better than, modern versions of Windows so there is no need to take note of the hardware of your system. Simply insert CD1 into the CD ROM drive and reboot your computer. It should be set to boot from CD in the BIOS.

The default installation mode is graphic and clicking **[enter]**will select this for you.
You can skip over the media check step unless you suspect your media has problems.

Welcome to Fedora core

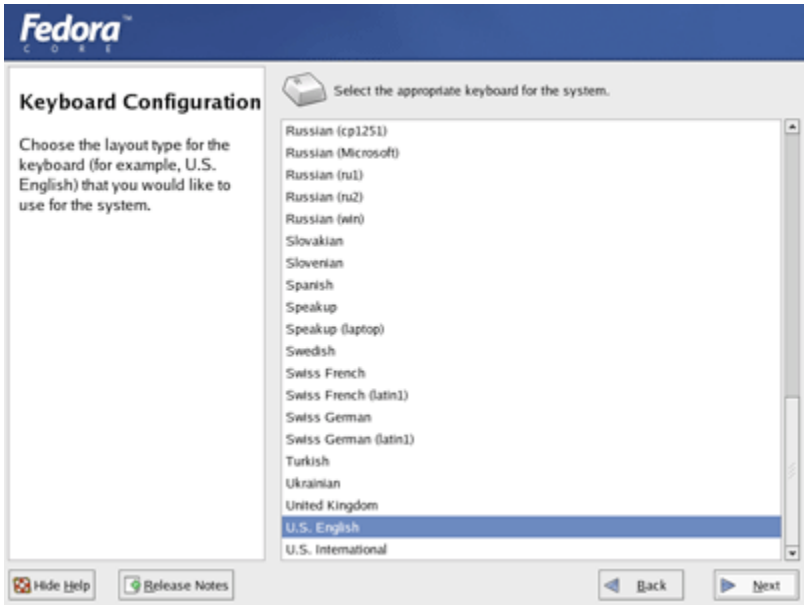


Language choice



Choose English for the installation process.

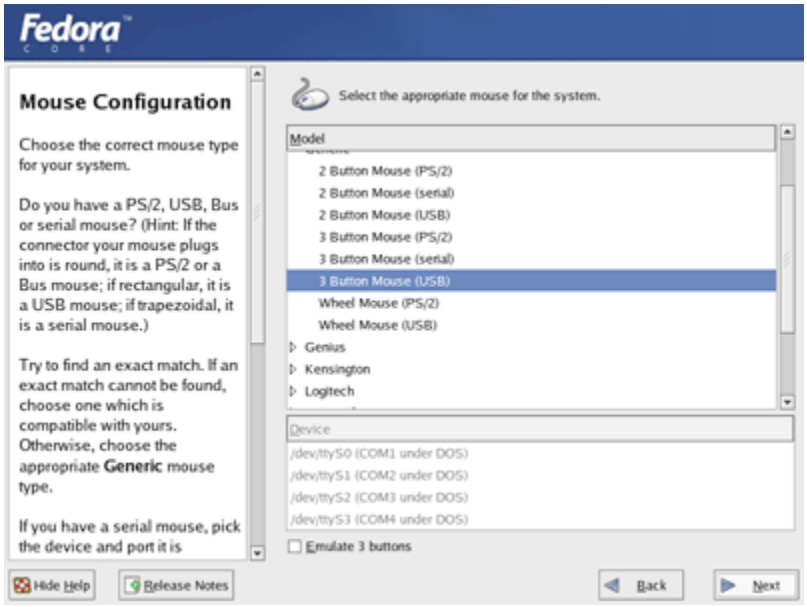
Keyboard configuration



Select US English for the keyboard language.

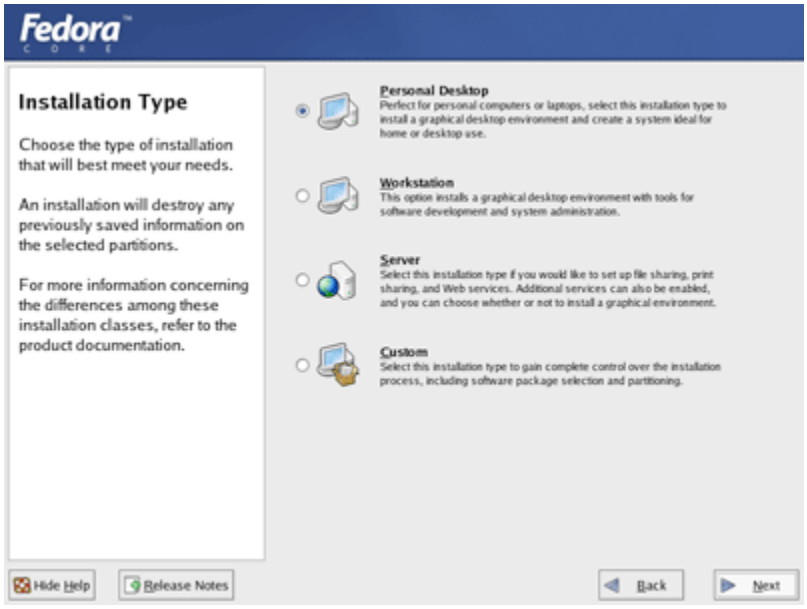
Mouse configuration

Your mouse should have been auto detected. If it wasn't, select the appropriate option.

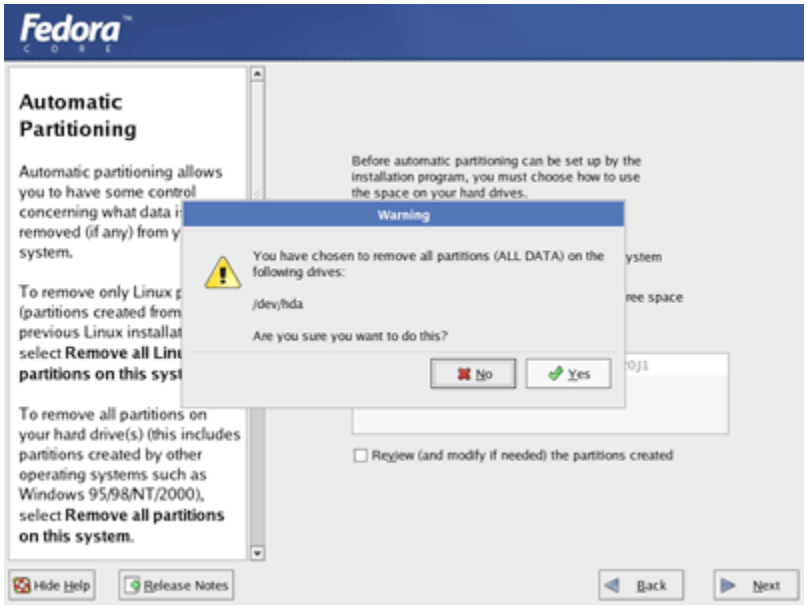


Installation type

The next screen prompts for a choice of installation type. Choosing "Personal Desktop" will install all the software required for the SOE.

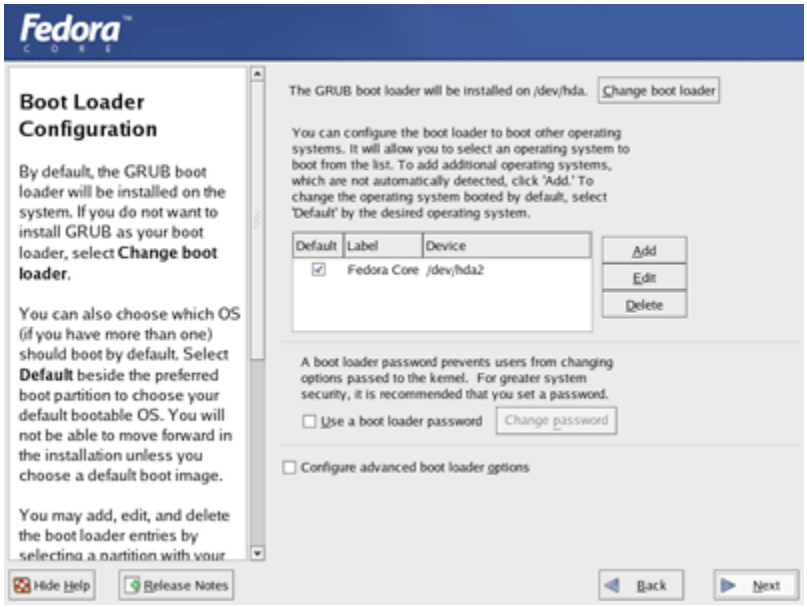


Disk partitioning setup



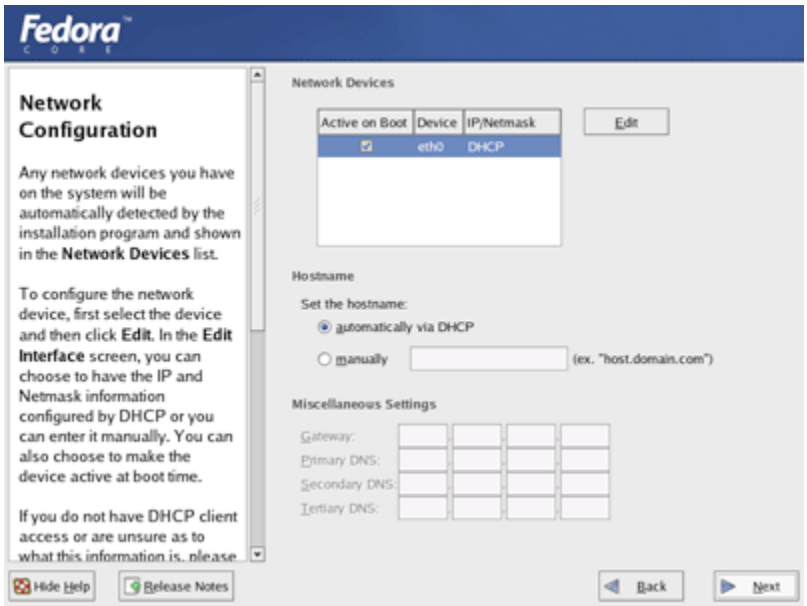
When prompted for disk partitioning setup, select "Automatically partition and "Remove all partitions on this system".

Boot loader configuration



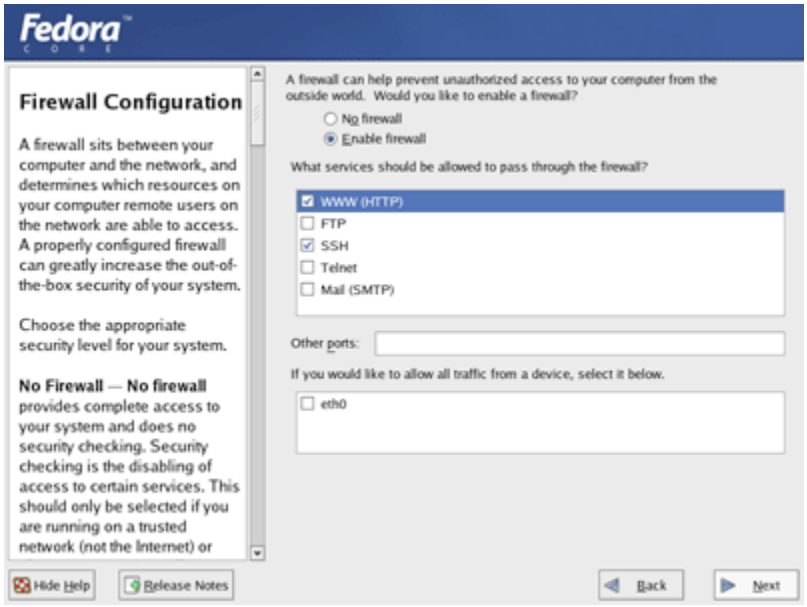
Choose Grub as your boot loader with no custom parameters and no password. Select Install on Master Boot Record if you will only have one operating system on this computer. If you would like to dual boot with windows, NEED A LINK HERE...

Network configuration



If you not using DHCP you will need to enter your network configuration options.

Firewall configuration

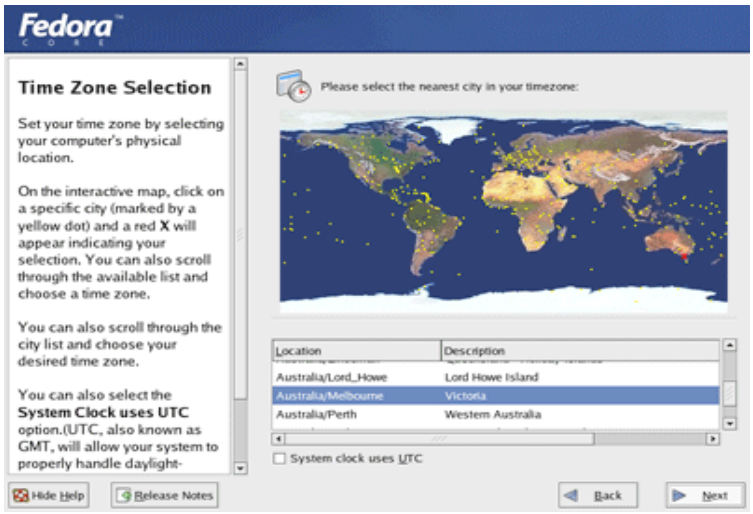


For most desktops uses, you should enable the firewall. All established connections will be allowed so the firewall will not affect your web browsing or email. Unless you wish to run servers, you do not need to allow any services to pass through the firewall.

Additional language support

If you wish to use additional languages, you should enable them here. Select "English (Australian)" and deselect "English (USA)".

Time zone selection



Choose your time zone by clicking on the map or selecting from the text options available.

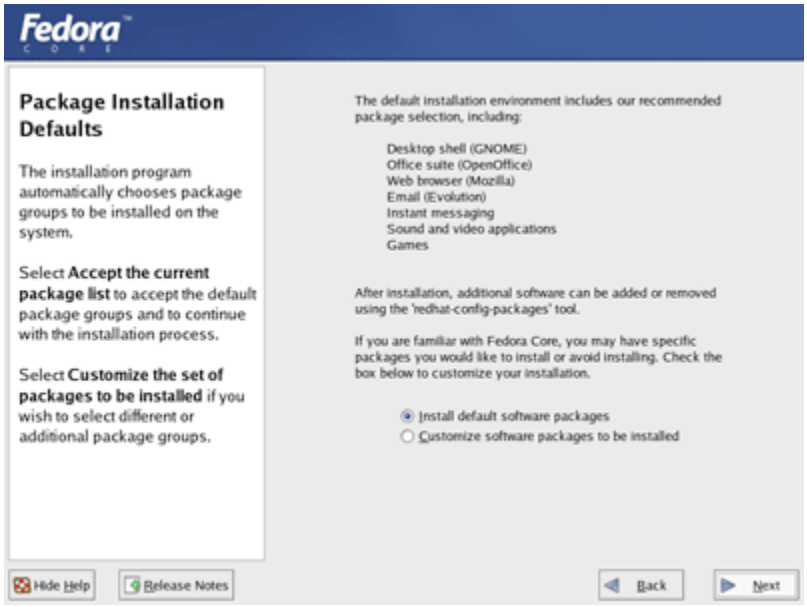
Root password

The root password is the main password for your computer system so make sure you remember it. If you forget the password, it is possible to change by doing the following:

- Reboot, and when you see the boot loader menu, type **e**
- Look for the line that looks similar to the **kernel /vmlinuz-2.4.20 root=/dev/hda2** and scroll down until this line is highlighted and press **e**.

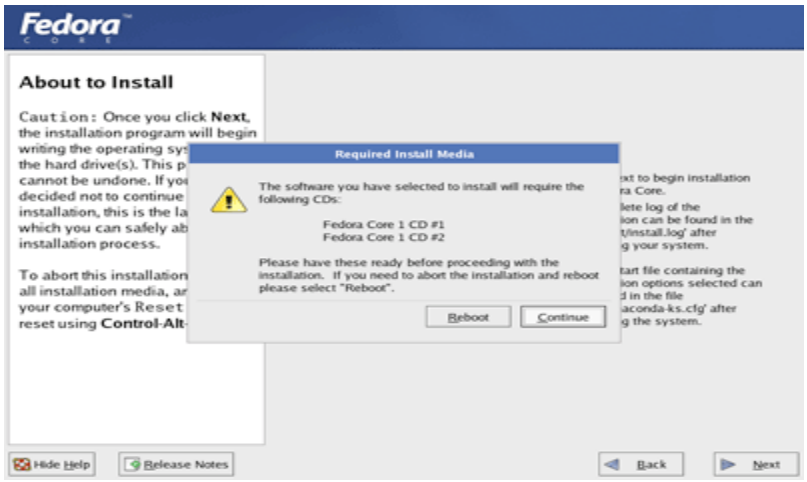
- Press **[space]** and add the word **single** to tell GRUB to boot into single user mode.
- Press [enter] to make the editing change take effect and then press **b**.
- At the **sh-2.05#** prompt, type **passwd root** and enter the new password when prompted.
- Reboot again and you will be able to log in with the new root password.

Package installation defaults



Click "Next" to accept the default installation packages.

About to install



Click "Next" to install the packages. You will need CD1 and CD2 for this installation.
Depending on the speed of your machine, Fedora will take between ten minutes and half an hour to install the packages. When it has finished, reboot your machine.

Welcome to Fedora Linux

On your first boot you will be greeted with several steps where you:

- Accept (or decline) the license agreement.
- Set the date and time (and optionally configure a Network Time Protocol Server).
- Create a standard user account. You should NOT use the root account for everyday use.
- Test your sound card.
- Optionally install extra software.

Once you have finished this process, you will be presented with a screen where you can log in and use your Linux system.

PRACTICAL NO: 7

Familiarization with vi editor

Introduction to the Vi Editor

Vi is the most commonly available screen editor for UNIX. It's the only one you can count on being installed on every Unix system.

Free versions are also available on just about any other operating system, from Atari to Xenix, including Windows and MacOS. There are even graphical versions with tear-off menus and color highlighting! See the links page for more info.

The focus of this page is to get you started which is often the hardest part of learning. More powerful features are left out, but I'll demonstrate some of them at then end.

I've handed out a reference card that I think is a good one. The author's web site is on the links page. I also recommend the book Unix in a Nutshell, which is available at most bookstores. There are books devoted to the just the Vi editor too.

Like most programs, Vi has a large number of commands, but you'll find yourself using only 4 or 5 most of the time.

IMPORTANT THING TO LEARN FIRST:

Editing Modes (or , "why Vi is (supposedly) 'confusing' ")

The most important thing to know about Vi is that it has two modes, Command Mode and Insert Mode.
In Command Mode, you can invoke editing commands, move the cursor, save or exit the file, or enter Insert Mode.
In Insert Mode, you can insert stuff.

Before We Start

We need to create a new file alphabet.html under vi_class directory.

Ensure that you are in your `~/public_html` directory.

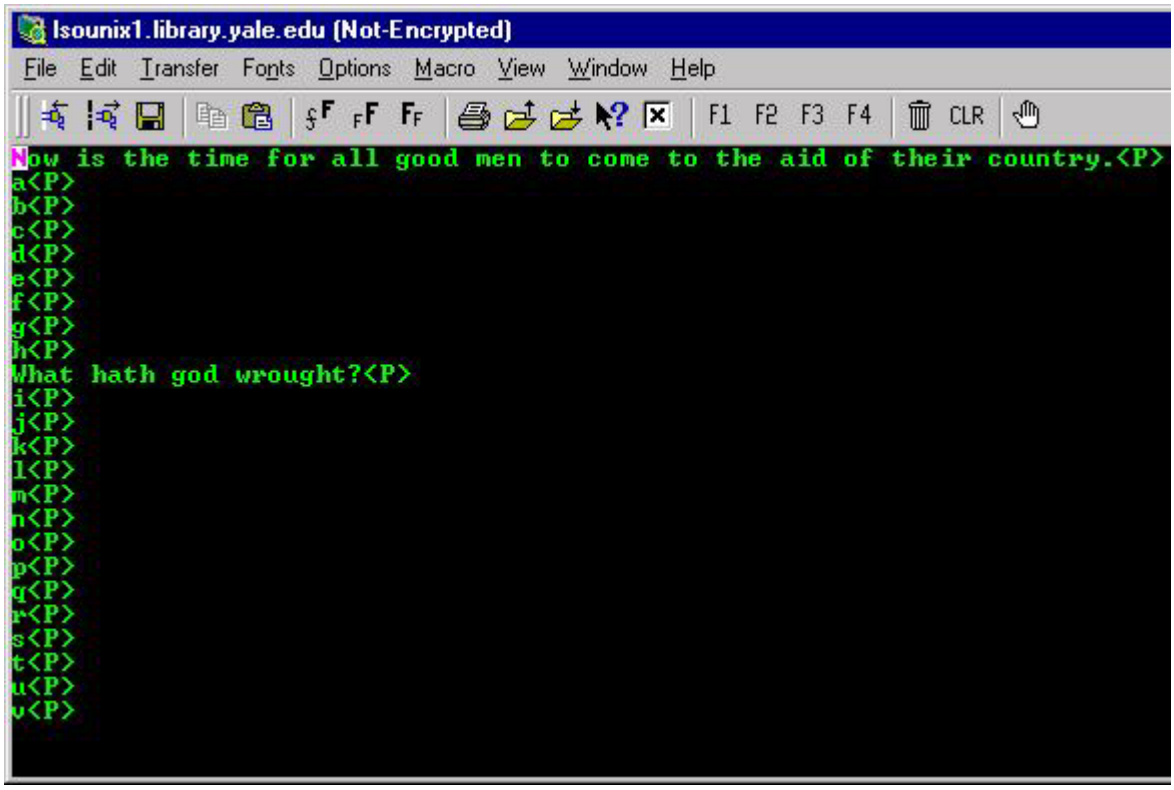
```
$ cd ~/public_html

$ mkdir vi_class

$ cd vi_class

$ vi alphabet.htm
```

Now type `:set showmode` and hit `Enter`.

Editing Your File

When you open a file with vi, you are automatically placed in command mode.

Press the `i` key (insert) to switch to input mode. Input the following text (the alphabet) exactly as presented. Put a <P> tag at the end of each line.
Don't worry if you make a mistake you can't correct. We'll deal with it later.



Return to command mode `Esc` Save your file `:`

Save your file `:w`

Moving by characters

Try these commands out

`h` left a character
`j` down a character
`k` up a character
`l` right a character
or use the arrow keys

Moving by words

`b` back one word (`B` to ignore punctuation)
`w` forward one word (`W` to ignore punctuation)
What would `2w` do?

Moving by lines

`0` beginning of line
`$` end of line
`:number` (`:14` moves to line 14)
`G` end of file
`1G` beginning of file
What would `4G` do?

Moving by screens

`Ctrl-u` scroll up 1/2 screen
`Ctrl-d` scroll down 1/2 screen
`Ctrl-f` scroll forward 1 screen
`Ctrl-b` scroll backward 1 screen
`H` move to the top of the screen
`M` move to the middle of the screen
`L` move to the last line of the screen

Fixing Mistakes and Making Changes

Deleting
`x` delete the character under the cursor
`dw` delete to the end of the current word
`dd` delete the current line

Yanking
`yw` yank a word into the buffer
`yy` yank the current line
What would 2yy do?

Pasting
`p` paste the buffer after current line (or cursor if you didnt yank an entire line)
`P` paste the buffer before current line/cursor
Try yanking a line, moving the cursor several lines down and pasting.

`cw` change to the end of the current word (hit Esc when you're done)
`r` replace one character (you're automatically put back in Command mode)

Try using `dW` and `cW`. What's the difference? Notice the effect when an HTML tag is next to the word.

Operators and Objects

You've probably noticed that `d c y` take an action on an object or objects. `dw` deletes a word, etc.

Try out some others:
`w` word forward
`e` end of word
`$` end of line
`G` end of file
These are just a few....

It's becoming easier now that you understand the logic?

Undoing Changes

`u` undo the last change you made anywhere in the file
`U` undo all recent changes to the current line

Adding text

On the same line:

`a` insert after the cursor
`A` add at end of current line
`i` insert text before cursor
`I` insert at beginning of current line

On a new line:

`o` open a new line after the current
`O` open a new line before the current

Saving and Exiting

`:x` save changes and exit
`:w` save changes
`:q!` quit without saving changes
`:e!` start over without saving changes

Searching

`/pattern` search forward
`?pattern` search backward
`/repeat` previous search forward
`?repeat` previous search backward

Adding Some More Text and Searching

1. Make this the first line of the file: `"The quick brown fox jumps over the lazy dog."<P>`
   How do you get to line 1?
   Instead of inserting the file before the existing line 1, how could you open up another line above?
2. Make this the last line of the file, type: `"Every good boy deserves fudge."<P>`
   How do you get to the last line?
   Using `A` to append to the existing line and then hitting `Enter` is one way to open a new line. What's another?

3. Move your cursor to the top of the file.
4. Search for the word fudge. Your cursor should be on the f.
5. Switch to input mode and add the word chocolate BEFORE fudge.
6. Now search backwards to the line with y. Make it a link to Yahoo.
7. Do the same with h. Make it a link to Hotbot.
8. Save your work without exiting the editor

Deleting, Replacing, Simple Use of Buffers, Recovering from Errors

1. Remove the alphabetic characters `b,e,z` and the <P> tags, but leave the blank lines in their place.
   Use the search command rather than the arrow keys to move about the file.
2. Now delete the blank lines.
3. Now replace f with z. What command do you use to replace a single character?
4. Save your work without leaving the editor.
5. Go to the first line in the file.
   Now delete the first 5 lines of the file with one command.
   Remember that most commands can be applied to multiple objects. You know how to delete 1 line. How do you delete 5?
6. You could delete the next 5 (repeat the most recent command) with a single . (a dot). Try it.
7. Then, undo this second delete. What's the command to do this?
8. Remove the first 10 lines of the file and put them at the end.
   Go to line 1 of the file.
   Delete the first 10 lines with one command. What is it? Now you have 10 lines in the buffer.
   Go to the last line of the file.
   You want to paste the buffer after this line. How?
9. Now you've decided that you've completely screwed up your file and you want to start over (from the point of the most recent save). How can you bail out and start over? (There are several ways).

Miscellaneous and Complex Stuff

Using set to customize the editing session
Options can be set temporarily in an editing session using the set command.

```
:set showmode
:set number
```

Undo them with

```
:set noshowmode
:set nonumber
```

Put them in your .exrc file in your home directory to make them the default. Type set: to see your current settings. Type :set all to display all options. Look at the man page for vi for explanations.

`J` joins lines

Ctrl-T tab (note that this command is run in INSERT mode)

`>>` Shift text right
`<<` Shift text left

`Ctrl-G or :f` shows information about the file you are editing

:r filename Read in a file. You could use this for templates. `:r footer`

:r! command Read in the output of a command. `:r! date`

`:n` Go to the next file (when you have opened multiple files) `vi *.html` for example.

`:p` Previous file

Substitution and filters
We're moving far afield here, but I do want to demonstrate these powerful features

```
:1,$! tr -d '\015'
```

Filter out that pesky ^M in files you edited on your PC.
Let's break down this command
`:1,$` Line addressing : Apply some action to lines 1 thru the end of the file (`$`).
`!` Use a Unix command as a filter: In this case we use the `tr` command to remove `\015` which is the ascii code for `^M`

```
: 1,$ s/seperate/separate/gc
```

On all lines `(1,$)`, substitute`(s)` /existing text/replacement text/ do it globally(not just the first instance on a line)`(g)` and prompt for confirmation`(c)`

```
:14,32 s/$/<P>
```

Put a paragraph tag at the end of lines 14-32. Could've used that one earlier :)

Other applications are limited only by your imagination

What if you wanted to create a macro to do the ^M edit?
You can. Here's how:

Open up the .exrc file in your home directory
Add the following line to map the key combination `*8`:

```
map *8 :1,$! tr -d '\015'
```

If you want the `Enter` key too, you have to do a little gyration:
Type `Ctrl-V` before you type `Enter` when editing your .exrc file. `Ctrl-V` means take the next character literally.
It will look like this:

```
map *8 :1,$! tr -d '\015'^M
```

Just typing ^M won't work.

How about abbreviations in insert mode?

```
:ab hr <HR>
```

Now when you type hr in your text it automatically expands to `<HR>`. Try it

You can make abbreviations and macros permanent by adding them to your .exrc file, or you can make temporary ones by preceding them with a colon `:ab hr <HR>`

You can undo it for the current session with `:unab hr`

PRACTICAL NO: 8

Write a Program to Find Factorial of Number using For Loop

Program

```
#!/bin/bash
echo “enter the number”
read a
for((f=1;a>=1a--))
do
f=”expr $f*$a”
done
echo factorial is: $f
```

Explanation

Factorial of number can be easily calculated using for loop. Factorial of number is visualized as follows
3! Means 3*2*1
4! Means 4*3*2*1
5! Means 5 *4*3* 2*1

Hence by multiplying the number by number decremented by one by its previous value till we get 1 forms the basis of the calculation of the factorial

PRACTICAL NO: 9

Write a Program that demonstrate five basic function of File System

```
clear
echo”OPTIONS ARE:”
echo”1.LIST FILES”
echo”2.CAT FILES”
echo”3.DELETE FILES”
echo”4.MOVE FILES”
echo”5.RENAME FILES”
ans=’y’
while[$ans=’y’]
do
echo”which operation you want to perform”
read ch
case “$ch” in
    1)echo “Enter the directory”
      read directory
       ls-l $directory;;
    2)echo “Enter file name”
      read filename
      echo”CONTENTS OF FILE $filename ARE SHOWN BELOW:”
      echo” “
      cat $filename;;
    3)echo “Enter file name”
      read filename
      rm $filename
    4) echo “Enter file name you want to move:”
       read old
    echo “enter the directory where you want to move:”
    read new
    my $old $new
    echo”FILE $old HAS BEEN MOVED TO THE DIRECTORY $new”;;
    5)echo “Enter file name which u want to rename:”
    read old
    echo “Enter  new name for file:”
    read new
    my $old $new
    echo “NOW $old HAS BEEN RENAMED TO $new”;;
    *) exit
    esac
    echo”Want to repeat:”
    read ans
    done
```

PRACTICAL NO: 10

Write a program to sort number in ascending order

Program

```
clear
echo”Enter elements of array”
read –a arr
no= ${arr[@]}
comp= $(($no-1))
while[$comp-gt 0]
do
index=0
while[ $index-lt $comp]
do
next=$(($index +1))
if[ ${arr[$index]}- gt $ arr[$next]}]
then
temp= ${arr[$index]}
arr[$index]=${arr[$index]}
arr[$next]= $temp
fi
index= $((index+1))
done
comp= $(($comp-1))
done
echo “After Sorting”:
index=0
while[$index-lt $no]
do
echo ${arr[index]}
index= $((index + 1))
done
```

PRACTICAL NO: 11

Write a Program that will output the desired patterns

Program

```
echo "Can you see the following:"

for (( i=1; i<=6; i++ ))
do
   for (( j=1; j<=i;  j++ ))
   do
    echo -n "$i"
   done
   echo ""
done
```

Output

1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
6 6 6 6 6 6

PRACTICAL NO: 12

Write a Program to Find out whether year entered is Leap or not

Program

```
#!/bin/bash
#shell script to find wheather a particular year is a leap year or not
echo -n "Enter the year(yyyy) to find leap year :- "
read year
d=`expr $year % 4`
b=`expr $year % 1`
c=`expr $year % 4`
if [ $d -eq 0 -a $b -ne 0 -o $c -eq 0 ]
then
echo "year is leap year"
else
echo "not leap year"
fi
```

Output

Enter the year 1990
Not leap year

PRACTICAL NO: 13

Script to find out Largest of three numbers

Program

```
if [ $# -ne 3 ]
   then
           echo "$0: number1 number2 number3 are not given" >&2
      exit 1
   fi
   n1=$1
   n2=$2
   n3=$3
   if [ $n1 -gt $n2 ] && [ $n1 -gt $n3 ]
   then
           echo "$n1 is Biggest number"
   elif [ $n2 -gt $n1 ] && [ $n2 -gt $n3 ]
   then
           echo "$n2 is Biggest number"
   elif [ $n3 -gt $n1 ] && [ $n3 -gt $n2 ]
   then
      echo "$n3 is Biggest number"
   elif [ $1 -eq $2 ] && [ $1 -eq $3 ] && [ $2 -eq $3 ]
   then
           echo "All the three numbers are equal"
   else
      echo "I can not figure out which number is bigger"
  fi
```

PRACTICAL NO: 14

Write a program to generate Fibonacci series

Program

```
#!/bin/bash
#shell script to generate fibonacci series
echo "how many fibonacci numbers do u want "
read limit
a=0
b=1
d=1
echo "------------------------------------------------------------"
echo -n $a
echo -n " "
while test $d -le $limit
do
c=`expr ${a} + ${b}`
echo -n $c
echo -n " "
b=$a
a=$c
d=`expr $d + 1`
done
```

Output

How many fibonacci numbers do u want 10
0 1 1 2 3 5 8 13 21 34

PRACTICAL NO: 15

Write a program to check whether given string is palindrome or not

Program

```
#Sun Nov 11 12:06:14 --By Mukesh Dawar
clear
echo -n "enter the name :-"
read name
len=`echo -n $name | wc -c`
echo "Length of the name is :-"$len
while [ $len -gt 0 ]
do
rev=$rev`echo $name | cut -c$len`
len=`expr $len - 1`
done
echo "Reverse of the name is :-"$rev
if [ $name = $rev ]
then echo "It is a palindrome"
else
echo "It is not a palindrome"
fi
```

Output

Enter the name Kanak

It is a palindrome